DESIGN AND VALIDATION OF SOFTWARE MAINTAINABILITY PREDICTION MODELS FOR IMBALANCED DATA USING OBJECT ORIENTED METRICS

By

KUSUM LATA

Roll No.: 2k16/Ph.D./CO/07

Under the guidance of

Dr. Ruchika Malhotra

Associate Professor, Discipline of Software Engineering,

Department of Computer Science & Engineering

Submitted in fulfillment of the requirements of the degree of

Doctor of Philosophy to the

DELHI TECHNOLOGICAL UNIVERSITY

(FORMERLY DELHI COLLEGE OF ENGINEERING)

SHAHBAD DAULATPUR, MAIN BAWANA ROAD, DELHI 110042

2020

Declaration

I, **Kusum Lata**, Ph.D. student Roll No.: 2k16/Ph.D./CO/07, hereby declare that the thesis entitled **"Design and Validation of Software Maintainability Prediction Models for Imbalanced Data using Object Oriented Metrics"** which is being submitted for the award of the degree of Doctor of Philosophy in Computer Science & Engineering, is a record of bonafide research work carried out by me in the Department of Computer Science & Engineering, Delhi Technological University. I further declare that the work presented in the synopsis has not been submitted to any University or Institution for any degree or diploma.

Date :

Place : Delhi

Kusum Lata

Roll No.: 2k16/Ph.D./CO/07

Department Of Computer Science & Engineering,

Delhi Technological University,

Delhi-110042

CERTIFICATE

DELHI TECHNOLOGICAL UNIVERSITY

(Govt. of National Capital Territory of Delhi)

BAWANA ROAD, DELHI-110042

Date: _____

This is to certify that the work embodied in the thesis titled **"Design and Validation of Software Maintainability Prediction Models for Imbalanced Data using Object Oriented Metrics"** has been completed by **Ms. Kusum Lata** Roll No.: 2k16/Ph.D./CO/07 under the guidance of **Dr. Ruchika Malhotra** towards fulfillment of the requirements for the degree of Doctor of Philosophy of Delhi Technological University, Delhi. This work is based on original research and has not been submitted in full or in part for any other diploma or degree of any university.

Supervisor

Dr. RUCHIKA MALHOTRA

Associate Professor

Discipline of Software Engineering

Department of Computer Science & Engineering

Delhi Technological University, Delhi-110042

Acknowledgment

First and foremost, with a profound sense of gratefulness, I would like to express my sincere thanks to my supervisor **Dr. Ruchika Malhotra**. She has been a mentor throughout the journey of this research work. Without her consistent support and continuous encouragement this research would not have been successfully completed. It is such a blessing with an opportunity to carry this research under her supervision. This would be impossible without her from dreaming about Ph.D. to unlocking my potential for Ph.D. I would love to thanks for her priceless and intellectual support, invaluable guidance, and sustained motivation. She has always guided me with patience and her incredible knowledge all the time of this research work. I convey my heartfelt thanks to her. I would remain obliged to her for motivating me throughout the period of this research work.

I offer my heartiest thanks to **Prof. Rajni Jindal**, Head of Department and DRC Chairperson, Department of Computer Science and Engineering, Delhi Technological University, for her precious insights and guidance. I respectfully express my gratitude towards all **faculty members** of the department for their support.

I praise my husband **Mr. Shiv Kumar** very much for his steadfast encouragement and moral support along with patience and understanding. He has always inspired me for hard work. I express my warm love and gratitude to **my parents** for all their unconditional love and encouragement. Without their support, appreciation and driving force it would never have been possible for me to achieve anything in my life.

I am grateful to my loving son, **Tanishq** who is the reason of immense happiness and joy that helped me in coming out of many tired situations and gave me strength during the various stages of my research.

Kusum Lata

Abstract

In today's era, software systems are becoming enormously large and complex. The principal challenge confronted by software practitioners and engineers is that such large and complex software projects have to be developed in a specified short period and satisfying the client's requirements. Developing maintainable software not only saves the effort during the maintenance, but it also results in saving the cost. The problem of predicting the maintainability of software is extensively recognized in the industry and much has been done on how maintainability can be predicted with the help of software metrics. Software maintainability prediction involves the use of various software metrics as predictor variables that are representative of software characteristics such as size, coupling, cohesion and inheritance. Furthermore, we need learning techniques for developing efficient prediction models that are able to determine the software parts having low maintainability or high maintainability. The various elements involved in the development of software maintainability prediction models are required to be analysed and improved to yield efficient software maintainability prediction models.

This thesis analyses and validates the relationship between various Object-Oriented metrics and software maintainability. The empirical analyses were conducted using machine learning, search-based and hybridized techniques with a goal of developing the effective models to predict software maintainability during the initial phases of software development.

For developing the models for the predictive modelling tasks, it is tremendously

essential to look into the data distribution of the underlying datasets. Because the imbalanced distribution of the dataset possess enormous hurdle in the training of the models. This thesis is focused on the improvement of software maintainability by developing models by handling imbalanced data. In the imbalanced data problem addressed in this thesis, the classes with low maintainability are regarded as minority classes while the classes with high maintainability are regarded as the majority classes. The prime contribution of this thesis is the investigation of techniques for developing effective software maintainability prediction models from imbalanced data.

In practice, researchers might not be able to obtain balanced training data with a proportionate number of low maintainability and high maintainability classes. The data resampling techniques have been applied in this thesis for developing software maintainability prediction models to solve the issue of obtaining impractical models yielded from imbalanced data. After obtaining balanced data by applying data resampling techniques, software maintainability prediction models are developed with various machine learning techniques.

Moreover, the use of search-based techniques, a sub-class of machine learning techniques is limited in the domain of maintainability prediction. The search-based techniques are meta heuristic techniques that find an optimal or near-optimal solution amongst a large population of candidate solutions. The research community consistently explores new methods and techniques for developing better and effective prediction models. A promising approach for improvement of existing classifiers is ensemble methodology that aggregates various individual classifiers to provide stable results. In this thesis, we have also investigated the use of ensemble methodology by aggregating them with data resampling techniques. The aggregation of ensemble techniques with data resampling is called as ensemble learners for imbalanced data problem. Another contribution of this thesis is an investigation of hybridized techniques that combine search-based and machine learning techniques into a single approach. The hybridized techniques have been explored after data resampling to

develop effective software maintainability prediction models from imbalanced data.

To contribute to handle imbalanced data in software maintainability prediction, we also propose a novel oversampling technique termed Modified Safe Level Synthetic Minority Oversampling Technique in this thesis.

The data collection for training the prediction model is one of the difficult tasks because in most cases either such data is unavailable or it is difficult to collect. To overcome the limitation of historical data collection, the development of generalized maintainability prediction models with inter-project validation is essential. Thus, the situation in which there is the inadequacy of resources and lack of time to capture training data for the development of maintainability prediction model, inter-project validation can be employed. The applicability of inter-project validation for software maintainability prediction has also been investigated in this thesis.

Contents

List of Tables

List of Figures

List of Publications

Papers Accepted/Published in International Journals

1. Ruchika Malhotra and Kusum Lata, "A Systematic Literature Review on Empirical Studies Towards Prediction of Software Maintainability", *Soft Computing*, 2020 (Impact factor: 2.784). (https://doi.org/10.1007/s00500-020-05005-4)

2. Ruchika Malhotra and Kusum Lata, "An Empirical Study to Investigate the Impact of Data Resampling Techniques on the Performance of Class Maintainability Prediction Models", *Neurocomputing*, 2020 (Impact factor: 4.072). (https://doi.org/10.1016/j.neucom.2020.01.120)

3. Ruchika Malhotra and Kusum Lata, "An Empirical Study on Predictability of Software Maintainability using Imbalanced Data", *Software Quality Journal*, 2020 (Impact factor: 2.141). (https://doi.org/10.1007/s11219-020-09525-y)

4. Ruchika Malhotra and Kusum Lata, "Using Ensembles for Class-Imbalance Problem to Predict Maintainability of Open Source Software", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 27, No. 5, 2020 (Impact factor: 3.275).

Papers Accepted/Published in International Conferences

5. Ruchika Malhotra and Kusum Lata, "Using Hybridized Techniques for Prediction of Software Maintainability using Imbalanced data, *In 10th IEEE International Conference on Cloud Computing, Data Science Engineering*, pp. 787-792, Amity University, Noida, India, 2020.

6. Ruchika Malhotra and Kusum Lata, "Improving Software Maintainability Predictions using Data Oversampling and Hybridized Techniques", *In IEEE World Congress on Computational Intelligence*, pp. 1-7, Glasgow, United Kingdom, 2020.

7. Ruchika Malhotra and Kusum Lata, "Tackling the Imbalanced Data in Software Maintainability Prediction using Ensembles for Class Imbalance Problem" *In International Conference on Recent Trends in Engineering, Technology and Business Management (ICRTETBM)*, Amity University, Noida, 2019.

8. Ruchika Malhotra and Kusum Lata, "On the Application of Cross-Project Validation for Predicting Maintainability of Open Source Software using Machine Learning Techniques", *In 7th IEEE International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 175-181, Amity University, Noida, 2018.

9. Ruchika Malhotra and Kusum Lata, "An Exploratory Study For Predicting Maintenance Effort using Hybridized Techniques", *In Proceedings of the 10th Innovations in Software Engineering Conference (ISEC)*, pp. 26-33, Jaipur, Rajasthan, 2017.

Papers Communicated in International Journals

10. Ruchika Malhotra and Kusum Lata, "Handling Class Imbalance Problem in Software Maintainability Prediction: An Empirical Investigation", *Frontiers of Computer Science*, 2020 (Communicated after major revision).

11. Ruchika Malhotra and Kusum Lata, "Using Hybridized Techniques with Class Imbalance Learning for Software Maintainability Prediction", *Automated Software Engineering*, 2020.

12. Ruchika Malhotra and Kusum Lata, "Modified Safe-Level Synthetic Minority Oversampling Technique for Handling the Imbalanced data in Software Maintainability Prediction", *IEEE Transactions on Reliability*, 2020.

Abbreviations

AdaBoost	**Adaptive Boosting**
ADT	**Abstract Data Types**
Adasyn	**Adaptive Synthetic Sampling**
AR	**Additive Regression**
AHF	**Attribute Hiding Factor**
AIF	**Attribute Inheritance Factor**
AMC	**Average Method Complexity**
ANA	**Average Number of Ancestors**
ANN	**Artificial Neural Network**
ANFIS	**Adaptive Neuro Fuzzy Inference System**
AUC	**Area Under Receiver Operating Characteristic Curve**
Bagging	**Bootstarp Aggregation**
BBN	**Bayesian Belief Network**
BN	**Bayesian Networks**
BNGE	**Batch Nested Generalized Exemplar**
BIOHEL	**Bioinformatics based Hierarchical Evolutionary Learning**
Ca	**Afferent Coupling**
CAM	**Cohesion Among Methods of a Class**
CART	**Classification and Regression Tree**
CBM	**Coupling Between Methods of a Class**
CBO	**Coupling Between Objects**

Ce	**Efferent Coupling**
CFS	**Correlation-based Feature Selection**
C&K	**Chidamber and Kemerer**
CKJM	**Chidamber and Kemerer Java Metrics**
CNN	**Condensed Nearest Neighbor**
CNN-TL	**Condensed Nearest Neighbor with Tomek's Link**
CPM	**Class Purity Maximization**
CPSO	**Constricted Particle Swarm Optimization**
CS	**Class Size metric**
CIS	**Class Interface Size**
DAC	**Data Abstraction Coupling**
DAM	**Data Access Metric**
DCC	**Direct Class Coupling**
DCRS	**Defect Collection and Reporting System**
DIT	**Depth of Inheritance Tree**
DT	**Decision Trees**
DT-GA	**Decision Trees with Genetic Algorithms**
DS	**Decision Stump**
FN	**False Negative**
FP	**False Positive**
EACH	**Exemplar-Aided Constructor of Hyper-rectangles**
GA	**Genetic Algorithm**
GA-Int	**Genetic Algorithm based classifier with Intervalar rules**
GA-ADI	**Genetic Algorithm based classifier with Adaptive Discretization Intervals**
GGA	**Generational Genetic Algorithm for Instance Selection**
GFS	**Genetic Fuzzy System**
GFS-AB	**Genetic Fuzzy System with AdaBoost**

GFS-LB	**Genetic Fuzzy System with LogitBoost**
GFS-MaxLB	**Genetic Fuzzy System with LogitBoost and Single Winner Inference**
GFS-GP	**Fuzzy Learning based on Genetic Programming**
GFS-SP	**GFS-GP with Grammar Operators and Simulated Annealing**
GFS-GPG	**Fuzzy Learning based on Genetic Programming Grammar Operators**
GFS-GCCL	**Fuzzy rule approach based on a Genetic Cooperative Competitive Learning**
GP	**Genetic Programming**
GMDH	**Group Method of Data Handling**
GP	**Genetic Programming**
HB	**Hybridized**
IC	**Inheritance Coupling**
IGA	**Intelligent Genetic Algorithm for Edition**
IH-ICP	**Information flow-based Inheritance Coupling**
IQR	**Inter Quartile Range**
IRBFNN	**Incremental Radial Basis Function Neural Network**
KEEL	**Knowledge Extraction based on Evolutionary Learning**
KNN	**K Nearest Neighbor**
LB	**LogitBoost**
LCOM	**Lack of Cohesion in Methods**
LCC	**loose Class Cohesion**
LCS	**Learning Classifier System**
LDWPSO	**Linear decreasing weight particle swarm optimization**
LDA	**Linear Discriminant Analysis**
LOOCV	**Leave-one-out Cross Validation**
LR	**Logistic Regression**
MARE	**Mean Absolute Relative Error**

MAE	**Mean Absolute Error**
MFA	**Method of Functional Abstraction**
MHF	**Method Hiding Factor**
ML	**Machine Learning**
MLP-BP	**MultiLayer Perceptron with Backpropagation**
MLP-CG	**MultiLayer Perceptron with Conjugate Learning**
MRE	**Magnitude of Relative Error**
MMRE	**Mean Magnitude of Relative Error**
MdMRE	**Median Magnitude of Relative Error**
MOA	**Measure of Aggression**
MOOD	**Metrics for Object-Oriented Design**
MLR	**Multiple Linear Regression**
MARS	**Multivariate Adaptive Regression Splines**
MPC	**Message Pass Coupling**
MPLCS	**Memetic Pittsburgh Learning Classifier System**
MSE	**Mean Square Error**
MSLSMOTE	**Modified Safe Level SMOTE**
NB	**Naive Bayes**
NCL	**Neighborhood Cleaning Rule**
NNEP	**Neural Net Evolutionary Programming**
NOA	**Number of Operations Added by a subclass**
NOC	**Number of Children**
NOH	**Number of Hierarchies**
NOM	**Number of Methods**
NOO	**Number of Operations Overridden**
NIH-ICP	**Non-inheritance information flow-based coupling**
NPM	**Number of Public Methods**

OO	**Object-Oriented**
PBIL	**Population-Based Incremental Learning**
PSO	**Particle Swarm Optimization**
PSOLDA	**Particle Swarm Optimization with Linear Discriminant Analysis**
Pred(q)	**Prediction at Level q**
PUBLIC	**Pruning and Building Integrated in Classification**
QMOOD	**Quality Model for Object-Oriented Design**
QUES	**Quality Evaluation System**
RBFNN	**Radial Basis Function Neural Network**
REP	**Reduces Error Pruning**
RF	**Random Forest**
RFC	**Response For a Class**
RIPPER	**Repeated Incremental Pruning to Produce Error Reduction**
RISE	**Rule Induction from a Set of Exemplars**
RMSE	**Root Mean Square Error**
ROC	**Receiver Operating Characteristic Curve**
RUS	**Random Undersampling**
RQ	**Research Question**
SB	**Search Based**
SE	**Standard Error**
SEM	**Standard Error of the Mean**
SGA	**Steady-State Genetic Algorithm for Instance Selection**
SLIPPER	**Simple Learner with Iterative Pruning to Produce Error Reduction**
SLOC	**Source Lines Of Code**
SSMA	**Steady-State Memetic Algorithm for Instance Selection**
SMOTE	**Synthetic Minority Oversampling Technique**
SafeSMOTE	**Safe Level SMOTE**

SMOTE-TL	**Synthetic minority oversampling technique with Tomek's Link**
SMOTE-ENN	**Synthetic Minority Oversampling Technique with Edited Nearest Neighbor**
SPIDER	**Selective Preprocessing of Imbalanced Data**
SIX	**Specialization Index**
UCS	**Supervised Classifier System**
SVM	**Support Vector Machine**
TN	**True Negative**
TP	**True Positive**
TCC	**Tight Class Cohesion**
UIMS	**User Interface Management System**
WEKA	**Waikato Environment for Knowledge Analysis**
WMC	**Weighted Methods of a Class**
XCS	**X-Classifier System**

Chapter 1

Introduction

1.1 Introduction

With the ever-increasing demands of new software applications based on cutting edge technologies, the complexity of software systems is increasing day by day. The development of maintainable software is becoming a challenge for the software programming community. Maintenance cost is approximated to be 80% of the total cost of software development, and with the increasing complexity of software systems, this cost can be even more [1]. It is very uneconomical and impractical to develop software that cannot be modified easily. So, the development of maintainable software systems substantially reduces future maintenance costs and time [2]. Nowadays, software development has shifted towards Object-Oriented (OO) software methodology because OO software systems have better understandability, and quality [3]. The classes are the fundamental units in OO software development that are expected to have high quality and maintainability [4]. Software classes with low maintainability must be precisely tested in the testing phase to get rid of future faults [5]. Various studies in the literature advocated the prediction of maintainability by establishing

a relation between the OO metrics with software maintainability [6–9]. However, significant problem is still unidentified and unsolved in software maintainability prediction (SMP). This problem is the imbalanced data problem. The software classes with low maintainability requires more revisions in the maintenance period, and the prediction of such classes in the earlier phases is the prime concern of the software developers. However, in most of the maintainability prediction datasets, the classes with low maintainability are rare. If in a maintainability prediction dataset, the data points of low maintainability class are few, it is challenging to train the prediction models so as the predict the unseen data points of both classes (i.e., a class with low maintainability or high maintainability) with reasonable accuracy. Therefore, the focus of this thesis is the improvement of SMP models handling imbalanced data problem. This chapter introduces the basic concepts involved in the thesis and the motivation of the work. First of all, the basic terms like software maintenance, types of software maintenance (Section 1.2), and software maintainability (Section 1.3) are defined. Next, the predictive modeling and its steps for developing a predictive model, and issues in predictive modeling for SMP are discussed in Section 1.4. Section 1.5 discusses the imbalanced data problem. The way to tackle the imbalanced data problem is discussed in Section 1.6. Section 1.7 discusses the literature work. The remainder of the chapter discusses the objectives of the thesis (Section 1.8) and the organization of the thesis is presented in Section 1.9.

1.2 What is Software Maintenance?

Software maintenance is a very wide-ranging activity. It is defined as the activity of making changes in the software system once it has been delivered to the customer and is operational at the customer's site. The definition of IEEE [4] is as follows:

"Software maintenance is the process of changing a software program or object

after delivery to fix errors, improve performance or other attributes, or adapt to a changing environment."

This definition reflects the general idea that software maintenance is a post-deployment activity, which is initiated after the final release of software. It includes all activities that keep the software operational and meet the user's requirements. The life cycle of software is the time from its initial conception until it is expired. In the software life cycle, the various phases include the collection of requirements, software design, implementation, testing and validation of software, and finally software maintenance. All of these phases are related to software development except software maintenance. Software maintenance is typically related to those activities that are carried out after the software development.

Software maintenance as a life cycle phase is defined as *activities required to keep a software system operational and responsive to its users after it is accepted and placed into production*. It is also defined as a set of activities that bring changes in the baseline products. The change is essential to improve the performance of the software consistently. Organizations employ a significant budget for maintenance and preservation of their software systems as they are vital assets for their business progress. As a result, organizations devote more money and resources to maintaining existing systems instead of developing new ones. The costs incurred in changing software are a considerable part of organization's budget [1]. Therefore, as maintaining a software system is pricey, maintenance must be managed and planned properly.

1.2.1 Types of Software Maintenance

Lientz and Swanson [10] categorized software maintenance into three types. These are: *perfective*, *adaptive*, and *corrective*. These types are briefly described as follows:

- *Perfective Maintenance*: To take care of the expanding and/or requirements

of the customers, few activities needed to be performed. These activities include updating, adding, removing, modifying, or extending a few capabilities of the system. These activities are intended to make the source code easier to comprehend and use. Perfective maintenance activities also include restructuring or documentation modifications. Some of these activities are considered optimization that makes the code run faster and uses memory efficiently.

- *Adaptive Maintenance*: Sometimes, the environment in which a software system operates, changes. Therefore, the maintenance activities originated as a consequence of changes in the environment in which software operates is termed as adaptive maintenance. The environmental variations include changes to the hardware devices, operating system, operating system tools like compilers, utility programs, etc.

- *Corrective Maintenance*: There may the errors introduced during the development. Such errors are triggered when the system is in operation. The maintenance activities carried out to fix such errors or bugs are called corrective maintenance. For the proper functioning of the software, such bugs are needed to be corrected immediately. The corrective maintenance usually includes functions that are considered “firefighting“ or “quick fixes“ to allow a system to function correctly.

1.3 What is Software Maintainability?

The software is not static. A software product that works seamlessly, but hard to modify and acclimate to new requirements, will not survive. Software maintainability is a long-term aspect that defines how effortlessly software can evolve or change. The process of changing the software that is delivered to the customer is called software

maintenance, and the ease with which software can be changed to incorporate the changing requirements of the customers is called software maintainability. Software maintainability is one of the important dimensions of software quality.

1.4 What is Predictive Modelling?

The procedure of making, testing, and approving a model to anticipate the probability of an outcome is known as predictive modelling. It uses techniques for the prediction of unknown events irrespective of the occurrence time of the event. For instance, identifying the suspects after the crime has occurred or predicting the crime rate that may occur in the future are a few of the applications of predictive modeling. It is a very useful technique as it helps in getting accurate insight knowledge of any given problem domain and allows the development team to forecast accordingly. The detection theory based on the amount of input data is used to select a model for estimating the probability of any event. One or more classifier is used by the model to anticipate the likelihood of a given dataset belonging to another set. For instance, a model can distinguish whether a given class has defects or not. A model is trained using historical data so that it can be reusable for analyzing the results of other familiar applications.

1.4.1 Steps in Predictive Modelling

The following are the steps that are used in predicting modeling and are shown in Figure 1.1.

1. **Objective of the problem under analysis:** The first step is to define the objectives of the application that needs to be analyzed.

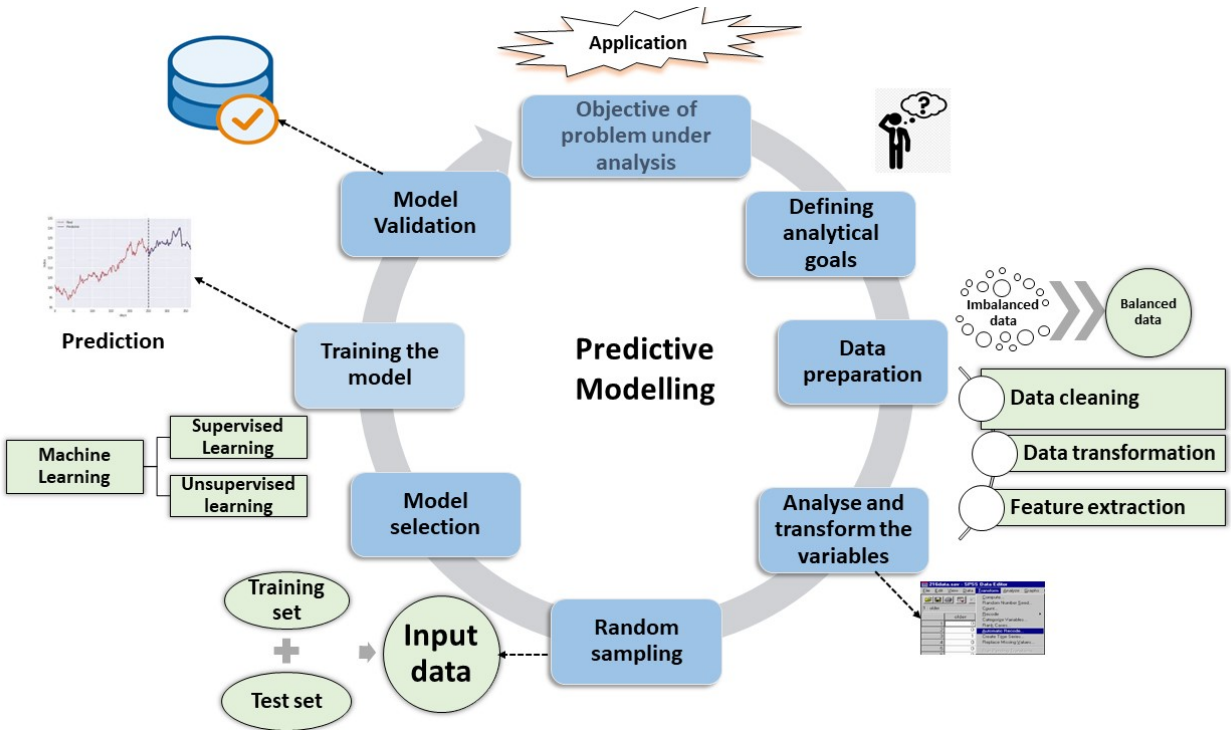


Figure 1.1: Steps in Predictive Modeling

2. **Defining analytical goals:** The goals are finalized after completing the feasibility study of the problem domain.

3. **Data preparation:** In this step, the analysis team understand the data and prepare data for analysis. The predictive model would be efficient if the underlying data is good enough for the analysis.

4. **Analyse and transform the variables:** Here, the null and missing values are dealt with properly and dimensions are reduced using various techniques like principal component analysis, factor analysis, etc.

5. **Random sampling:** The data is divided into training and testing set depending on the ratio selected.

6. **Model selection:** Depending on the goals of the application, models are selected

either supervised or unsupervised.

7. **Training the model:** Models are trained by feeding them with the data prepared at step 3.

8. **Validation:** In the final step, results are validated using inter-validation so that accuracy can be improved.

   As more and more data are being produced, it is important to generate a model that will be able to understand that data thoroughly. Predictive modeling helps the organizations to take suitable decisions.

1.4.2 Predictive Modelling for Software Maintainability

Predictive modeling has its applications in various areas such as finance, transportation, healthcare, social media, etc. where the historical data is used to build predictive models to capture the trends in data and predict future outcomes. Predictive modeling has its application in the software engineering domain too. Over the years several models have been developed by the researchers to predict various software quality attributes such as defect-proneness, change-proneness, development effort, and software maintainability. Managers must identify and predict which parts of the software are likely to possess low maintainability and what effort would be expended in maintaining these parts. SMP models support software project managers in strategizing allocating constraint software resources such as time, cost, and effort. A manager should allocate more resources to software components with low maintainability as these components must be rigorously tested to ensure that changes have been incorporated efficiently and no new errors have been introduced while maintaining these components [11, 12]. Estimating software maintainability in the early phases of software development with the aid of predictive modeling would lead to lower cost and higher quality products

because effective development plans to deal with changes can be prepared well in advance. Predictive modeling for software maintainability results in the development of models to predict maintainability. It involves understanding the relationship between the software system's internal structural characteristics and maintainability. The structural characteristics can be quantified with the help of several software metrics. The software metrics are indicators of different attributes of software like coupling, cohesion, polymorphism, size, etc. The prediction models are developed with various techniques such as statistical, Machine Learning (ML) and Hybridized (HB) that learn from historical software data and predict the maintainability of the future versions of the software product.

1.4.3 Issues in Predictive Modelling for Software Maintainability

Li&Henry [5], in 1993, defined software maintainability in the form of the lines of source code changed during the period of maintenance to correct faults. They advocated that software maintainability has a strong correlation with OO metrics describing various software characteristics such as inheritance, coupling, and cohesion. Later various researchers [7, 13–16] measured the maintainability in the form of the lines of source code changed during the maintenance. They developed software maintainability models to predict maintainability in the form of number of lines of code changes. The more the changes encountered in a class in the maintenance phase, the lower would be the class's maintainability and vice versa. Thus, these studies tried to predict software maintainability in terms of code lines changed, i.e., a numeric value. The struggle with these studies was difficulty in predicting a numeric value to find ideal results.

Furthermore, most of the time, it is superfluous to predict the exact number of code lines changed in a class or a module. Subsequently, research has bowed towards

binary classification, and researchers have come up with models for predicting change prone classes, i.e., the classes that have the probability of changing in the future [11, 12, 17–21]. However, the class in which there is a change of one line of code changed or 1000 code lines are changed, predicted as change-prone classes in these studies. But software practitioners are more interested in those classes that have the probability of a greater number of code lines changes, i.e., the classes with low maintainability.

This research work aims to develop maintainability prediction models that predict low and high maintainability classes. SMP is regarded as a binary classification problem in this study. It is essential that for effective training of the model, the dataset consists of low maintainability and high maintainability classes. The low maintainability requires more revisions in the maintenance period, and the prediction of such classes in the earlier phases is the prime concern of the software developers. However, in most of the maintainability prediction datasets, the classes with low maintainability are rare, and the datasets are imbalanced. If in a maintainability prediction dataset, the data points of low maintainability class are few, it is very challenging to train the prediction models to predict the unseen data points of both classes (i.e., a class with low maintainability and high maintainability) with reasonable accuracy.

In this research work, the effort is devoted to improve maintainability prediction performance by dealing with imbalanced data as no study has addressed this problem in the area of SMP. This study preprocessed the imbalanced dataset carefully with data resampling techniques to develop effective maintainability prediction models. Also, the use of the statistical test has been decidedly less evident in past studies. The study applies statistical analyses to strengthen the results.

1.5 What is Imbalanced Data Problem?

Imbalanced learning or imbalanced data problem is well recognized in various real-life problems like predicting fraud credit card transactions [22], detection of fake website [23], sentiment analysis [24] etc. In an imbalanced dataset, there is a huge difference in terms of number of data points of different classes. This thesis work deals with binary class imbalanced data problem where the training dataset has only two classes namely low maintainability and high maintainability. We mostly find that low maintainability classes are less in number in datasets as software datasets follow the Pareto principle [12]. According to this principle, the major changes in a dataset originate from only 20% of software modules or classes [12]. Therefore, it is essential to build SMP models to recognize low maintainability classes accurately. The low maintainability classes need to be well-designed and meticulously tested to have quality software. An efficient model should be able to detect both low and high maintainability classes with good accuracy. But most often, with the imbalanced datasets, high maintainability classes are detected with high accuracy but low maintainability classes are not that much correctly recognized. For imbalanced datasets, SMP models have lower accuracy rates for low maintainability classes. In the case of a highly imbalanced dataset, the lower recognition rate of the model for low maintainability classes is overlooked and the models show good overall accuracy. These models are biased but are still labeled as decent models. But this situation is relatively unfavorable and may lead to incorrect decisions leading to heavy loss for a software organization. Suppose in a dataset there are 1000 data points. Each data point is comprised of OO metrics and low or high maintainability as the class labels. The aim is to develop an efficient and unbiased model that detects both low and high maintainability classes accurately. Suppose, the training dataset is imbalanced with only 10% data points of low maintainability classes

and 90% of data points belong to high maintainability class. A prediction model trained on such a dataset would give accurate predictions for high maintainability class (nearly 100%) but very low accuracy for low maintainability classes generally in the range from 0 to 10% [25]. The accuracy of 10% of low maintainability classes means 100 out of 1000 low maintainability classes are predicted correctly. This low accuracy for low maintainability class is not acceptable. It would lead to a bad quality software product because low maintainability classes need sufficient resources so that these classes can be effectually designed, verified, and tested i.e., these classes should be appropriately handled to avoid the existence of defects in forthcoming software versions. The lower accuracy of SMP models for such classes means ignorance of these classes. This would badly impact software quality. Therefore, it is critical to effectively handle the imbalanced data problem in SMP.

1.6 Tackling with the Imbalanced Data Problem

The solutions to handle the problem of imbalanced data have been categorized into two types namely data level and algorithm level solutions. The data level solutions involve the modifications of the imbalanced datasets to alleviate the imbalanced data problem. The algorithm level solutions involve modifications in the learning procedures of the learning techniques to eliminate their bias towards learning majority class data points. This section presents a brief explanation of these solutions.

1.6.1 Data Level Techniques

The data level techniques modify the imbalanced training data to make it appropriate for the learning algorithm [26]. The data level techniques make a balance in the data distribution of different classes by removing or adding more data points in the dataset.

The data level techniques are classified into two types namely undersampling and oversampling. The undersampling techniques tend to balance the datasets by randomly dropping the data points of the majority class The oversampling techniques add more data points belonging to the minority class into the dataset to make minority data points proportionate with the majority class. The oversampling and undersampling techniques are regarded as data resampling techniques [27, 28].

1.6.2 Algorithm Level Techniques

With the imbalanced datasets, the learning techniques strongly bias their learning towards the majority class data points [26, 29]. Unlike data level techniques, algorithm level techniques modify the learning procedure of the learning techniques to lessen their favoritism towards learning majority class data points. The cost-sensitive learning mechanism is the popular technique for this in which the higher cost is assigned to the underrepresented data points i.e., data points of the minority class. Assigning a higher cost to the minority data points during the learning boosts their weightage in the learning process. To alleviate the imbalanced data problem, hybrid techniques have also been proposed in the literature. These techniques combine the data level techniques with the algorithmic level techniques thereby generating robust learners by learning from the imbalanced data [26].

1.7 Literature Survey

As discussed, software maintenance is an important activity that should be planned and monitored to get good quality software and ensure customer satisfaction. Researchers have been working hard in this direction to optimize the processes and other facets associated with software maintenance. A comprehensive study of the existing literature

is essential to recognize the research gaps and get the motivation to further work in this direction.

In this section, software metrics used in the literature to predict software maintainability are discussed. Also, various models that have been developed in the research for predicting software maintainability are discussed in this section.

1.7.1 Software Metrics

The study by Swanson [30] advocated that software maintainability is imperative as a significant proportion of the total budget is spent on this activity. He suggested that if a software product or component is not maintainable, it cannot accommodate changes flawlessly, resulting in the loss of many business opportunities. Further, various researchers, Martin and McClure [31], Nosek and Palvia [32] identified problems in the process of software maintenance, and suggested various metrics such as Halstead [33] and McCabe's cyclomatic complexity [34]. These metrics are intended to measure the quality of the source code of software systems developed in procedural programming languages. Rombach [35] claimed that all the useful metrics for procedural language could not be practical for OO languages as the procedural programming focuses primarily on functional decomposition. In contrast, the OO programming paradigm emphasizes the identification of classes, objects, and characteristics of classes. Also, software practitioners started giving importance to software design instead of the code. Thus, various metrics were proposed by the researchers to capture and quality the different design aspects of OO paradigms such as inheritance, encapsulation, cohesion, polymorphism, etc. Further, with the help of empirical investigations, a strong correlation between software design metrics and maintainability was recognized, and various metrics suites were proposed for the OO paradigm such as Chidamber and Kamerer (C&K) metric suite [36], Li&Henry metric

suite [5], Wei Li metric suite [37], Lorenz and Kidd metric suite [38], Metrics for Object-Oriented Design (MOOD) [39], etc. A brief description of metrics included under these metric suites are given as follows:

- Chidamber and Kemerer [36] presented a metrics suite which contains six OO metrics namely Response For a Class (RFC), Depth of Inheritance Tree (DIT), Coupling Between Object classes (CBO), Weight Methods per Class (WMC), Lack of Cohesion in Methods (LCOM) metrics and Number Of Children (NOC).

- Lorenz and Kidd [38] have given the OO metric suite containing the following OO metrics; Number of Operations Overridden (NOO), Class Size metric (CS), Specialization Index (SIX), Number of Operations Added (NOA), and Number of Public Methods (NPM).

- Li&Henry [5] suggested an OO metric suite containing the following metrics; Number of Methods (NOM), Message Pass Coupling (MPC), Data Abstraction Coupling, Size metrics namely SIZE1 and SIZE2.

- Briand et al. [40] proposed a metric suite containing eighteen OO metrics. These metrics under this suite capture types of interactions among the classes. These metrics guide software practitioners on the kind of coupling that affects the cost of maintenance cost. In addition to various coupling metrics proposed by Briand, two more coupling metrics, Afferent Coupling (Ca) and Efferent Coupling (Ce) were given by Martin.

- Bansiya and Davis [41] recommended the Quality Model for Object-Oriented Design (QMOOD) metrics suite. The following are the constituents of this metric suite; Number of Polymorphic Methods (NOP), Design Size in Classes (DSC), Number of Hierarchies (NOH), Average Number of Ancestors (ANA), Data Access Metric (DAM), Direct Class Coupling (DCC), Cohesion among

Methods of a Class (CAM), Measure of Aggression (MOA), Method of Functional Abstraction (MFA), Class Interface Size (CIS).

Tang et al. [42] proposed Coupling Between Methods of a Class (CBM), Number of Object/Memory Allocation (NOMA), Average Method Complexity (AMC), and Inheritance Coupling (IC).

- To distinguish between the non-inheritance-based coupling from inheritance-based coupling, Lee et al. [43] proposed the following metrics; information flow-based inheritance coupling (IH-ICP) and Non-inheritance information flow-based coupling (NIH-ICP). The information flow-based coupling (ICP) metric is the sum of NIH-ICP, and IH-ICP was included in this metric suite.

- Bieman and Kang [44] have given two cohesion based metrics, namely loose class cohesion (LCC), and tight class cohesion (TCC) to quantify cohesion in the classes.

1.7.2 Software Maintainability Prediction

The relationship between software metrics and maintainability is well established in the literature. Several models have been proposed in the literature to predict software maintainability. Chidamber and Kamerer [36] in 1991, language-independent OO metrics to predict software maintainability. Li and Henry [5] added two more metrics into the existing C&K metrics suite to quantify different types of coupling. To measure the coupling through message passing, Message Passing Coupling (MPC) metrics were added and to measure the coupling through Abstract Data Types (ADT), Data Abstraction Coupling (DAC), and added to C&K metrics.

Li and Henry [5] also observed that C&K metric suite does not take into account the structural quality of the code. Therefore, later two metrics more metrics capturing

the same were added into this metrics suite. These metrics were SIZE1, which measures the line of code for software, and SIZE2 for counting the total number of attributes and methods in a class. The Li and Henry and C&K metric suites were validated in many empirical studies to predict software maintainability.

The non-availability of datasets for empirical validations for software maintainability was the big hurdle. The datasets of two proprietary software systems encompassing of User Interface Management System (UIMS) and Quality Evaluation System (QUES) were made public by Li and Henry and termed as Li &Henry [5] dataset. Later, many research studies emerged in SMP validating the Li&Henry dataset with the application of various techniques and various researchers used this dataset in their studies. Few of the studies that validated Li&Henry dataset are Aggarwal et al. [9] , Elish and Elish [15] , Dagpinar and Jahnke [6], Koten and Gray [13], Tang et al. [42] , Malhotra and Chug [45], Kaur et al. [46], Thwin and Quah [14], and Zhou and Leung [7]. These studies advocated that OO metrics can be used to measure the structural quality of code in general and predict software maintainability.

Li and Henry tried [5] linear regression model using UIMS and QUES dataset for appraising the relationship between C&K metrics and maintainability. The results of the study showed that the C&K metrics of UIMS and QUES datasets have a significantly high correlation with software maintainability. The study by Aggarwal et al. [9] advocated that Artificial Neural Network (ANN) is a very suitable technique for SMP. In this study, Li&Henry dataset was used and it was found that using ANN, the SMP model achieved the accuracy up to 72%.

Basili et al. [47] conducted empirical validations on C&K metrics for predicting software maintainability. The study used eight medium-sized software systems written in the C++ language. The C&K metrics and traditional metrics of the datasets were extracted and used for empirical validations. The results of the study advocated that OO metrics are good predictors of software maintainability.

Dagpinar and Jhanke [6] recommended that OO metrics should be used for developing SMP models. The study also suggested that direct coupling metric and size metric have a significant impact on software maintainability instead of cohesion, inheritance, and indirect coupling. Elish and Elish [15] employed gradient boosting in classification and regression tree (TreeNet) to improvise it, which was then used to build SMP models. The performance of TreeNet is compared against Support Vector Machine (SVM), Neural Network (NN), Regression Tree (RT), and Multivariate Adaptive Regression Splines (MARS). In this study, TreeNet was found better in its predictions than SVM, NN, MARS, and RT on UIMS and QUES datasets. The results were analyzed using various prediction accuracy measures such as Magnitude of Relative Error (MRE), Mean Magnitude of Relative Error (MMRE), Mean Absolute Relative Error (MARE), and Prediction accuracy with less than 25% error known as Pred(0.25).

Thwin and Quah [14] used OO metrics extracted from two commercial software systems to build SMP models. The principal component analysis on the extracted OO metrics has been carried out in this study. The NN is used to develop SMP models in this study.

The study by Fioravanti and Nesi [48] presented a metric analysis to identify which metrics would be better predictors of OO software maintainability. Kaur et al. [46] conducted the study using UIMS and QUES datasets with hold out cross-validation. The study applied Adaptive Neuro-Fuzzy Inference System (ANFIS) technique. The results of the study advocated that ANFIS technique gives a better predictive performance as compared to previous studies for developing an efficient SMP model.

Koten and Gray [13] proposed Bayesian Belief Network (BBN) using ten-fold cross-validation on the Li&Henry dataset and found it to be significantly better in terms of prediction accuracy. In this study, for the UIMS dataset, BBN model outperformed

compared to the RT well as the multiple linear regression model. For the QUES dataset, even though BBN was not as good as it was for the UIMS dataset, but still reasonably accuracy was achieved for QUES compared to regression models. The study concluded that the performance of BBN based model is dependent on the characteristics of datasets.

Malhotra and Chug [45] examined the performance of SMP model developed with the application of Group Method of Data Handling (GMDH) technique on the UIMS and QUES dataset. They analyzed the performance of the models in terms of MRE, MMRE, MARE, and Pred(0.25) performance metrics. The outcomes of the study exhibited that the GMDH has the competence to apprehend the design characteristics of datasets using C&K metrics, due to which the results were found to be better.

Dallal [49] examined 19 OO metrics on software maintainability. The prediction models are constructed using statistical techniques, univariate, and multivariate regression. The study results revealed that cohesion and size metrics have higher involvement in SMP than coupling metrics. Tang et al. [42] empirically validated C&K metrics for investigating the correlation between OO design metrics and software maintainability.

The studies by Jin and Liu [50] and Misra [51] developed SMP models by extracting the datasets developed from the student's projects. Jin and Liu [50] collected datasets containing OO metrics from the software projects developed by the students. SVM with ten-fold cross-validation was used in this study for predicting maintenance effort. This study also advocated that there is a significantly high correlation between C&K metrics and software maintainability. The study by Misra [51] used the datasets collected from C++ programs and developed SMP models using Linear Regression. He concluded that the two most important measures should be kept in mind while coding is that functions (modules) should not be more than two screens long. He found that the AMC metric is a significant predictor for determining software maintainability.

The software maintainability can be measured only once the software is in the operational phase but can be estimated early in the software development. In this regard, as suggested by Jorgensen [52] and Lucia et al. [53] the essential technique to take care of this issue are by developing prediction models using OO metrics that can be deployed during the early phases of the software project development. According to [52, 53], the prediction of software maintenance early during software development reduces the future maintenance cost and effort because software developers can improve the design or code by detecting the determinants of software maintainability. It also gives cautionary signs to project managers well in advance with evidence for making effective plans regarding the resource allocations and taking curative actions using their valued resources more cautiously.

In open-source software systems, practitioners worldwide are allowed to change, inflate, and redistribute the newly developed version without any pre-requisite of licensing [54]. The open-source software systems are changed continuously by many software developers to enhance their functioning and practicality. The determination of the maintainability of open-source software systems is challenging due to the lack of technical support and the absence of documentation. In the literature, there is an inadequate number of studies on witnessing the maintainability of open-source software systems. Few studies Zhou and Xu [55], Zhang et al. [56], Malhotra and Chug [57] developed models to predict the maintainability of open-source software systems.

Ramil et al. [58] surveyed many empirical studies that have taken datasets from open-source software systems. However, all those studies oriented towards judging the software evolution based on the characteristics instead of developing the prediction models. The datasets used in proprietary software also have certain constraints on the generalizability of results because each system has been developed in different programming languages with different environment settings.

Wang et al. [59] proposed a fuzzy network framework to predict software maintainability using two widely used commercial datasets. The study advocated that the proposed framework improves the accuracy of software maintainability models compared to standard fuzzy-based models.

Kumar et al. [16] used class level software metrics with three different types of neural networks to train software maintainability models. The genetic algorithm with a gradient descent approach is used to find the optimal weights of neural networks. Schnappinger et al. [60] extracted software metrics using static analysis tools and predicted software maintainability using diverse ML techniques. The majority of the SMP studies have practiced ML and statistical techniques for developing SMP models.

The evolutionary algorithms are the set of algorithms inspired by the metaphor of natural biological evolution such as Ant-Colony optimization, Bees Algorithms, Cuckoo Algorithms, and Particle Swarm Optimization, etc. [61–64]. Certain operators are applied to potential solutions to produce better approximations in these algorithms. Each time, at each generation various operators such as selection, recombination, mutation, migration, locality, and neighborhood are applied to produce the next generation, and each [63] individual solution of the next generation is compared against the survival of the fittest, and the weak solutions are discarded [62]. On repeating this process, again and again, it leads to optimized and better solutions. In the next decades, such evolutionary and Search-Based (SB) techniques were being widely used for solving many problems in various areas of software engineering, like the prediction of development effort estimation [65], prediction of maintenance effort [66], prediction of preventive maintenance [67]. However, the use of these SB techniques for SMP is found to be immensely limited.

1.7.3 Software Quality Predictive Modelling using Imbalanced Data

For predictive modeling in software engineering, the imbalanced data problem (i.e., one class has more number of data points compared to another class) is encountered prediction of classes that are likely to be change-prone and defective. This problem is solved in different ways to uplift the performance of the predictive models. Choeikiwong and Vateekul [68] proposed an algorithm level solution to the imbalanced data problem for software fault prediction. They implemented a classifier in which the separation hyperplane of SVM was adjusted to cut down the bias from the dominant class. Gao et al. [69] examined four different scenarios of feature selection and data sampling to boost the predictive capability of defect prediction models developed with imbalanced datasets. This study confirmed that feature selection on resampled data improves the predictive capability of the models. Laradji et al. [70] proposed an Average Probability Ensemble (APE) incorporating several base classifiers to cope up with the imbalanced data problem. To further improve the prediction capability, feature selection was combined with APE in this study. Siers and Islam [71] proposed a cost-sensitive classifier, which was an ensemble of the decision trees to tackle the problem of imabalanced data. Pelayo and Dick [72] investigated the synthetic minority oversampling technique, which balances the proportion of the defective and non-defective modules. The paper by Sun et al. [73] used ensemble and coding schemes to handle the imbalance data problem for predicting defective classes. The study first converted the unbalanced binary class data into multiclass balanced data by using coding-based schemes and then trained the defect prediction models from this multiclass data. Wang and Yuo [74] investigated different methods, including resampling, ensembles, and threshold moving to improve defect prediction models. The study also proposed a dynamic version of AdaBoost to handle the imbalanced

data issue in the area of defect prediction. A paper by Zheng [75] proposed three cost-sensitive boosting techniques to improve the prediction rate of neural networks. Malhotra and Khanna [76] developed software change prediction models from imbalanced data by employing three data resampling methods and meta cost learners. In this way, various studies in the literature have dealt with imbalanced data in predictive modeling in the software engineering domain to improve defect prediction and change prediction models. However, the imbalanced data problem is untouched in the literature for SMP. This thesis deals with improving the performance of SMP models with the help of data level and algorithm level techniques.

1.8 Objectives of the Thesis

1.8.1 Vision

Improving software maintenance by developing SMP models that deal with imbalanced data problem.

1.8.2 Focus

The focus of the work has been to suggest methods that help in improving software maintainability. In this regard, effective SMP models are developed from open-source systems. This thesis endeavors to conduct the research in predictive modeling using OO metrics and ML, SB, and HB techniques. Keeping in mind the challenges of imbalanced data while developing the prediction models, the current study carefully handles the imbalanced datasets to develop effective SMP models. The cross-validation technique and statistical tests have been used while developing the models to ensure that the results are unbiased. Thus, this study explicitly addresses the following

perspectives:

1. To analyze the OO metrics available in the literature to examine their relationship with software maintainability.

2. To investigate the effectiveness of ML and ensemble techniques with data resampling to handle the imbalanced data for software maintainability.

3. To investigate the effectiveness of SB and HB techniques with data resampling to handle the imbalanced data for software maintainability.

4. To devise a new imbalanced data handling technique by improvising the existing technique in the literature.

5. To investigate whether one technique outperforms others in terms of performance measures with the help of statistical tests.

6. To develop SMP models with inter-project validation.

1.8.3 Goals

Aggarwal and Singh [77] have emphasized the importance of software maintenance. They stated that more than half of the total project cost goes into just maintaining a given software. Also, the literature survey conducted by the researchers such as Riaz et al. [78], Calzolari et al. [79], Ghosh et al. [80], and Saraiva et al. [81] showed that the cost of maintenance has been increasing day by day. This truly gives the researchers a motivation to think that how can we make the software product highly maintainable so that the future maintenance cost and efforts can be reduced. It is an unpleasant reality that a major portion of the time and effort in the software industry is expended in software maintenance as opposed to development. If maintainability of the software product or component is predicted early in software development,

resources and effort during the maintenance can be managed in a better way. Thus, the aim of this thesis is to develop effective SMP models. The summary of the goals is stated below:

1. Perform an extensive review of existing literature to identify the relationship between OO metrics and software maintainability with the following objectives:
    - Study the use of prediction models for maintainability in the early phases of development.
    - Extensive study of existing literature to understand the significance of software metrics that are related to maintainability.
    - Study of various ML and statistical techniques used in literature for developing SMP models.
    - Comparing the performance of various techniques used in the literature.
    - Practitioners would be able to improve the quality of systems and thus optimize maintenance costs.
    - To gain insights about both quantitative and qualitative perspectives of predictive modeling for software maintainability.
    - To identify the research gap to work further in the field of software maintainability.
2. Develop new models for prediction of software maintainability using ML techniques by handling the imbalanced data with the following objectives:
    - Empirically validate the relationship between OO metrics and software maintainability.
    - To preprocess the imbalanced data using data resampling techniques.

- To determine which data resampling technique performs statistically better from others.

3. Exploration of ensemble learners for handling the imbalanced data to improve software maintainability predictions.

    - Modeling techniques based on ensemble methodology are known to develop stable and robust models, which improve the capability of their constituent techniques. Such techniques can be used to develop SMP models with improved accuracy as they are based on diverse constituent techniques.
    - To develop new models by using ensemble learners which combine the data resampling techniques to handle the imbalanced data for software maintainability.
    - To assess the capability of ensembles to handle inbalanced data for prediction of software maintainability and compare them with the traditional ensemble learners.
    - To perform statistical analysis of the compared techniques with statistical tests.

4. To evaluate and compare various SB techniques for SMP and compare them with the ML techniques by dealing with the imbalanced data problem.

    - With a large number of available classification techniques, which have different abilities and characteristics, it is important to conduct studies to evaluate their capability in the domain of SMP.
    - Such studies can be used by researchers and practitioners as guide for the selection of an appropriate technique.

- The performance of compared techniques is assessed on the imbalanced and balanced datasets for SMP.

5. Exploration of HB techniques for prediction of maintainability by handling imbalanced data.

    - To explore HB techniques that tend to combine ML and SB techniques into a single approach and to assess their applicability for SMP.
    - To evaluate the performance of each technique so that the effectiveness of each technique SMP can be judged.
    - The performance of HB techniques for maintainability prediction is empirically investigated after employing data resampling techniques.

6. To propose new data resampling technique to alleviate the imbalanced data problem for SMP.

    - To contribute to imbalance data problem in SMP and to use it to improve the predictions of SMP models, a novel oversampling technique is proposed.
    - To conduct extensive empirical investigation comparing the performance of the proposed technique with state-of-art data resampling techniques.
    - To statistically analyze the performance of the proposed technique with other prevalent data resampling techniques.

7. To carry out inter-project validation to obtain generalized results for SMP.

    - The use of inter-project validation should be explored where training data is taken from some other similar software. There is a lack of work done in this field.

- To develop SMP models developed using inter-project validation to produce unbiased and generalized results.

1.9 Organization of the Thesis

This section presents the organization of the thesis. **Chapter 1** presents basic concepts of software maintenance, software maintainability along with the motivation of the thesis. **Chapter 2** describes the research methodology followed in carrying out this research work. **Chapter 3** presents a systematic review of existing literature and identifies the prevailing research gaps. **Chapter 4** presents the construction of SMP models using ML techniques by handling imbalanced data. **Chapter 5** presents the ensemble techniques for dealing with imbalanced data. Subsequently, **Chapter 6** analyses SB techniques for developing SMP models by handling imbalanced data and compares them with ML techniques. **Chapter 7** analyzes the capability of HB techniques for developing models by handling imbalanced data, while **Chapter 8** proposes a new data resampling technique Modified Safe Level Synthetic Minority Oversampling Technique (MSLSMOTE) to handle the imbalanced data problem and develop SMP models using ML techniques. **Chapter 9** analyses inter-project validation for developing effective SMP models. In **Chapter 10**, the conclusions of the thesis are presented. A brief description of each chapter is given below.

Chapter 1: This chapter describes the basic concepts about software maintenance, its types, and the definition of software maintainability. The benefits of the early prediction of software maintainability are enumerated in this chapter. Various software metrics have been used in software quality predictive modeling to develop prediction models. An overview of software metrics is given in this chapter. This chapter also describes the steps of developing maintainability prediction models.

Chapter 2: This chapter describes the detailed description of the research method-

ology followed in this research work i.e., a brief explanation of the dependent and independent variables, classification techniques, validation methods, and statistical tests. Further, the datasets used in the work, and the data collection procedure is described in this chapter. This chapter also describes the data preprocessing steps followed before developing SMP models. This chapter also describes various performance measures used to assess the performance of the developed models. Furthermore, ten-fold and inter-project validation techniques used in this research work for validating the prediction models have also been explained in this chapter.

Chapter 3: A comprehensive systematic literature review of the studies related to software maintainability from January 1990 to October 2019 is presented in this chapter. The research questions (RQs) have been framed that summarized the empirical evidence with respect to software metrics, datasets, predictive performance of data analysis techniques, statistical tests and threats to validity in these studies. The review results are analyzed systematically and research gaps have been identified.

Chapter 4: This chapter deals with the development of SMP models using ML techniques (Multilayer Perceptron with Conjugate Learning (MLP-CG), Multilayer Perceptron with Back-propagation (MLP-BP), Radial Basis Function Neural Network (RBFNN), Incremental Radial Basis Function Neural Network (IRBFNN), Bootstrap Aggregation (Bagging), Adaptive Boosting (AdaBoost), K Nearest Neighbour (KNN), KSTAR)) on eight open-source datasets (Bcel, Betwixt, Io, Ivy, Jcs, Lang, Log4j, Ode). The data resampling techniques including ROS (Random oversampling), Synthetic Minority Oversampling Technique (SMOTE), Borderline Synthetic Minority Oversampling Technique (BSMOTE), Adaptive Synthetic Sampling (Adasyn), Safe Level Synthetic Minority Oversampling Technique (SafeSMOTE), Synthetic minority oversampling technique with Tomek's link (SMOTE-TL), Synthetic Minority Oversampling Technique with Edited Nearest Neighbours (SMOTE-ENN), Selective Preprocessing of Imbalanced Data (SPIDER), Random Undersampling (RUS), Con-

densed Nearest Neighbor (CNN), Condensed Nearest Neighbor with Tomek's Link (CNN-TL), and Neighborhood Cleaning Rule (NCL) are examined in this chapter to deal with the imbalanced data problem.

Chapter 5: The application of ensemble techniques for imbalanced data problem is examined in this chapter. In this chapter development of SMP models has been done using the following ensembles (i) Boosting-based ensembles (SMOTEBoost, MSMOTEBoost, RUSBoost, DataBoost, and EUSBoost. (ii) Bagging-based ensembles UnderBagging, OverBagging, SMOTEBagging, MSMOTEBagging, UnderOverbagging, and IIVotes. (ii) Hybrid ensembles (EasyEnsemble, BalanceCascade). The empirical validation has been performed using eight open-source datasets, and results are evaluated using G-Mean and Balance performance measures.

Chapter 6: This chapter analyzes the capability of SB techniques with data resampling. The data resampling used in this chapter to balance the datasets are SMOTE, BSMOTE, SafeSMOTE, Adasyn, SMOTE-ENN, SMOTE-TL, SPIDER, NCL, RUS, ROS and CNN. The SB techniques used in this chapter for developing the SMP models are Genetic Algorithm based classification system with Adaptive Discretization Intervals (GA-ADI), Genetic Algorithm based Classification System with Intervalar Rules (GA-Int), X-Classification System (XCS), Supervised Classification System (UCS), Bioinformatics based Hierarchical Evolutionary Learning (BIOHEL), Memetic Pittsburgh learning classification System (MPLCS), Constricted Particle Swarm Optimization (CPSO), Linear Decreasing Weight Particle Swarm Optimization (LDWPSO), Generational Genetic Algorithm for Instance Selection (GGA), Steady State Algorithm for Instance Selection (SGA), Adaptive Search for Instance Selection (CHC), Population Based Incremental Learning (PBIL), Intelligent Genetic Algorithm (IGA), and Steady-State Memetic Algorithm for Instance Selection (SSMA)). The performance of SB techniques is also compared with ML techniques (MLP-CG, MLP-BP, RBFNN, IRBFNN, Bagging, AdaBoost, KNN, KSTAR, Partial

Decision Tree (PART), Repeated Incremental Pruning to Produce Error Reduction (RIPPER), Simple Learner with Iterative Pruning to Produce Error Reduction (SLIPPER), Batch Nested Generalized Exemplar (BNGE), Exemplar-Aided Constructor of Hyperrectangles (EACH), and Rule Induction from a Set of Exemplars (RISE), and Pruning and Building Integrated in Classification (PUBIC)) in this chapter.

Chapter 7: This chapter evaluates the effectiveness of HB techniques for SMP with data resampling. The SMP models are developed with the following HB techniques: Tree-Genetic Algorithm (DT-GA), Tree Analysis with Randomly Generated and Evolved Trees (TARGET), Fuzzy LogitBoost (GFS-LB), Logitboost with Single Winner Inference (GFS-MaxLB), Fuzzy AdaBoost (GFS-AB), Genetic Programming-based learning of Compact and Accurate fuzzy rule-based classification systems for High-dimensional problems (GP-COACH), Particle Swarm Optimization with Linear Discrimination Analysis (PSOLDA), Fuzzy Learning based on Genetic Programming (GFS-GP), GFS-GP with Grammar Operators and Simulated Annealing (GFS-SP) and Fuzzy Rule approach based on Genetic Cooperative Competitive Learning (GFS-GCCL). The predictive performance of the models is assessed with the help of G-Mean and Balance performance measures.

Chapter 8: This chapter proposes a data resampling technique namely MSLSMOTE to alleviate the imbalanced data problem. The empirical investigation comparing the performance of MSLSMOTE with SMOTE, BSMOTE, and SafeSMOTE at different rates of oversampling has been carried out in this chapter. The SMP models are developed with ML techniques used in Chapter 6. The performance of the developed models is assessed with the performance measures G-Mean, Balance and Area Under Receiver Operator Characteristic Curve (AUC-ROC). The results are statistically evaluated with the help of Friedman test and Wilcoxon signed-rank test.

Chapter 9: In this chapter, inter-project predictions are carried out for SMP where the training and validation of the maintainability models have been done on

different projects rather than the same project. The experiments are conducted using dataset extracted from three open-source projects, namely Htmlunit, Maven, and Click. The prediction models are developed using eleven ML techniques and two statistical techniques. The performance of the models is analyzed using MMRE, Pred(25), Pred(30), and Pred(75).

Chapter 10: This chapter summarizes the conclusion of the research work and provides future directions.

Chapter 2

Research Methodology

2.1 Introduction

We need to follow a systematic procedure to accomplish our objectives and perform reliable empirical experiments. The research methodology is a systematic and well-defined order of steps that are required to conduct effective empirical experimentation. This chapter describes the research methodology that is followed in the thesis. The organization of the chapter is as follows: Section 2.2 presents the research process followed for conducting empirical experiments. Section 2.3 presents the definition of the research problem. Section 2.4 describes the literature review conducted to provide an overview of the existing literature. In section 2.5 the independent and dependent variables used in this work are defined. Section 2.6 is designated for various data analysis techniques used in this work. Also, the parameter setting for various data analysis techniques is defined in this section. Section 2.7 demonstrates the experimental design framework of the thesis.

2.2 Research Process

The research process involves a well-ordered and planned sequence of steps required for the exploration of a research problem. Figure 2.1 describes the research process that is followed in the chapters of this thesis. The various steps of the research process are explained in the following sections.

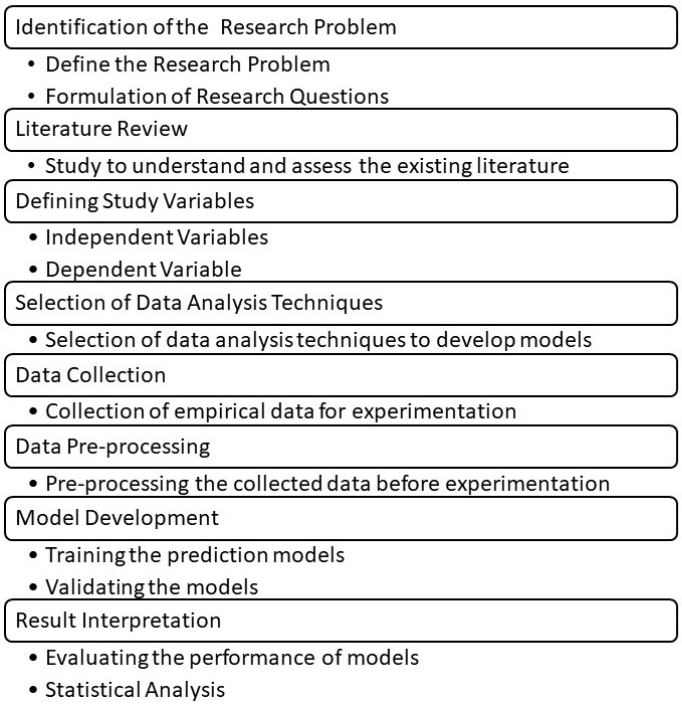


Figure 2.1: Research Process

2.3 Define Research Problem

The first step in the research process is to formulate the research problem. In this step, the research problem at hand is stated and defined in the form of RQs. Research experiments are conducted to determine the answers to the RQs. The following RQs

are addressed in this thesis:

1. What is the current state of literature studies in the domain of SMP and what are the research gaps?
2. What is the performance of SMP models developed from imbalanced data?
3. What techniques can be used by researchers to develop efficient SMP models from the imbalanced data?
4. What is the performance of different data analysis techniques such as ML, SB, and HB techniques for developing SMP models?

2.4 Literature Survey

To understand the research problem, a literature survey of existing studies is essential. With the help of the review, we get to know to what extent the research problem in literature has been investigated. In the past, researchers have developed various models for predicting software maintainability [6, 9, 13, 15, 16, 46, 57, 66, 82, 83]. These models have been developed by establishing the relationship between the internal characteristics of the software and software maintainability. The internal characteristics of software have been captured with the help of software metrics. These models have been developed on commercial and open-source project datasets. These models help the software practitioners to predict the software maintainability when the software is in the initial development stages. Thus, identifying the software maintainability, early during the software development would help the project managers to allocate the resources optimally to the software components with low maintainability. Therefore, it has been well recognized in the literature that developing effective SMP models is crucial and important to improve software quality.

2.5 Defining Variables

There are two types of variables involved in an empirical investigation i.e., the independent (predictor) and the dependent (response) variable. The dependent variable is a variable the researcher is interested in. In this research work, the dependent variable is software maintainability. The dependent variable is influenced by the variables called independent variables or predictor variables. The dependent variable can be predicted from the independent variables. The independent variable should be independent in nature i.e., these variables should not be affected by the other variables under analysis. This thesis is indented to develop SMP models that learn the independent variables to predict software maintainability i.e., dependent variable. The independent variables in this research work are OO metrics that represent the various characteristics of the software classes. We explore the independent variables from various popular OO metrics suites and determine their relationship with software maintainability.

2.5.1 Independent Variables (OO metrics)

The applicability of OO metrics for developing models for the prediction of faulty classes, predicting change-prone classes, and predicting software maintainability has been well-acknowledged in the literature. The OO metrics quality aspects of the software systems like coupling, reusability, cohesion, size, etc. To effectively administer and supervise a software product during its development, it is essential to monitor these metrics.

The OO metrics used in the thesis include the following:

- Chidamber and Kemerer (C&K) metric suite [36] contains six OO metrics namely WMC, DIT, CBO, LOCM, and RFC. C&K metrics are widely used in the literature [16, 84, 85].

- Quality Model for Object-Oriented Design (QMOOD) [41] metric suite is also validated in the thesis. The metrics contained in QMOOD metric suite are MOA, DAM, MFA, NPM, and CAM.

- Martin metrics [31] namely Ce and Ca have also been analyzed in the thesis. Other metrics used in the thesis are AMC, SLOC, and LCOM3 (given by [86]), CBM, and IC.

Table 2.1: Independent Variables

Metric	**Definition**	**Source**
Weighted Method per Class (WMC)	The sum of cyclometic complexities of all methods of a class is called weighted method per class.	[36]
Depth of Inheritance Tree (DIT)	It is defined as the length of the longest path in the inheritance hierarchy.	[36]
Number of Children (NOC)	It is defined as the number of immediately derived classes of a particular class.	[36]
Coupling between Objects (CBO)	This metrics shows the number of classes to which a specific class is coupled where the coupling can be due to data accesses, function calls, inheritance etc.	[36]
Response for Class (RFC)	The number of functions executed in response to a message received by an object of a class is called response for class.	[36]
Lack of Cohesion in Methods (LCOM)	This metric measures the number of methods in a class that is not related to themselves by sharing some of the class data.	[36]
Afferent Coupling (Ca)	This metrics measures merely the number of classes that use a particular class.	[31]
Efferent Coupling (Ce)	Number of classes used by a specific class is called efferent coupling of that class.	[31]

Number of Public Methods (NPM)	This is the count of the number of public methods defined in a class.	[41]
Data Access Metrics (DAM)	It is described as the number of protected or private attributes declared in a class divided by total number of attributes declared in that class.	[41]
Measure of Aggregation (MOA)	This metrics is a count of number of data fields in a class whose type is user-defined.	[41]
Measure of Functional Abstraction (MFA)	This metric defines the number of methods that are inherited by a specific class divided by the sum of methods accessible by that class.	[41]
Cohesion Among Methods of Class (CAM)	This metrics measure the relationship amongst the class methods. The association is found by their list of arguments and defined as: the sum of the number of unique argument types used by all of the class methods divided by product of total number of methods in the class and total count of unique argument type in that class.	[41]
Inheritance Coupling (IC)	The number of parent classes to which given class is coupled is called inheritance coupling of that class.	`http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/`
Coupling Between Methods (CBM)	This metric computes the total number of methods in a class to which all of the metrics that are inherited, coupled.	`http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/`
Average Method Complexity (AMC)	This average method size of each class is called average method complexity.	`http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/`

Lines of Source code (SLOC)	It is defined as total number of lines in the binary code of a class.	[36]
Lack of cohesion (LCOM3)	LCOM3 is given as: $$LCOM3 = \frac{1}{n}(\sum_{i}^{n} f(pi)) - \frac{ma}{1-ma} \quad (2.1)$$ Here n= number of attributes in a class; ma = number of methods in a class and $f(pi)$ = number of functions that access an attribute.	[86]

These metrics describe different aspects of OO systems namely cohesion, coupling, size, inheritance, composition, and encapsulation etc. The metrics WMC, NPM, LOC, DAM, and AMC are indicators of the size of a class. The metrics CBO, RFC, Ca, Ce, IC, and CBM measure the coupling. The inheritance property is measured with the help of NOC, DIT, and MFC. The metrics LCOM, CAM, and LCOM3 are indicators of class cohesion, whereas MOA measures composition. The metrics that quantify the different characteristics of a class are regarded as internal quality attributes. The internal quality attributes used have significant relation with software maintainability [8, 13, 49]. For instance, the attributes WMC, NPM, LOC, DAM, and AMC measures the size of a class. If the size increases, code would likely be less maintainable, i.e., likely to require high maintainability effort [49]. Table 2.1 shows a brief explanation of the above OO metrics. These metrics have been widely used by the researchers for predictive modeling in the domain of software engineering [85, 87–90]. Radjenovic et al. [91] conducted a review of 106 papers predicting defects in classes. This review revealed that C&K metrics have frequently been used for predicting faults in classes. Therefore, our study also has taken C&K metrics to validate them for predicting software maintainability. The paper Lu et al. [92] assessed sixty-two OO metrics for

estimating the change-prone classes. It was discovered in the study that LOC, CBO, LCOM, and CAM are significant metrics. The C&K metrics, combined with QMOOD metrics, are used by [93] to predict change-prone classes. The study advocated the combination of C&K and QMOOD metrics be the competent predictors for predicting classes that are likely to be changed in the future. Therefore, in this research work, an effective combination of OO metrics is used for determining software maintainability.

2.5.2 Dependent Variable

In this thesis, software maintainability is the dependent variable. It is a binary variable i.e., there will be two values of maintainability for any class namely low maintainability and high maintainability. A low maintainability class would require more revisions in its source code as compare to a high maintainability class in the maintenance period and increase the overall cost of software development. In order to determine the maintainability, two consecutive versions of the same software i.e., the old version and next subsequent version are analyzed. The common classes between the two consecutive software versions are determined. Then, the total number of changes in the lines of source code in common classes i.e., lines of source code added, deleted, modified, and total lines of source code changed are calculated. The studies in the literature have determined the lines of code changed in the same way [5, 7, 46, 66]. The total lines of source code changed in a common class is a continuous variable. We convert this continuous variable into a binary variable by dividing it into two bins with two values, “low maintainability“ and “high maintainability“. In this thesis, we also predict maintainability in the form of a continuous variable.

2.6 Data Analysis Methods

Several data analysis techniques including statistical, ML, SB, and HB are used in this thesis to develop SMP models. Statistical techniques model the relationship between the independent and dependent variables using mathematical equations. ML techniques learn from historical data in order to model the relationships between the independent and dependent variables without relying on a predetermined mathematical equation as a model. The developed model is then used to predict the outcome of future data points. SB techniques aid in the development of optimized models with improved predictive capability as such techniques are effective in solving optimization problems. The HB techniques combine the statistical/ML techniques with SB techniques to give a single efficient technique to be used for a prediction task.

2.6.1 Artificial Neural Network

ANN is a computing system inspired by biological neural network. Such systems learn to do tasks by considering examples, generally without task-specific programming. MLP is an ANN that maps a set of inputs to the target or desired outputs [94]. MLP consists of three types of layers namely, input layer, an output layer and, one or more hidden layers. The inputs from the input layer are mapped to the desired outputs using the hidden layers. MLP-BP is one of the common forms of training the network. Training the MLP-BP comprises two passes i.e. forward and backward pass. In the forward pass, the inputs are applied at the input layer and the output is of the network is calculated. The calculated output is termed as the actual output of the network. In the backward pass, an error which is the difference of actual output and the target output is propagated backward from the output layer to the input layers. In this process, weights of the network are adjusted and input with modified weights is again

propagated forward in the network so that the calculated output becomes close to the desired output. In this thesis, MLP-BP is simulated using the Waikato Environment for Knowledge Analysis (WEKA) tool [95] with default parameter settings. The parameters used for MLP-BP are a learning rate of 0.3, a validation threshold of 20, and a momentum of 0.2.

MLP-CG is also a feed-forward and back-propagation neural network. MLP-BP is based on the gradient descent approach and it has poor convergence because learning rate and momentum need to be given by the user and there is no theoretical basis to choose these parameters. MLP-CG avoids this limitation of MLP-BP [94]. It has fast convergence as it performs search along with the conjugate directions.

RBFNN is a type of non-linear feed-forward neural network with a single hidden layer. It has guaranteed learning because of a single layer of weights that are adjustable and calculated by a linear optimization problem. This neural network can represent non-linear transformations [96].

2.6.2 Decision Tree

Decision tree (DT) algorithms develop the prediction model by constructing a decision tree and predict the label of data points by traversing the nodes of the DT from the root node to the leaves. The interior nodes in the DT represent the attribute test whereas the leaves represent the class labels. During the course of constructing the DT, the most significant predictor is identified that splits the available training data in the best possible way. DT is built iteratively by dividing the training data into partitions based on test conditions on one of the predictors in the training data. The attribute spitting schemes differentiate the decision trees. For easy interpretability, DT can easily be transformed into decision rules that are to be traversed in order to find the class label of future instance. Three DT based classifiers are used in this thesis:

C4.5, Classification and Regression Tres (CART), Decision Stump (DS) and Reduced Error Pruning Tree (REPTree). C4.5 uses gain ratio for attribute splitting criteria whereas CART uses the Gini index [97, 98]. REPTree develops a DT and then applies reduced-error pruning to it in order to develop a fast learner. REPTree algorithm generates the DT in two phases. In the first phase, complete DT is built and the second phase comprises pruning the fully developed DT. However, constructing the DT in two distinct phases a significant amount of effort is wasted [95]. Because fully grown DT is pruned in the subsequent phase. PUBIC overcomes this limitation of REPTree. In this algorithm, the building and pruning are combined into one single phase that aid in the construction of efficient DT as repeated dataset scans got eliminated [99]. For C4.5 and CART decision tree algorithms, the parameter setting includes a confidence factor of 0.25 and two instances per leaf. The parameter settings for REPTree and PUBLIC involve using two instances per leaf node and pruning factor as true. DS is a one-level DT in which split at root node is based on attribute-value pair. It has fast model building speed [100]. The parameter setting of DS include bag size of 100.

2.6.3 Rule-Based Classifiers

Rule-based classifiers refers to the classification algorithm that use "if-then" classification rules in order to predict the class labels [101]. The classification rules are of the following form: $Condition_i \rightarrow Y_i$. Here $Condition_i$ is the left-hand side of the rule and is termed as "rule antecedent". Y_i is called "rule consequent". $Condition_i$ comprehends conjunction of tests on predictor variable and it is represented as follows: $Condition_i : \ (P_1 \ operator \ V_1) \wedge (P_2 \ operator \ V_2) \wedge ....(P_n \ operator \ V_n)$.
Here, $(P_i \ operator \ V_i)$ is predictor-value pair, $operator$ is any logical operator from the set $(=, >, <, \leq, \geq)$ and Y_i is the class label. The rule-based classifiers used in this thesis are RIPPER, SLIPPER, and PART. RIPPER works by extraction of rules or rule

set from the available training data where each rule identifies the relationship between the independent variables and the class label [102]. From the training data set, the rules are derived directly by sequential covering. Rules are derived from one class at a time. For the binary classification, first, all the rules of one class are derived then rules for the second class are constructed. The algorithm starts with the empty rule list and then learn one rule function is applied to the training data set to extract the best possible rule. Here, the best rule means the rule which covers more data points of one particular class type. The newly derived rule is then added to the ruleset. When a new rule is added to the ruleset, all the data points covered by that rule are removed from the training data set. After forming the ruleset, pruning is applied to construct an optimal rule set. SLIPPER algorithm develops a weighted rule set. There is a confidence factor associated with each rule in the ruleset. To predict the class label of an instance, all rules covering the instance are determined and their confidence factors are summed up [102]. The label for the instance is then predicted according to the sign of the sum. The PART algorithm constructs the partial DT using the training data set instead of developing the fully grown DT. The tree construction and pruning are combined to develop a stable tree. Once the stable tree is developed, tree construction is stopped and the decision rules are read off [103].

2.6.4 Ensemble Learning Techniques

Ensemble learning techniques train the multiple classification models and collate their predictions to obtain the distinct class label. The ensemble learning techniques used in this thesis are AdaBoost and Bagging. These techniques modify the original training examples and create multiple models from them. The final predicted outcome is produced after combining the predictions of the individual models.

AdaBoost is based on the boosting method. The boosting technique develops a

powerful classification model by using some weak classifiers. AdaBoost improves the predictive power of weak classifiers. The learning of weak classifiers is carried out by using the weighted training data samples, and the misclassification rate of the individual classification models is determined. This algorithm includes a weight updating process. The correctly classified data points get small weights, and the wrongly classified data points get large weights. In this manner, AdaBoost focuses more on difficult to learn data points [104]. The parameter setting for the AdaBoost includes C4.5 tree as a base learner, two instances per leaf node, the confidence factor is 0.25, and 10 number of classifiers.

Random Forest (RF) constructs multiple decision trees using the training set. In the testing stage, each DT predicts the class label [95]. The final predicted class is decided based on majority voting. In this work, RF uses 100 decision trees.

Bagging is an ensemble technique that creates individual subsets of the training dataset randomly with replacement. A predictor is developed corresponding to each subset [105]. The results of individual predictors are then either averaged or combined using majority voting. The parameter setting for the Bagging includes C4.5 DT as a base learner, two instances per leaf node, the confidence factor is 0.25, and 10 number of classifiers.

2.6.5 Logistic Regression

Logistic regression (LR) is one of the most widely used statistical techniques in the literature. LR estimates the variance in the values of the dependent variable caused by the independent variable [106]. LR analysis is of two types, (i) Univariate and (ii) Multivariate. With univariate LR analysis, the relationship of each independent variable with the dependent variable is determined, whereas with multivariate LR, using the complete set of independent variables, the prediction model is developed

to estimate the dependent variable. In multivariate LR, the independent variables are used in combination. Forward selection and backward elimination are two specific ways to select independent variables among the set of the available set of independent variables. In the forward selection method, one variable at a time is selected and included in the model. In backward elimination, model development starts with all variables, and then one variable at a time is eliminated and this process continues until the desired model is obtained [106]. The univariate LR is defined by the following formula:

$$Odds = \frac{p}{1-p} \tag{2.2}$$

Here, p is the probability of class being low maintainable. In LR, we estimate an unknown p for any given linear combination of the independent variables. The simple logistic model is based on the linear relationship between a numerical independent variable and the natural logarithm of an event. The simple logistic model is thus given by the following equation:

$$\ln(Odds) = \beta_0 + \beta_1 x_1 \tag{2.3}$$

Here, x is the independent variable, β_0 is the y-intercept and β_1 corresponds to the slope. Using equations 2.2 and 2.3, the univariate formula is given as follows:

$$p = \frac{e^{f(x)}}{1 + e^{f(x)}} \tag{2.4}$$

Where, $f(x) = \beta_0 + \beta_1$. The equation for the univariate LR can be extended to the multivariate equation as follows:

$$p = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\beta_3 x_3+....+\beta_n x_n}}{1 + e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\beta_3 x_3+....+\beta_n x_n}} \tag{2.5}$$

where $x_i; 1 \leq i \leq n$ are the independent variables.

Maximum Likelihood Estimation is a well-known technique for estimating the coefficients of the model. The likelihood function determines the probability of perceiving the set of dependent variable values. The objective of is to determine the values of coefficients, such that the log of likelihood function is maximized. The large value of coefficient denotes higher impact of the independent variables on dependent variable. The ridge estimator is used in this thesis with LR to enhance the parameter estimates and lower the error when maximum-likelihood estimators cannot fit the data [106].

2.6.6 Support Vector Machine

SVM are important techniques for solving classification problems. These are used in many real-world problems like text classification, face recognition, medical diagnosis, etc. A data point the training data set is represented by an n-dimensional vector and belongs to one of the classes. SVM creates a hyperplane to separate the data points belonging to the two classes [107]. Apart from creating a hyperplane, it also creates two marginal lines and these two marginal lines are some distance apart so this it will be easily linearly separable for both the classification points. The sum of distances of the hyperplane from the two marginal lines is termed as ”margin”. The significance of the margin lies in the fact that it aids in the creation of a generalized classification model that gives better accuracy for predicting the class label of unseen instance. It is to be noted that there may be many hyperplanes that correctly separates the data

points. In this situation, SVM selects the hyperplane that maximizes the margin.

In the case of the ”non-linearly” separable data points i.e., the data points that can not be separated from any of the hyperplanes, SVM makes use of kernel. SVM kernel converts the data points from a low-dimensional space to a high-dimensional space. The data points on the high-dimensional space are then separated with the help of a hyperplane. The parameter setting for SVM in this thesis includes a c value of 100, a degree of 1, γ value of 0.01, v value of 0.1, and polynomial kernel.

2.6.7 Instance-based Learning Techniques

Instance-based learning techniques make predictions by memorizing the training data instead of developing the generalized models. To determine the target class label of an unseen data point, instance-based learning techniques compute the similarity of that data point with the data points of the training data. The instance-based learning techniques used in this thesis are KNN, KSTAR, BNGE, EACH, and RISE. In KNN entire training dataset is stored in memory. To determine the target class of a new instance, the distance is computed between the instance and each of the instances of the training data set stored in the memory. More specifically, the *k* nearest neighbors of the new instance are determined. The class label of the new instance is then determined based on the labels of the majority of the neighbors amongst the *k* nearest neighbors [108]. The distance measure used in KNN is the euclidean distance. KSTAR is also an in-memory technique. Like KNN, it also determines the label of the new instance according to its neighbors. The difference lies in the distance measure. KSTAR uses entropy distance measure [109]. KNN and KSTAR require huge memory space as the entire data set is stored in the memory for classification. BNGE [110], EACH [111], and RISE [112] algorithm stores data in a compact way in memory. The training examples corresponding to the same class are represented in the form of hyper

rectangles that lead to fast classification.

In BNGE and EACH, an exemplar corresponds to a single training data point and a generalized exemplar is a hyperrectangle that covers multiple training data points. To find the label of a test data point, its distance from the generalized exemplar is computed. RISE combines instance-based learning and rule induction into one single approach.

Unlike the other instance-based learning techniques that classify a test instance based on its distance from the data points kept in memory, RISE finds the label of an unseen data point by assigning it to the class label of the nearest rule in the ruleset.

The instance-based techniques generally work not well on large training datasets. Therefore, this thesis also applies few evolutionary instance selection techniques that tend to determine an optimal training dataset for classification. The basic concept in GGA is to maintain a population of chromosomes. The chromosomes epitomize probable solutions to the problem that evolves in the succeeding iterations through a process of competition. A fitness value is associated with each chromosome that aid in the selection of chromosomes in the competition process to form new chromosomes. GGA consists of three operations namely assessment of the fitness of individuals, development of an intermediate population through selection, and recombination using crossover and mutation operators [113].

SGA picks two parents from the population. An offspring is created with the help of crossover and mutation which is then evaluated using the fitness function. Then an individual in the population is decided to be replaced by the offspring depending on the replacement strategy [113].

IGA employs an intelligent cross-over operator that intelligently selects the better genes for the formation of the chromosomes of children [114]. Intelligent crossover replaces the traditional generate-and-test search for children using a random combination of chromosomes with a systematic reasoning search technique using an intelligent

combination for selecting better genes. The algorithm works as follows: A random population of individuals is initialized, and their fitness is evaluated. Then a rank selection is performed that replaces the worst individual with the best one. individuals to form a new population. Random individuals are then selected for performing intelligent crossover. Then mutation is applied using a bit inverse mutation operator. The parameter setting of GGA, SGA, and IGA include population size of 51, 10000 number of evaluations, 5 number of neighbors, mutation probability of 0.01, and cross-over probability of 0.6.

CHC is an adaptive search algorithm that uses a parent population of size n to produce an intermediate population of n individuals. These n individuals are arbitrarily paired and offspring are produced. The survival competition is held where the best chromosomes from the parent and offspring populations are selected and the next generation is formed [113]. The parameter setting for CHC includes population size of 50, the number of evaluations is 10000, percentage of change in restart is 0.35, number of neighbors is equal to 5 and the distance function is Euclidean.

PBIL technique uses stochastic exploration to adjust a probability array, which converges toward an optimal solution with the help of GA. The probability array is adjusted iteratively by considering the solutions explored in the previous iterations [113]. If the feasible solution is represented as a binary string, the probability array is represented as a vector. Each bit in this vector stores probability related to each bit of the optimal solution. Once probability entry converges to one, it depicts the corresponding value of the associated bit for the optimal solution. The parameter settings of PBIL are as follows: a number of generations are equal to 10000, the population size of 50, mutation probability of 0.05, the number of neighbors are 5, and the Euclidean distance function.

SSMA works by forming all subsets of training data and chromosome represents the subsets of training data using binary representation [115]. The chromosome

contains n genes i.e., one for each example in training data with two states 1 and 0. If the corresponding bit for a gene is 1, the example would be contained in the subset of of training data and if it is 0, the example is would not be contained in the subset.

$$Fitness function = a * C.R + (1 - a) * P.R \tag{2.6}$$

Here, $C.R$ is a classification error associated with the selected subset, and $P.R$ indicates the percentage of reduction of examples of the subset with respect to training data.

2.6.8 Genetic Algorithm based Classifier System

Genetic Algorithm Based Classifier System (GAssist), the knowledge representation takes the form of classification rules and GA is used for the evolution of these rules [116]. A chromosome in a GA is representative of a classification rule and it is coded in the form of a bit array where the individual bits represent the attributes of the training example. The bits correspond to genes of the chromosomes. To characterize a real-valued attribute in the form of a chromosome, discretization using the proper number of intervals is required. Therefore, the bits signify a discretization interval. The traditional technique used in GA is a set of rules where the antecedent of the rule is defined by a prefixed finite number of intervals to handle real-valued attributes. Therefore, the competence of these systems is highly reliant on the correct choice of the intervals. Hence, a rule representation with adaptive discrete intervals is essential. In GA-ADI, the intervals are split and combined during the evolution process [117]. GA-Int uses intervalar rules [118]. The fitness of individual rules is evaluated based on the proportion of examples that are being correctly classified. The parameter settings for GA-ADI and GA-Int include 500 iterations, Strata of 2, min rule deletions equal to 12, and 4 as the size of the penalty of minimum rules. in addition to above parameters

4,5,6,7,8,10,15,20,25 discretization intervals are used. The fitness function in GA-ADI and GS-Int is $Accuracy$ which is defined in Section 2.7.7.

2.6.9 Learning Classifier System

Learning Classifier System (LCS) is a versatile classification system that is trained to perform achieve the best actions from the given input. In the context of classification, "input" is the set of independent variables and actions characterize the predicted target class. The rule-set for LCS comprises rules of the form of "condition-action". For a specific set of input comprising the independent variables, sometimes more than one rule can be triggered and predict different target classes. In this situation, LCS computes the average of the predictions of the classifiers that target class. And the target class having the highest average is predicted as the resultant target class. Some amount of payoff is returned by the system when the target class with the highest average prediction is returned. This payoff is used to modify the prediction during the training of LCS. Each classifier in LCS also keeps track of the errors of the misclassifications. During each iteration of the training, LCS regulates the fitness by moving close to the inverse of error. Also, in each iteration, the classifiers with higher fitness are evolved over the less fit variants by modifying the âoffspringâ using the genetics operators i.e., cross-over and mutation.

XCS makes use of genetic algorithm with reinforcement learning scheme. With this scheme, when the input is given to the system, the different rules can be triggered based preconditions of the rules matching with the input. The competing rules get reward based on their actions. The algorithm evolves as a population of classifiers, where each classifier consists of a rule and parameters for estimating the quality of the rule. The GA in XCS is applied to the action sets, rather than over all the population. First, it selects two parents from the actual action set with probability proportional

to fitness. Then, the parents are crossed and mutated. The resulting offspring are introduced into the population [119]. The parameter for XCS technique used this thesis are, a population size of 6,400, a crossover probability of 0.8, 1,00,000 explores, two-point crossover type, a mutation probability of 0.04, free mutation, roulette wheel selection, δ = 0.1, θ_{mna} = 2, θ_{sub} = 50.0, θ_{ga} = 50.0, α = 0.1, β = 0.2, θ_{del} = 50.0 μ = 10.0, r = 1.0, m = 0.1, 0.4 as size of tournament, 10.0 as initial prediction, 0.0 as initial prediction error, 0.25 as reduction of prediction error, 0.01 as initial fitness value and GA subsumption as true.

$$Error = (1/R\,(|R - P_c| * P_{cl} + |\theta - P_c| * (1 - P_{cl}))) \tag{2.7}$$

Here R denotes reward, P_c denotes prediction given by classifier, and P_{cl} is probability of correct classification.

UCS is intended for supervised environments. The training in UCS is performed in a supervised way where each training instance is accompanying with a class label. This is different from XCS where reinforcement learning is performed and the system gets rewards corresponding to the actions [120]. In UCS, when a with input instance e is presented during training, a match set (M) is formed. The match set contains classification rules whose conditions match with the input e. The classifiers in (M) that predict the correct class of e form a set C, regarded as the correct-set. Remaining classifiers in M said to belong to incorrect set $(!C)$. In the testing phase, when an instance e is presented, the system has to predict the associated class. The predicted class is determined by the weighting of the vote of all the classifiers in the match set (M). The weights are assigned according to fitness. In UCS, GA is applied only to correct-set (C). Two classifiers from set (C) are selected with probability proportional to fitness. The crossover and mutation are applied to the selected classifiers. The parameter settings used for UCS in this thesis are the same as that of XCS. The fitness

function of UCS is accuracy which is given by equation 2.7 where v is a constant.

$$Accuracy = (\frac{Number\ of\ Correct\ Predictions}{Number\ of\ Matches})^{v} \tag{2.8}$$

BIOHEL entails the iterative learning of rules. It is strongly influenced by the GAssist Pittsburgh learning classifiers systems and works well for large data sets [121]. The various features of BIOHEL are inherited from GAssist. The learning process is guided by creating one rule at a time using GA. Once a rule is found, the training instances covered by that rule are taken out of the training data set and GA is enforced to discover other areas of the search space to learn another rule. The discovered rules are iteratively inserted into the ruleset. Also, o default rule covering the majority of the class domain is added into the ruleset. The learned rules are continuously evolved. For BIOHel, the parameters used in this thesis are, the population size of 500, tournament size of 4, and crossover and mutation probability is equal to 0.6 with 100 iterations.

MPLCS is another classification technique used in this thesis. In MPLCS, Pittsburgh refers to the approach that comprises rule-sets rather than individual rules and Memetic denotes the combination of population based global search along with cultural evolution in the search cycle i.e., local refinement [122]. MPLCS incorporates GAssist with a local search algorithm using a rule-set wise operator (RSW). RSW works in three stages. In the first stage, all the available rules are evaluated with respect to all the instances in the training dataset. This process would result in correct and incorrect classification of each rule. In the second stage, the position in the rule-set is determined where inserting the candidate rule would lead to an increase in the accuracy of the rule-set. In the last stage, the highest accuracy rules are picked and offspring is generated out of the selected rules.

The parameters used for MPLCS include 750 iterations, 0.05 as local search probability, 0.1 as rule set-wise crossover probability, 4 as the size of penalty rules,

and 5 rule ordering repetitions. The fitness function for MPLCS is as follows:

$$FitnessFunction = Exception\ Bits + Weight * Theory\ Bits. \tag{2.9}$$

In the above equation, Weight assigns the weight for adjustment of exception. $TheoryBits$ signifies the length of all classification rules that are alive and $Exception\ Bits$ implies all examples which are wrongly classified.

2.6.10 Decision Trees with Genetic Algorithm

DT algorithms are biased towards generalization i.e., they generally favor rules covering a large number of instances i.e., large disjuncts. They ignore the rules with small disjuncts which cover a few numbers of instances. Therefore, with DT algorithms, it is challenging to predict the target class of small disjuncts instances. Small disjuncts should not just be ignored because they may cover few instances but set small disjuncts may cover a large number of instances [123]. GA is a robust and flexible search algorithm that can cope with attribute interaction better than DT algorithms. GA is suitable for finding rules that cover a small number of instances. It employs flexible search and does not get trapped in local minima. DT-GA is a hybridization of DT and GA that integrates the advantages of its constituent techniques. The training in DT-GA is done in two distinct stages. The first stage runs the C4.5 DT algorithm on the training dataset. DT is pruned and transformed into decision rules. In the second phase, GA determines the decision rules covering instances belonging to small disjuncts. In this thesis, DT-GA is used with 2 instances per leaf, a confidence of 0.20, 10 as the threshold for considering small disjuncts, 50 generations for GA, 200 chromosomes in the population, a crossover probability of 0.7, and a mutation probability of 0.01. The fitness function is the multiplication of sensitivity and specificity.

The TARGET initializes a forest of random trees developed using CART. the fitness of each tree in the forest is evaluated and the current forest is evolved to a new one with the help of genetic operators [124]. This process is continued until the improvement in the fitness function not get stabilized. The parameter setting for TARGET includes 100 generations of GA, the number of trees generated by crossover is 30, the number of trees generated by mutation is 10, 0.5 as the probability of splitting the node. The fitness function of TARGET is Gini-index.

2.6.11 Constricted and Linear Decreasing Weight Particle Swarm Optimization

CPSO and LWPSO are variant of the Particle Swarm Optimization (PSO) algorithm. PSO is based on the fact that an intelligent optimization solution can be accomplished by cooperative behavior as contrasting to an isolated individual reactive response to the environment. The solutions in PSO are looked in a dispersed way [125]. The algorithm maintains the following conditions, (i) function which is required to be optimized (ii) the global best (g_{best}) that characterizes the best value obtained by any particle in the solution space (ii) the termination condition to stop the algorithm. A particle in PSO also contains the required data that is the representation of a probable solution, velocity value that specifies the degree to which the data can be changed, and) the best solution (p_{best}) indicating the best particle value attained at a moment. The algorithm investigates the entire search area and looks for the utmost favorable region. In each iteration, velocity (v_t) of each particle at time t is updated according to p_{best}, g_{best}, current position (x_{t-1}) and current velocity (v_{t-1}). During the exploration of the search space for an optimal solution, it is important to avoid search space. To do so, the use of proper constriction coefficients (δ) has been suggested by Clerc and Kennedy [126]. The constriction coefficient (δ) help in controlling the exploration. And the

PSO variant that uses constriction coefficients is termed as the CPSO algorithm. In addition to δ, CPSO algorithm also uses two additional parameters namely cognitive parameter (C_1) and social parameter (C_2). The position of a particle according to its local best is updated by C_1 and the position of a particle according to its global best is updated by C_2. The velocity of the particle is updated as follows:

$$v_t = \delta \left(v_{t-1} + C_1 \left(p_{best} - x_{t-1} \right) + C_2 \left(g_{best} - x_{t-1} \right) \right) \tag{2.10}$$

The position of the particle is updated according to the following equation:

$$x_t = \left(x_{t-1} \right) + v_t \tag{2.11}$$

In this work, the following CPSO parameters are used for model development: 25 particles, convergence radius of 0.1, 2.05 as maximum weights for C_1 and C_2, a maximum of 0.1 uncovered instances, δ of 0.73, 0.1 as the threshold for indifference and a convergence platform of width 30

In LDWPSO, a linearly decreasing inertia W_{ldw} parameter is used that balances out the global and local search abilities of the swarm in an effective manner [127]. The parameter (W_{ldw}) decreases linearly from 0.9 to 0.4 during the search process. The equation for the linearly decreased weight (W_{ldw}) is given below:

$$W_{ldw} = (W_{maximum} - W_{minimum}) * \frac{It_{maximum} - It_i}{It_{maximum}} + w_{minimum} \tag{2.12}$$

Here, $W_{maximum}$ = 0.9, $W_{minimum}$ = 0.4, $It_{maximum}$ denotes maximum number of iterations in the algorithm, and It_i denotes i^{th} iteration. Using, linearly decreasing

inertia (W_{ldw}) parameter, the updated equation of velocity is given as follows:

$$v_t = W_{ldw} * (v_{t-1} + c_1(p_{best} - x_{t-1} + c_2(g_{best} - x_{t-1)}) \tag{2.13}$$

The fitness function used by the CPSO and LDWPSO in this work is product of $Sensitivity$ and $Specificity$.

2.6.12 Particle Swarm Optimization with Linear Discriminant Analysis

PSOLDA is a hybridization of PSO and LDA [128]. In this technique, PSO is used for feature subset selection and LDA for developing the model. LDA suffers from the problem of a small sample size when the dimensions of the data are much more than the number of data points. The hybridization of LDA with PSO overcome this problem as PSO can be used to select the useful features and to improve the classification accuracy of LDA [128]. PSO is a prevailing meta-heuristic technique that can maximize the performance of classification by reducing the number of features. The development of a model using PSOLDA involves data preprocessing as the first step. In the pre-processing step, normalization is done and each independent variable is scaled in the range (0,1) and instances with missing values are removed. After this, each particle in the PSO technique is representative of the solution i.e. set of independent variables. Initially, the position and velocity of each particle are random in the n-dimensional space where n is the number of independent variables. The fitness of each particle for LDA is evaluated. For each particle, local best and global best is determined by the fitness value and position vectors are updated accordingly. The fitness function for PSOLDA is classification accuracy and the parameters settings involve 400 iterations, 0.8 as a cognition learning factor, 1.2 as social learning factor,

an inertia weight of 0.5, and the number of particles are 15.

2.6.13 Genetic-Fuzzy Based Classification Techniques

A fuzzy take the form of $if\ x_i\ is\ A_i\ then\ \ y\ is\ B_i$. The if part of the rule i.e., $x_i\ is\ A_i$ is termed as rule antecedent and $y\ isB_i$ is called rule consequent. The antecedent part may have conjunctive antecedents in a fuzzy-rule. A fuzzy rule assigns a class label to an instance with a specific confidence. For a particular set of predictors, a fuzzy rule outputs the class label and a number representing the degree of confidence of the classification. The genetic algorithm is used to find a fuzzy classifier in three phases. In the first phase, a population of several fuzzy rules evolves. This phase is called as *fuzzy rule generation*. These rules may correctly classify the training instances. Out of these rules, the one with the largest coverage is added into the intermediate rule base. In the second phase, all the training instances are re-weighted based on how well they are classified according to the new rule. This process is repeated in iterations until the desired number of rules are obtained or the desired accuracy is obtained. In this thesis, three boosting techniques are used with GFS to develop hybridized models. The first technique combines LB with GFS i.e., GFS-LB. Boosting entails combining several weak classifiers to obtain a classification technique having high classification accuracy than its constituents. In GFS-LB, a logit boost boosting mechanism is employed to boost the performance of fuzzy classifiers [129]. While boosting fuzzy rules, a single fuzzy rule can be fit on a set of weighted examples. The algorithm is repetitively performed for each rule in the base. GFS-AB is also used in this thesis. In GFS-AB, the AdaBoost algorithm computes the number of votes each rule is assigned and recomputes the weight of each instance if a new rule is added to the base. In this way, AdaBoost combines low-quality classifiers with a voting scheme to produce a classifier better than any of its constituents [130]. GFS-maxLB is also used in the

thesis. This technique employs a “single winner“ method as compare to “sum of votes“ to output the label of a test example. The parameters used for GFS-LB, GFS-AB, and GFS-maxLB in this thesis are 5 labels and 25 rules in the base [131]. The fitness of a fuzzy rule is computed as the squared error between the desired output and the logistic transform of the classifier's output.

In the above GFS based classification techniques, fuzzy rules have linear representation. In GFS-GP, the classification knowledge is expressed in the form of tree-shaped genotypes [132]. The GP algorithm is similar to GA in terms of finding an efficient solution in a vast search space of candidate solutions. Both the algorithms constantly improve their solutions with the help of various operators like mutation, selection, recombination, etc. However, there is a dissimilarity in the representation of the candidate solutions for both the algorithms, while both the GA as well as the GP algorithm use chromosomes whose length is fixed throughout, in the GP algorithm the chromosomes are later represented as optimal expression trees. In this thesis, the parameter settings for the GFS-GP technique include the number of labels of 3, 8 number of rules, the population size of 30, 4 as the size of tournament and 10000 evaluations, and tree height of 8. GFS-SP uses a simulated annealing search to discover the optimum solution from fuzzy rules expressed in the form of tree-shaped genotypes [132]. The parameter settings for GFS-SP are labels of 3, 8 number of rules, the population size of 30, 4 as the size of tournament and 10000 evaluations, cross-over and mutation probability of 0.1, and tree height of 8. In GFS-GCCL an initial population is generated that is in the form of fuzzy rules [133]. Each generated fuzzy-rule is evaluated and transformed into new rules using genetic operators. This process is repeated until the number of specified iterations are not completed. The parameter setting for GFS-GCCL is as follows: population size is 100, the number of evaluations is 10000, cross-over probability is 1.0, mutation probability is 0.1 and the number of individuals to be replaced in the papulation are 20. GP-COACH is

also a fuzzy rule-based classification technique combining feature selection in the rule learning process in order to generate a compact rule base with the help of genetic programming [134]. The parameter setting of this technique are 5 number of labels, a number of generations are 20000 and an initial number of fuzzy rules is 200.

2.7 Experimental Design Framework

The diagrammatic representation of the experimental design of the thesis is presented in Figure 2.2. The various steps are described in detail in the following sections.

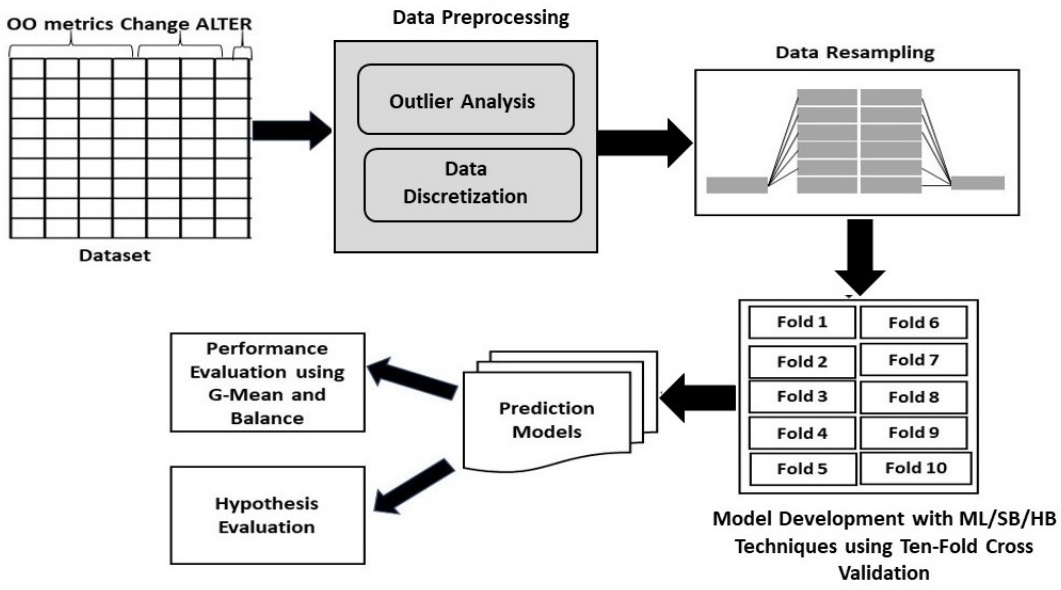


Figure 2.2: Experimental Design for Developing SMP Models

2.7.1 Empirical Data Collection

Empirical data collection is a systematic method of gathering and analyzing specific information to offer answers to questions and appraise the results. For empirical

validations, the data may be collected from industrial software, open-source software, or academic projects. This work focused on datasets extracted from the open-source software systems. These software systems are continuously modified by developers all over the world but due to lack of documentation and technical support, it is difficult to estimate their maintainability. In the area of software predictive modeling, hardly a few studies have dealt with maintainability prediction by extracting datasets from open-source software systems. Therefore, we decided to target data extraction from the open-source software systems. The source of these systems could easily be downloaded. This section describes open-source software used in the thesis for data extraction and data collection process. The source code of the open-source software systems used in this thesis has been downloaded from `https://sourceforge.net`.

2.7.2 Software Systems used for Data Collection

Empirical data is collected from Apache software systems, namely Apache Io, Ivy, Betwixt, Java caching system (Jcs), Lang, Orchestration Director Engine (Ode), and Log4j, and Byte Code Engineering Library (Bcel), HtmlUnit, Maven, and Click. The brief explanation of these software systems is as follows:

- Apache Bcel gives a very convenient way to create, manipulate, and analyze Java class files. The objects represent the classes and provide all necessary symbolic information about the corresponding classes.

- Apache Betwixt provides users a convenient way of converting the beans into Extensible Markup Language (XML) and generating digester rules. A rule is fired when a pattern is matched.

- Apache Io is a library of utilities that assist in input and output functionality. It includes utility classes, filter classes, file monitoring, and comparator classes.

- Apache Ivy is an application package to manage project dependencies. It includes tracking, recording, and resolving project dependencies and is characterized by its flexibility. Written in Java language.
- Apache JCS is a Java Caching system which tends to manage the cache data to speed up applications. It acts like a front-tier cache that is configured to maintain consistency across multiple web servers.
- Apache Lang provides various utilities like basic numerical methods, string manipulation, and serialization of different system properties.
- Apache Log4j is a fast and reliable logging package with three primary components: logger, appender, and layout. The logger captures the logging information. Appender is accountable for publishing the logging information, whereas the layout component formats the information in different formats.
- Apache Ode software package designed to talk to the web services by sending and receiving messages, handle error recovery, and data manipulation.
- HtmlUnit is Apache licensed headless browser which is intended for use in testing web-based applications.
- Click is Apache licensed a Java Enterprise Edition web application framework, providing a natural rich client style programming model.
- Maven is a software project management tool which manages project's build, documentation and reporting from one central place.

2.7.3 Data collection Procedure

The steps followed for data collection are described in Figure 2.3.

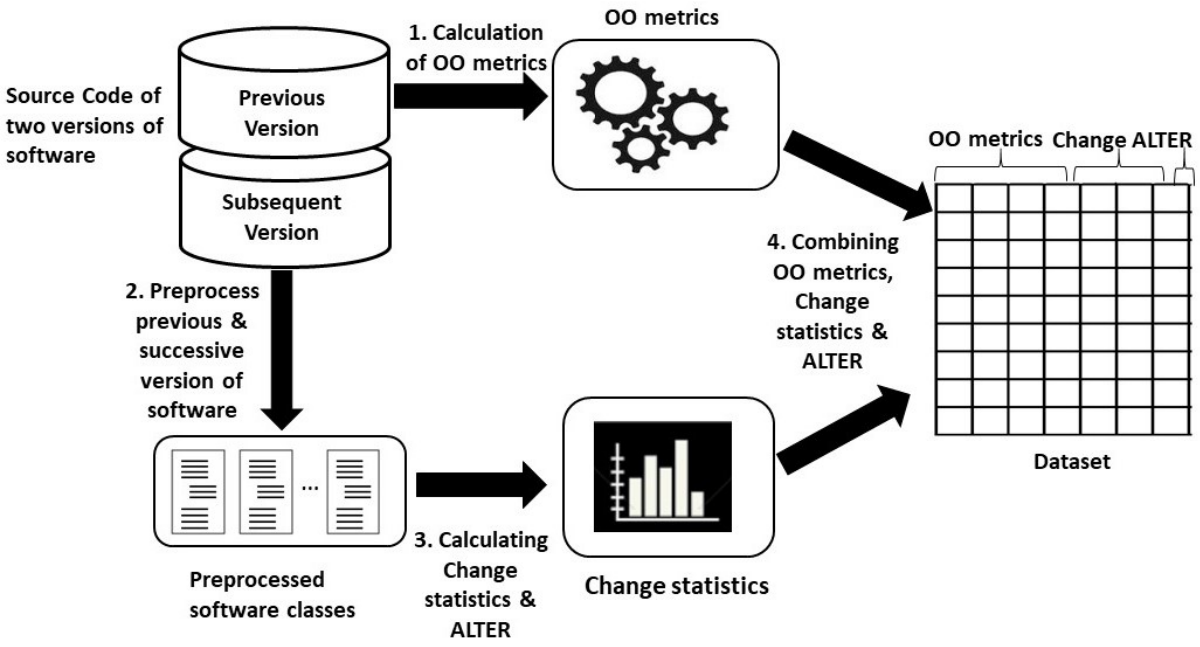


Figure 2.3: Data Collection Procedure

1. *Calculation of OO Metrics*: For data extraction, we analyzed two versions of the particular open-source software i.e., the previous version and successive version. OO metrics are extracted from the previous version of the software. The OO metrics quantify the various OO characteristics like coupling, inheritance, cohesion, etc. OO metrics are calculated with the help of the Chidamber and Kemerer Java Metrics (CKJM) tool that computes the metrics at the " method level" and "class level". The CKJM tool can be downloaded from `http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metrc.html`. The CKJM tool calculates OO metrics from the bytecode of compiled Java files. In this thesis, only "class level" metrics are considered.

2. *Prepossessing Previous and Successive Versions*: This step involved preprocessing the previous and successive versions of the software to determine the common classes in two versions. We analyzed the classes that were present both

versions. The classes that were present in the previous version but not there in the next version are not considered. Also, the classes that were newly added in the next version are not considered. In this way, we left with common classes between the two versions. We extracted these common classes and each such class represents a data point.

3. *Determining Change Statistics in the Common Classes*: In the next step, we computed the change statistic in the common classes analyzed in Step 2. The change statistic is in the form of changes in lines of source code (SLOC) in the common class. The changes in common class may be due to the addition of SLOC ($SLOC_{added}$), deletion of SLOC ($SLOC_{deleted}$), or modification of SLOC ($SLOC_{modified}$) in the recent version with respect to the previous version of the software. The total SLOC ($SLOC_{total}$) changed in a common class are calculated as the sum of $SLOC_{added}$, $SLOC_{deleted}$ and $SLOC_{modified}$ [5]. The calculation of change statistics was accomplished with the help of the Data Collection and Reporting (DCRS) tool [135]. This tool is developed by undergraduate students of Delhi Technological University in Java programming language. Data extraction has successfully been carried out from various open-source repositories using this tool. Only the prerequisite for utilizing this tool is that the repository must use GIT as version control. DCRS tool is successfully employed for data extraction corresponding to open-source software systems used in this thesis. The data extraction procedure with the DCRS tool is described below.

    - Two successive versions of the software have been fed as input to the DCRS tool. The changelog corresponding to common classes between the two analyzed versions has been extracted in the form of change records with the DCRS tool. A change record encompasses change data such as an

identifier, commit timestamp, description of the change, and a file listing of all the files which are modified including the lines of changed code. The change statistics discussed above are extracted from changelogs.

- The CKJM tool, open-source software for the extraction of OO metrics, is embedded with the DCRS tool to collect the OO metrics corresponding to the common classes. As discussed, the OO metrics extracted by the CKJM tool (`http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metrc.html`) are the quantitative measure of cohesion, coupling, inheritance, etc., of a class.
- For each common class, apart from change statistics, also the values of a special variable, ALTER are derived. The ALTER variable for a class contains two values "yes" and"no". The "yes" value of the ALTER variable represents the change in common class from the previous version to the next version. The "no" value of ALTER signifies that there is no change in SLOC in the corresponding common class. In other words, if total SLOC in a common class is greater than 0, the ALTER variable has "yes" values for that class otherwise it has "no" value.

4. *Combining OO metrics, Change Statistics and ALTER*: In the last, OO metrics computed in Step 1, change statistics and the ALTER variable derived in Step 3 are combined to form data points for each common class. The combination of these data points from the dataset for analysis in this thesis.

2.7.4 Data Preprocessing

This section describes the descriptive statistics of OO metrics of the datasets and data preprocessing steps.

2.7.4.1 Descriptive Statistics

For each dataset, the descriptive statistic of various OO metrics is reported in this section. The descriptive statistics aid in evaluating the characteristics of the datasets used for analysis. The following descriptive statistics are reported for OO metrics.

- Minimum: For an OO metric, the minimum denotes the minimum value of the corresponding metric in the dataset.
- Maximum: For an OO metric, maximum denotes the minimum value of the corresponding metric in the dataset.
- Mean: The average value for an OO metric in the dataset is depicted by Mean. The mean is computed by dividing the total value of the OO metric in the dataset with the number of data points in the dataset.
- Standard Deviation (Std.Dev.): It projects the central tendency of the OO metric. The low Std. Dev. values are an indication that metric values are close to the mean whereas high Std.Dev. values indicate that metric is dispersed.
- Median: Median provides information about the frequency distribution of the classes in a specific metric. In the case of a dataset having outliers, the median is a good means of measuring the central tendency as compared to the mean.

The descriptive statistics for each dataset are given in Appendix. After analyzing the descriptive statistics, the following observation have been made.

- The mean of the DIT metric is in the range of 0.32 to 2.64 for all datasets. Also, for all datasets, the median of NOC metric is 0.00. These statistics indicate that inheritance property was not much used in these software systems.

- Median of CBO metric for all datasets was in the range of 0.00 to 1.00. The low values of CBO metric indicate that the classes in all of the software systems are loosely coupled with each other.

- The LCOM metrics value was high up to 16920. Low LCOM values were observed in Io, Ivy, HtmlUnit, Maven and Click. Therefore, except Io, Ivy, HtmlUnit, Maven and Click other datasets have very low cohesion.

2.7.4.2 Removal of Common Classes not Changed

In this thesis, maintainability is predicted in the form of lines of code changes. Therefore, after data extraction, we removed all those classes from the datasets that were not changed from the previous version to the next release. To do this, we scanned all data points in the dataset and removed those data points in which the value of the ALTER variable was "no". We did this because the value of the ALTER variable represents a change in the common class. If the ALTER has a "no" value, it means that there is no change in terms of line of code in that class from the previous version to the next version. The detail of the software projects used in the thesis with their names, version analyzed, number of common classes, number of common classes changed and percentage of change (% Change) is given in Table 2.2.

Table 2.2: Details of Software Projects

Name of Software	**Version Analysed**	**No. of Common Classes**	**No. of Common Classes Changed**	**% Change**
Bcel	5.0-5.1	363	360	99.17
Betwixt	0.6-0.7	290	279	96.2
Io	2.0.1-2.2	274	272	99.27
Ivy	1.4.1-2.2.0	619	429	69.3
Jcs	1.2.6.5-1.2.7.9	333	219	65.76
Lang	2.4-2.6	434	267	61.52
Log4j	1.2.14-1.2.15	491	437	89
Ode	1.3.1-1.3.2	1060	1004	94.71
HtmlUnit	2.19 to 2.20	1874	656	35

Maven	3.3.3 to 3.3.9	832	275	33
Click	2.2.0 to 2.3.0	437	270	62

2.7.4.3 Outlier Analysis

We analyzed data points that were having extreme variability from the remaining data points in the dataset. Such data points are regarded as outliers. It is imperative to remove such data points for the development of an effective and unbiased prediction model. For the outlier analysis, Inter Quartile Range (IQR) available in WEKA tool [95]. IQR is computed as the difference between upper quartile (Q_3) and lower quartile (Q_1) i.e.,

$$IQR = Q_3 - Q_1 \tag{2.14}$$

A data point representing a class is considered an outlier if any one of the following conditions holds for any of the independent variable x of the class.

$$Q_3 + OF * IQR < x \leq Q_3 + EVF * IQR \tag{2.15}$$

$$Q_1 - EVF * IQR \leq x < Q_1 - OF * IQR \tag{2.16}$$

In equations 2.15 and 2.16, EVF represents Extreme Values Factor and OF represents Outlier Factor. The default values of EVF and OF in the WEKA tool are 6.0 and 3.0 respectively. The outliers were removed from each data set before further analysis. The number of data points in each dataset after removal of outliers are given in Table 2.3.

Table 2.3: Results of Outlier Analysis

Dataset	**No. of data-points**	**No. of outliers**	**No. of datapoints after outlier removal**
Bcel	360	26	334

Betwixt	279	34	245
Io	272	24	248
Ivy	429	58	371
Jcs	219	22	197
Lang	267	47	220
Log4j	437	32	405
Ode	1004	115	889
HtmlUnit	656	20	636
Maven	275	10	265
Click	270	13	257

2.7.4.4 Data Discretization

In this thesis, the dependent variable maintainability is formed by converting the continuous variable $SLOC_{total}$ into a binary variable. The new binary variable can take one of the two possible values of a class, i.e., low maintainability or high maintainability. Low maintainability classes require more maintenance efforts due to more number of changes in comparison to high maintainability classes [49]. As discussed in Section 2.5.2, maintainability is defined as a function of the number of $SLOC_{total}$ revised in a class from the previous version to the next subsequent version. The values of the dependent variable "Maintainability" for a particular data point in our analysis are assigned based on $SLOC_{total}$ based on the criteria proposed by [49]. According to this criterion, if the value of $SLOC_{total}$ for a data point is greater than average $SLOC_{total}$ values for all data points in the dataset, the Maintainability value was set to "low maintainability" or 1; otherwise, the value of Maintainability was set to "high maintainability" or 0. The detail of datasets after discretization i.e., number and percentage of data points of low maintainability (LM) and high maintainability (HM) data points for each dataset and Imbalance Ratio (IR) is shown in Table 2.4.

Here, IR is given as:

$$IR = \frac{Number\ of\ HM\ datapoints}{Number\ of\ LM\ datapoints} \tag{2.17}$$

Table 2.4: Results of Data Discretization

Dataset	**LM datapoints**	**HM datapints**	**% of LM datapoints**	**% of HM datapoints**	**IR**
Bcel	18	316	5.69	94.61	17.55
Betwixt	27	218	12.38	88.97	8.07
Io	10	238	4.20	95.96	23.80
Ivy	28	343	8.16	92.45	12.25
Jcs	27	170	15.88	86.29	6.29
Lang	27	193	13.98	87.72	7.14
Log4j	42	363	11.57	89.62	8.64
Ode	57	832	6.85	93.58	14.59

2.7.4.5 Correlation Based Feature Selection

In a predictive modeling task, the selection of appropriate features has a substantial influence on the performance of the learned model. The feature selection process eliminates the irrelevant and redundant features from the training dataset. The advantages of feature selection can incorporate a decrease in the information expected to accomplish learning, reduction in execution time, and enhanced prediction accuracy of the model. In this thesis correlation-based feature selection (CFS) [136] has been used to select the relevant features in the training datasets. CFS selects those features that have a high correlation with the dependent variable and the least correlation among themselves. A study by Hall and Holmes [137] assessed six different feature selection techniques on fifteen datasets. This study discovered that although there is no excellent technique for feature selection, the results of the CFS technique were promising. A comprehensive review of 64 fault prediction studies in the time of 1991 to 2013 was

conducted by [138]. This study revealed that CFS was one of the most commonly used feature selection techniques in software engineering predictive modeling tasks. The CFS technique has successfully applied in various software quality modeling studies [139–141], we used the CFS technique in the thesis for feature selection.

2.7.5 Data Balancing

After data discretization, we observed that there is a huge difference in terms of the number of data points of low maintainability and high maintainability classes i.e., the resultant datasets were imbalanced in nature (Table 2.4). The numbers of instances of high maintainability class were far more in number as compared to the number of instances of low maintainability class. To build effective and unbiased SMP models, it was essential to handle the imbalanced data problem. In this thesis, the imbalanced data problem is handled with the help of (i) data level techniques and (ii) algorithm level techniques. The data level techniques make a balance in the data distribution of different classes by removing or adding more data points in the dataset. Unlike data level techniques, algorithm level techniques modify the learning procedure of the learning algorithms to lessen their favoritism towards learning majority class data points. The detailed description of various techniques to handle imbalanced data is given in Chapters 4 and 5.

2.7.6 Prediction Model Development and Validation

This thesis develops SMP models to recognize the maintainability of classes in the future i.e., unseen versions of the software. The maintainability prediction models are trained using historical data to determine which class would require more maintainability effort in the future during software maintenance. The data analysis techniques described in Section 2.6 are used to develop prediction models in a supervised learning

mode. The models are developed using the training data consisting of independent variables (OO metrics) and class labels ("low maintainability" or "high maintainability"). The supervised techniques aid in learning the data points and class labels. Once a model is developed, its validation is done by providing the validation or test data to know that how the developed model would behave on the unknown data points. The validation data points comprise of independent variables only. When the test data point is given to the developed model, it is supposed to predict the class label. Later, in order to know, how accurate are the predictions of the model, the predicted label is matched with the actual label. The following subsection describes the validation methods used in this thesis.

2.7.6.1 Ten-fold Cross Validation

The ten-fold cross-validation mechanism works by dividing the training dataset into ten equal-sized partitions. The training and validation data are derived from the same project. At random, nine partitions are utilized for learning the model, and the performance of the learned model is estimated with the leftover tenth partition. This process is repeated ten times so that every partition is used at least once for testing the performance of the developed model [142]. The ten-fold cross-validation method diminishes validation bias. The ten-fold validation method is depicted in Figure 2.4.

2.7.6.2 Inter-Project Validation

Unlike the ten-fold cross-validation, in which the training and test data are used from the same project, in inter-project validation, the training data is obtained from a different project, and the model is tested on a different project dataset. The data collection for training the prediction model is one of the difficult tasks because in most cases either such data is unavailable or it is difficult to collect [143, 144]. To

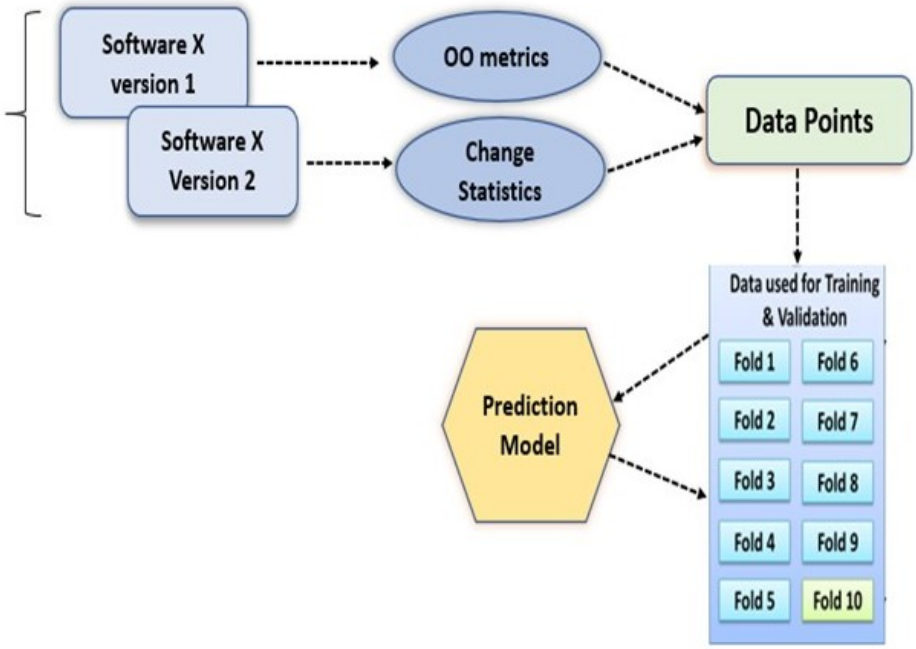


Figure 2.4: Ten-fold Cross-Validation

overcome this limitation of historical data collection, the development of generalized maintainability prediction models is necessary where the historical data of a particular project can be utilized for other similar kinds of projects. This approach is called inter-project validation. Thus, the situation in which there is the inadequacy of resources and lack of time to capture training data for the development of maintainability prediction model, inter-project validation can be employed. Figure 2.5 describes inter-project validation.

2.7.7 Performance Measures

Given training data points of classes labeled as low maintainability or high maintainability, a model can be learned from the data points and used to classify the unknown classes be of low maintainability or high maintainability The performance of the

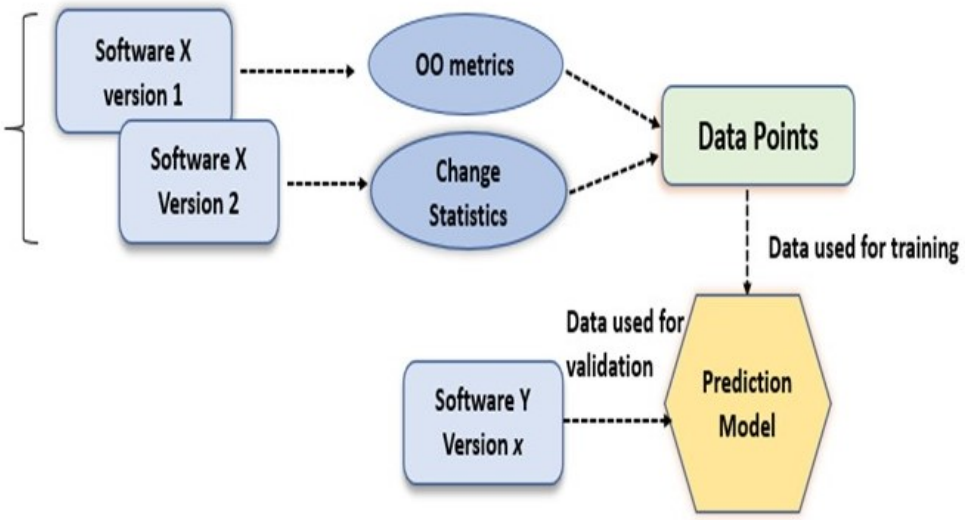


Figure 2.5: Inter-Project Validation

model is assessed by examining the confusion matrix given in Figure 2.6.

		Actual Class: Low Maintainability	Actual Class: High Maintainability
Predicted Class	Low Maintainability	True Positive (TP)	False Negative (FN)
	High Maintainability	False Positive (TP)	True Negative (TN)

Figure 2.6: Confusion Matrix

The confusion matrix contains the class values in the form of positives and negatives. In this thesis, positive value corresponds to low maintainability, and the negative value corresponds to high maintainability classes. There are four entries in the confusion matrix i.e., True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). From the confusion matrix, various performance measures to derived

to assess the performance of maintainability prediction models. These measures are defined as follows:

- *Accuracy*: It is defined as the percentage of correctly predicted low and high maintainability classes.

$$Accuracy = (\frac{TP + TN}{TP + TN + FP + FN}) * 100 \tag{2.18}$$

- *Sensitivity*: It is defined as the percentage of correctly predicted low maintainability classes amongst actual low maintainability classes. It is also regarded as the True Positive Rate (TPR).

$$Sensitivity = (\frac{TP}{TP + FN}) * 100 \tag{2.19}$$

- *Specificity*: It is defined as the percentage of correctly predicted high maintainability classes amongst actual high maintainability classes. It is also regarded as the True Negative Rate (TNR).

$$Specificity = (\frac{TN}{TN + FP}) * 100 \tag{2.20}$$

- *Precision*: It is defined as the percentage of correctly predicted low maintainability classes amongst total predicted low maintainability classes.

$$Precision = (\frac{TP}{TP + FP}) * 100 \tag{2.21}$$

- *False Positive Rate (FPR)*: It is defined as the percentage of high maintainability classes that are incorrectly predicted as low maintainability classes amongst

actual high maintainability classes.

$$FPR = (\frac{FP}{FP + TN}) * 100 \tag{2.22}$$

- *(G-Mean)*: It is defined as the geometric mean of specificity and sensitivity.

$$\text{G-Mean} = \sqrt{Sensitivity * Specificity} \tag{2.23}$$

- *Balance*: It is defined as the Euclidean distance between the pair of FPR and TPR and the optimal value of this pair. The optimal value for FPR is 0 and that of TPR is 1.

$$\textit{Balance} = 1 - \sqrt{\frac{(0 - FPR)^2 + (1 - Sensitivity)^2}{2}} \tag{2.24}$$

- *AUC*: AUC approximates the area under the Receiver Operator Characteristic curve constructed by plotting sensitivity values of the prediction model on the y-axis and FPR on the x-axis. The trade-off between the TPR and FPR by the curve serves as the model's performance. A good prediction model produces higher AUC [145].

The maintainability prediction model is considered good, if except FPR all the above-discussed metrics have high values. In the literature, the use of accuracy, sensitivity, precision, and specificity has been criticized for evaluating the models developed using imbalanced data [25, 146, 147]. Therefore, many researchers considered a few stable performance measures to evaluate the prediction models developed from imbalanced data. He and Garcia [25] advocated that G-Mean is a stable metric for assessing the prediction models developed from the imbalanced data. A robust performance

evaluator, Balance, to evaluate prediction models developed on an imbalanced dataset is given by Menzies et al. [147]. So, our study assesses the maintainability prediction models using stable and strong evaluators, namely Balance and G-Mean. The use of AUC to assess the performance of models has been advocated by Lessmann et al. [148] and Shatnawi [149]. In this thesis, the maintainability prediction models have also been developed to predict maintainability as a continuous variable i.e., the number of SLOC changed in maintenance. Those models are evaluated using the following performance metrics.

- *Magnitude of Relative Error (MRE)*: It is defined as the absolute of the relative error.

$$MRE = \frac{|Dep_a - Dep_p|}{Dep_a} \tag{2.25}$$

  Here, Dep_a is the dependent variable's actual value, and Dep_p is the dependent variable's predicted value.

- *Mean Magnitude of Relative Error (MMRE)*: It is defined as the mean magnitude of relative error over all data points in the data set.

$$MMRE = \sum_{i=1}^{N} MRE_i \tag{2.26}$$

  Here, N is the total number of data points and MRE_i is MRE value of the ith data point.

- *Prediction at level q (Pred(q))*: Pred(q) is defined as the percentage of observations where the magnitude of relative error is less or equal to q.

$$Pred(q) = \frac{K}{N} \tag{2.27}$$

In Equation 2.26, K is the total number of observations giving MRE is less than or equal to q. A specific value of q is taken, accordingly $Pred(q)$ is calculated. Commonly used values of q are 0.25 and 0.30. For instance, if q is taken as 0.25, then accordingly $Pred(0.25)$ means the percentage of observations where MRE is less than or equal to 25%.

2.7.8 Statistical Analysis

In empirical research, deriving conclusions entirely from the empirical results without the statistical analysis can be misleading [148]. Statistical tests establish confidence in the outcomes of an empirical investigation and help to validate the hypothesis formed. We apply the Friedman test [150] and the Wilcoxon signed-rank test [151] in this thesis to validate the results statistically.

2.7.8.1 Friedman Test

Friedman test allows us to compare k techniques over multiple datasets [150]. This is a non-parametric that is performed independently of any presumptions about the distribution of performance measures of k techniques over multiple datasets. The hypothesis validated using the Friedman test is stated as follow:

- *Null Hypothesis (H_o)*: Different techniques are not statistically different from each other in terms of their performance.
- *Alternative Hypothesis (H_a)*: Different techniques are statistically different from each other in terms of their performance.

The Friedman test statistic Chi-Square (χ^2) is calculated as follows:

- The performance of k techniques for a particular dataset is sorted in decreasing order. Each technique within a specific dataset is assigned a rank according to

its performance. Rank 1 is assigned to the best performing technique, whereas the worst-performing technique is allocated the lowest rank. If the performance of two or more techniques is equal, then each technique is assigned a mean rank.

- The total rank of each technique is computed by adding its rank on all datasets. The total rank assigned to each technique is represented as T_1, T_2, T_3....T_k.

- The χ^2 statistic is computed as follows:

$$\chi^2 = \frac{12}{nk(k+1)} \sum_{i=1}^{k} {T_i}^2 - 3n(k+1) \tag{2.28}$$

In equation 2.28, n signifies total number of datasets and ${T_i}^2$ denotes squared sum of ranks for i^{th} technique.

If the computed the Friedman statistic lies in the critical region, then the null hypothesis rejected and it is concluded that different techniques are statistically equal in terms of their performance. In this thesis, the Friedman test statistic is computed at a 95% level of confidence (α = 0.05).

2.7.8.2 Wilcoxon Signed Rank Test

In this thesis, the Wilcoxon signed-rank test is used for post-hoc analysis. This test is used after the results of the Friedman test are fount statistical. This test is a non-parametric test that is applied to investigate the existence of a significant difference between the pair of techniques [151]. The hypothesis evaluated with the Wilcoxon test are stated below:

- *Null Hypothesis (H_o)*: The pair of compared techniques are not statistically different in terms of their performance.

- *Alternative Hypothesis (H_a)*: The pair of compared techniques are statistically different in terms of their performance.

The following steps are performed to calculate the Wilcoxon test statistics, W.

- For each pair of performance of two compared techniques, calculate the difference score S_i.
- Ignore the sign of the difference and find a set of n absolute differences $|S_i|$ where n is the number of pairs.
- Remove all pair in which the difference score is 0 to obtain n_r pairs where $n_r \leq n$.
- Allocate ranks 1 to n_r to individual $|S_i|$ such that the minimum $|S_i|$ obtains 1 rank and the greatest $|S_i|$ allotted rank n_r. In case two or more $|S_i|$ are equivalent, assign the mean average rank of the ranks they would have been allotted independently.
- Assign sign "-" or "+" to each of the n_r ranks, based on whether S_i was initially negative or positive.
- Compute two variables V^+ and V^-. V^+ is the summation of ranks wherever the difference was positive and V^- is calculated as the summation of ranks wherever the difference is negative.
- Calculate the Wilcoxon test statistic, W, as follows:

$$W = \frac{Q - \frac{1}{4}n_r(n_r + 1}{\sqrt{(\frac{1}{24}n_r(n_r + 1)(2n_r + 1)}} \tag{2.29}$$

Here Q is the minimum of V^+ and V^-. If the W statistic is in the critical region with a specific level of significance, then we reject the null hypothesis. The rejection of

the null hypothesis means that the performance of the two compared techniques is significantly different from each other. The Wilcoxon test statistics is computed at $\alpha = 0.05$. The Wilcoxon test was performed with Bonferroni correction to remove the family-wise error. In the Wilcoxon test with Bonferroni correction, a p-value is considered significant only if it is less than b (α value divided by the total number of pairs compared).

Chapter 3

Systematic Literature Review

3.1 Introduction

Software maintenance is one of the pricey phases in the life cycle of software development that requires the attention of researchers to accurately predict the software maintainability in the earlier stages of software development to take the decisions regarding resource allocation optimally and to develop a cost effective, good quality and maintainable software. The researchers attempted to forecast software maintainability by building the relation of software metrics with maintainability. Thus, in this way, various SMP models have been built by employing statistical techniques to predict software maintainability at the initial stages of software development to decide on resource allocation. To gain insight into the work done on SMP, it is essential to summarize the existing literature. Thus, this systematic review is strived to analyze and summarize the use of datasets, software metrics, techniques, performance measures, and statistical tests in the estimation software maintainability. The following research questions are investigated in this review:

- RQ1: Which techniques have been used for SMP?

- RQ2: Which data sets have been used for developing SMP models?
- RQ3: Which metrics have been used for SMP?
- RQ4: What performance evaluators have been for SMP?
  - RQ4.1: What are the commonly used performance measures to judge the performance of SMP models?
  - RQ4.2: What validation method has been used for developing models for SMP? the method used for developing SMP models.
  - RQ4.3: Which statistical tests of significance have been used to statistically examine the predictive capability of SMP models?
- RQ5: What is the predictive ability of different techniques used for SMP?
  - RQ5.1: What is the performance of SMP models that are built with the help of ML techniques?
  - RQ5.2: What is the overall performance of SMP of models built using statistical techniques?
  - RQ5.3: What is the overall performance of SMP models developed using HB techniques?
- RQ6: Is there anyone technique that outperforms over the other in handling datasets with varying properties?
- RQ7: What are the potential threats to validity in SMP?

The review analyses various aspects of SMP models developed in the literature and also aid in the identification of research gaps to further work in the field. Based on the outcome of the systematic review, we provide directions for future work for the

researchers and software practitioners about the applicability of the prediction models for SMP.

The organization of this chapter is as follows: Section 3.2 the review protocol. Section 3.3 presents the results of the review in the form of answers to the research question and finally, section 3.4 presents the directions of future work. The results of this chapter are published in [152].

This review is carried out as per the guidelines of Kitchenham [153]. As per Kitechenham, the systematic review should consist of three stages: planning the review, conducting the review, and reporting the review. The first stage incorporates identification of the need for systematic review, framing research questions to be answered by the systematic review, development protocol to conduct the review, and its evaluation. The research questions are the objectives that we keep in mind to address the demands of the systematic review. Development of review protocol encompasses designing an appropriate strategy to search relevant studies, formulating criteria for inclusion and exclusion, the formation of quality assessment criteria, preparation of forms for data extraction, and deciding data synthesis methods. Relevant search criteria help to identify appropriate candidate studies to cater to the need for review. Based on criteria set for inclusion and exclusion, individual candidate studies are decided to be considered relevant or not relevant for the review. To assess the quality of each selected candidate study, assessment criteria in the form of the quality questionnaire have to be developed. The next phase is conducting the review, which involves the execution of a search strategy to search the relevant studies, extraction of relevant facts out of the primary studies, putting the extracted facts in data extraction forms, and data synthesis.

3.2 Review Protocol

The review protocol comprises the search strategy for the selection of relevant primary studies, inclusion and exclusion criteria, and quality assessment criteria for assessing the quality of candidate studies.

3.2.1 Search Strategy

The review aimed to select the most relevant and significant papers on the subject under study to provide concrete results of the investigation. Thus, the initial search was started by searching keywords such as “software maintainability”, “software maintenance”, “SMP”. After analyzing the studies retrieved after the initial search, we formed the following search string to select the most promising studies to be considered for review.

“Software” AND (“maintainability” OR “maintenance effort”) AND (“prediction” OR “probability”) AND (“software metrics” OR “OO metrics” OR “ procedural metrics”) AND (“classification” OR “regression”) AND (“machine learning techniques” OR “statistical techniques” OR “hybrid techniques”).

After the formation of a search string, most essential and appropriate digital libraries such as ACM Digital Library, IEEE Xplore, Science Direct, Springer Link, Wiley Online Library, Web of Science, and Scopus were used to search candidate primary studies. We searched the papers from January 1990 to October 2019. In total, 107 studies were extracted after exhaustively searching the mentioned digital libraries and by going through the reference sections of the studies extracted from the above mentioned electronic libraries. These selected candidate studies were then subjected to the inclusion and exclusion criteria described in Section 3.2.2.

3.2.2 Inclusion and Exclusion Criteria

The following inclusion and exclusion criteria were used for selecting or rejecting a study based on research questions. We got 39 studies after applying inclusion and exclusion criteria.

Inclusion Criteria

The empirical studies on SMP using the change in terms of line of code added, deleted, and modified as the dependent variable are considered for the review. The empirical studies using statistical, ML, and HB techniques for developing the SMP model were considered. The studies comparing two or more techniques for SMP were also considered for the review.

Exclusion Criteria

Studies based on dependent variable change-proneness were excluded. Also, the studies in which no empirical evidence was provided for software maintainability were not considered. The studies in which software maintainability is based on expert judgments were also not considered. The review studies and poster paper were also excluded. The studies with two similar papers of the same author in the conference and extended version in the journal, only the journal paper was considered.

Table 3.1: Quality Assessment Questionnaire

S. No.	Question	Yes	Partly	No
1	Whether the objectives of the study stated clearly?	89	11	0
2	Whether the study states the maintainability facet in a clear manner?	100	0	0
3	Is the scope of study stated clearly?	61	39	0
4	Does the study provide relevant literature?	86	8	6
5	Are the predictor variables clearly stated and defined?	100	0	0
6	Are the data collection and data preprocessing procedures described adequately?	92	8	0
7	Does the study effectively justify the importance of prediction techniques used?	50	42	8
8	Do the research questions, purposes or hypotheses stated clearly?	58	39	3
9	Are the research questions addressed clearly?	50	47	3
10	Whether the comparative analysis conducted properly?	50	44	6

11	Does the study have sufficient number of citations?	28	28	39
12	Does the results of study reported in clear manner?	61	39	0
13	Are limitations of the study stated?	42	3	55
14	Are the performance measures stated clearly ?	50	39	11

Table 3.2: Description of Primary Studies

Study#	**Name**	**Reference**	**Study#**	**Name**	**Reference**
SI1	Li (1993)	[5]	SI19	Olatunj(2013)	[154]
SI2	Khoshgoftaar(1994)	[155]	SI20	Malhotra (2014)	[45]
SI3	Khoshgoftaar (1994a)	[156]	SI21	Malhotra (2014a)	[157]
SI4	Dagpinar (2003)	[6]	SI22	Zhang (2015)	[56]
SI5	Thwin (2005)	[14]	SI23	Elish (2015)	[158]
SI6	Aggarwal (2006)	[9]	SI24	Manchanda (2015)	[159]
SI7	Koten (2006)	[13]	SI25	Mishra (2015)	[160]
SI8	Zhou (2007)	[7]	SI26	Kumar (2015)	[161]
SI9	Elish (2009)	[15]	SI27	Sharma (2015)	[162]
SI10	Wang (2009)	[8]	SI28	Kumar (2015)	[163]
SI11	Kaur (2010)	[46]	SI29	Kumar (2015a)	[164]
SI12	Jin (2010)	[50]	SI30	Kumar (2016)	[66]
SI13	Chandra (2012)	[165]	SI31	Tiwani (2016)	[166]
SI14	Dubey (2012)	[167]	SI32	Chug(2016)	[57]
SI15	Malhotra (2012)	[45]	SI33	Malhotra (2017)	[168]
SI16	Sharawat (2012)	[169]	SI34	Kumar(2017)	[170]
SI17	Jamimi (2013)	[171]	SI35	Malhotra (2018)	[172]
SI18	Kaur (2013)	[173]	SI36	Kumar (2019)	[16]

3.2.3 Quality Assessment Criteria

To provide concrete results of the review, the selected primary studies needed to be of the utmost quality. So, we designed quality assessment criteria in the form of 14 quality questions, as shown in Table 3.1. Each of the 39 selected primary candidate studies were then evaluated and ranked on 14 quality questions. Each question in the quality questionnaire was assigned scores of 1 (“yes“), 0 (“no“), and 0.5 (“partially“). The total score assigned to a study is defined as the aggregate of scores assigned to

each quality question. Thus, the highest quality score that could be given to a study was 14, and the lowest was 0. The selected primary studies were ranked for quality according to the quality score. The studies scoring greater than or equal to 8.5 were kept, while studies below a score of 8.5 were rejected. The rejected studies based on the quality score were ([174–176]). The studies scoring a score of greater than or equal to 8.5 are then sorted according to the ascending order of their publication year and listed in Table 3.2. Each primary study is assigned a unique identifier prefixed with SI (Study Identifier) for reference in the following subsections of this chapter.

3.3 Review Results

This section presents the results of the review. Figure 3.1 shows the year-wise distribution of primary studies from January 1990 to October 2019. The first study for SMP was published in 1993 by Li and Henry [5], where maintainability was taken as the dependent variable. Li and Henry proved that software maintainability could be best predicted from OO metrics. Later in 1994, two studies were published by Khoshgoftaar and Szabo [155, 156] on maintainability prediction. It has been observed that most of the work on SMP was started from the year 2003 when researchers agreed that maintainability could be predicted as the number of changes made to code during the maintenance period and software design metrics are helpful for quality prediction at early stages of software development life cycle. From 2003 onward, researchers evaluated the effectiveness of various statistical, ML, and HB techniques for maintainability prediction.

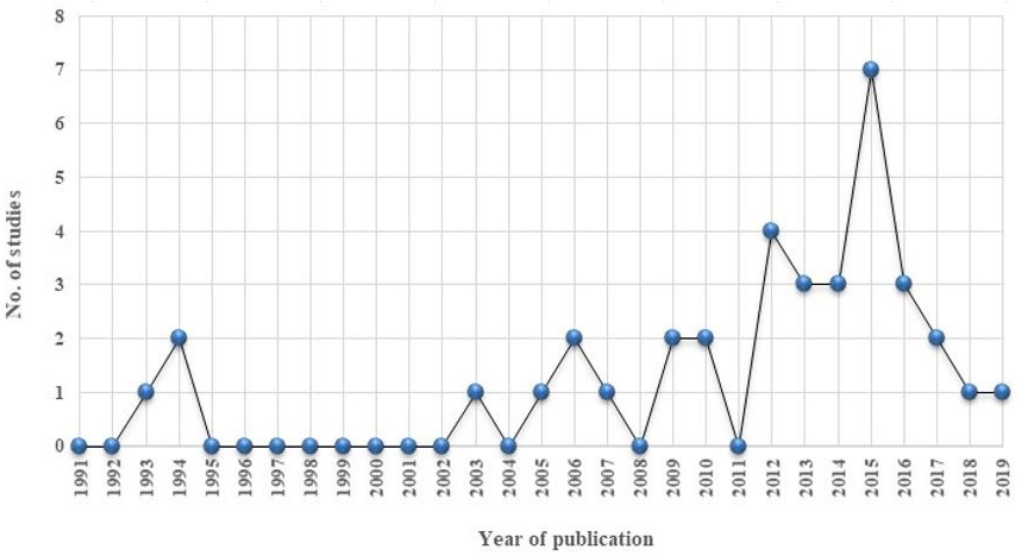


Figure 3.1: Year-wise Distribution of Primary Studies.

3.3.1 Results Specific to RQ1

To accurately predict the maintainability of software, various techniques have been used by researchers and practitioners in the last two decades. These techniques tend to establish the relationship between predictor and response variables to build maintainability prediction models. We classify techniques used for SMP into the following three broad categories: (i) Statistical techniques, (ii) ML techniques, and (iii) HB techniques. We divide the ML techniques used for developing prediction models into the following categories:

- Neural Networks
- Decision Trees
- Bayesian Networks
- Support Vector Machine
- Instance-Based Learning

- Rule-Based Learning
- Ensembles
- Fuzzy Rule Learning

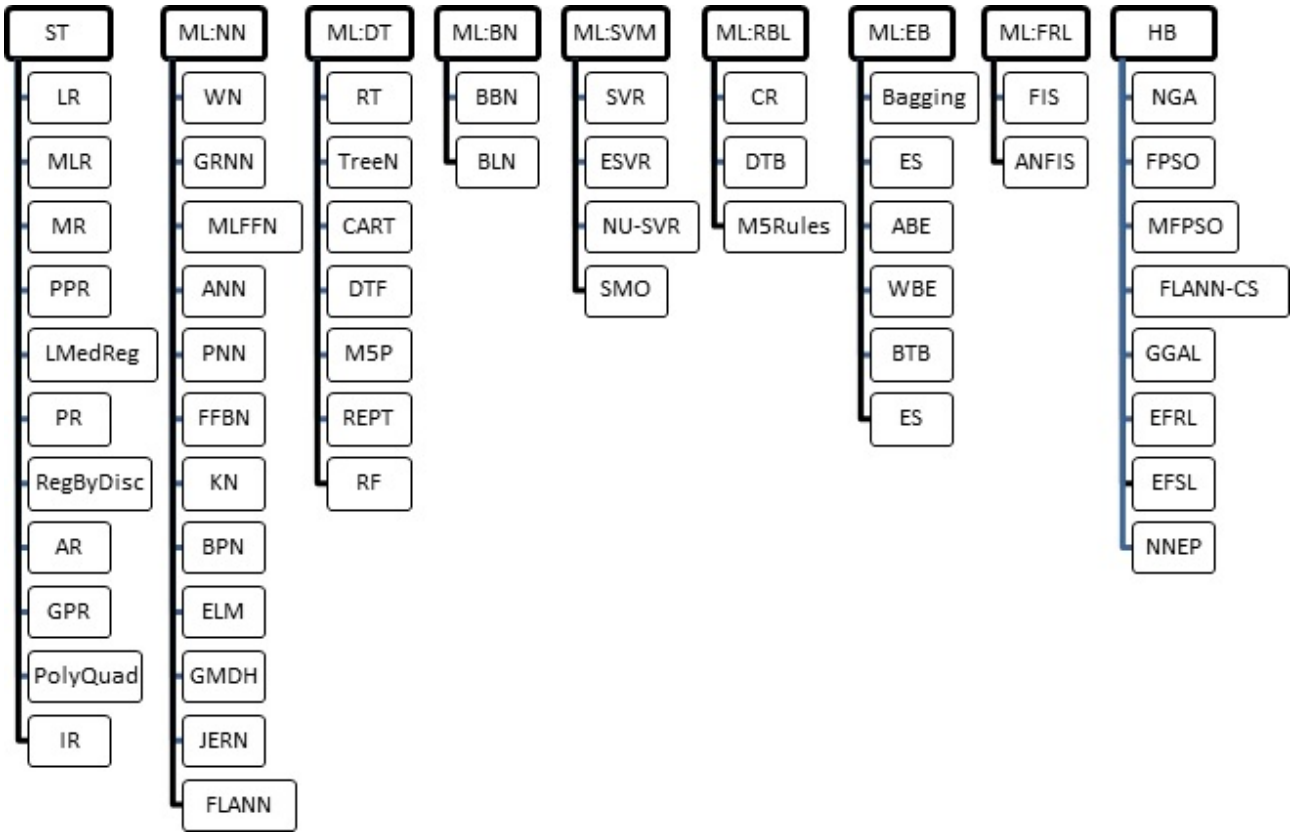
The taxonomy of various techniques used for SMP is presented in Figure 3.2.

Distribution of studies according to the above-mentioned techniques used for SMP along with study identifier is presented in Table 3.3.

Table 3.3: Distribution of Studies according to various Techniques

Technique used	**Study Identifier**	**Percentage**
Statistical techniques	SI1, SI4, SI10	08
Only ML techniques	SI2, SI5, SI6, SI9, SI11, SI12-SI16, SI20, SI23, SI24, SI25, SI27, SI32	44
HB techniques	SI21, SI26, SI28, SI29, SI30,SI33, SI34, SI36	20
Statistical and ML techniques	SI3, SI7, SI8,SI17, SI18, SI19, SI22, SI31, SI35	25
Statistical, ML & HB techniques	SI21, SI33	05

It is evident from Table 3.3 that the majority of studies (44%) applied ML techniques for building maintainability prediction models; statistical techniques are used in 8% of the studies, whereas HB techniques are used in 20% of the studies. 25% of the studies have applied ML and statistical techniques combined, whereas 5% of the studies have applied all three types of techniques combined. 11 primary studies applied more than one technique to compare the performance of a technique over the other one. We found ML techniques to be the most prominent techniques used for SMP. Among the primary studies using ML techniques, NN was the frequently used category, which was explored in 50% of the studies followed by SVM and DT, which were used in 26% and 27% of the studies, respectively. Further, studies using different categories of ML techniques for SMP have applied different variants of each category. The most widely used algorithm in each category of the ML technique along with the



WN: Ward Network, GRNN: General Regression Neural Network, MLFFN: Multilayer Feed Forward network, ANN: Artificial Neural Network, FFBN: Feed Forward Back Propagation Network, MLP: Multilayer Perceptron, PNN: probabilistic Neural Network, KN: Kohan Network, BPN: Back Propagation Network, ELM: Extreme Learning Machines, GMDH: Group Method of Data Handling, JERN: Jordon Elman Recurrent Network, RT: Regression Tree, TreeN: TreeNet, CART: Classification and Regression Tree, DTF: Decision Tree Forest, M5P: Quinlan's M5 algorithm, REPT: Reduced Error Prone Tree, BBN: Baysian Belief Network, BLN: Baysian Learning Network, SVR: Support Vector Regression, ESVR: Epsilon Support Vector Regression, NU-SVR: NU- Support Vector Regression, SMO:Sequential Minimal Optimization, CR: Conjunctive Rules, DT: Decision Table, ES: Ensemble Selection, ABE: Average Based Ensemble, WBE: Weight Based Ensemble, BTB: Best in Training Based Ensemble, FIS: Fuzzy Inference System, ANFIS: Artificial Neuro Fuzzy Inference System, LRN: Linear Regression, MLR: Multiple Linear Regression, MR: Multivariate Regression, PPR: Project Pursuit Regression, LMedReg: Least Median of Squares Regression, PR: Pace Regression, RegByDisc: Regression by Discretization, AR: Additive Regression, GPR: Gaussian Process Regression, PolyQuad: Polyquadratic, IR: Isotonic Regression, NGA: Neuro-Genetic, FLANN: Functional Link Artificial Neural Network, FPSO: Functional Link Artificial Neural Network with Particle Swarm Optimization, MFPSO: Modified Functional Link Artificial Neural Network Particle Swarm Optimization, GGAL: GRNN with Genetic Adaptive Learning, FLANN-CS: Functional Link Artificial Neural Network-Clonal Selection, EFRL: Evolutionary Fuzzy Rule Learning, EFSL: Evolutionary Fuzzy Symbolic Learning, NNEP: Neural Network with Evolutionary Programming.

Figure 3.2: Taxonomy of Techniques used for SMP

study identifier is presented in Table 3.4.

Table 3.4: Most Popular Algorithm from Various Categories of ML Techniques

ML Technique	Popular algorithm	Study Identifier
NN	GRNN	SI5, SI14, SI18, SI20, SI32
DT	RT	SI7, SI8, SI19
BN	BLN	SI17, SI19
SVM	SVR	SI12, SI13, SI17, SI23, SI31, SI32
IBL	KSTAR	SI18, SI22, SI24, SI32

GRNN is one of the most popular techniques in the category of NN for developing SMP models. The strengths such as the ability to deal with sparse data, and to model nonlinear functions, and ability to work well in a noisy environment made GRNN one of the most popular techniques in NN category. The primary studies that used statistical techniques for maintainability prediction, three studies applied linear regression (SI22, SI31, SI32), and another three studies applied multiple linear regression (SI3, SI7, SI19). Eight SMP studies (SI21, SI26, SI28, SI29, SI30, SI33, SI34, SI36) explored the use of HB techniques. The HB techniques for SMP combined ML and evolutionary or SB techniques as a single technique. Out of eight studies using HB techniques for SMP, four studies reported the use of NGA in which NN is used for learning, and GA is used for optimizing the weights of NN. The study SI21 has applied EFRL, EFSL, and NNEP techniques to develop SMP models. NGA is used is SI26, SI28, and SI29. The HB techniques, namely FLANN, Adaptive FLANN, FPSO, and MFPSO, are used in SI30, whereas SI36 applied FLANN.

3.3.2 Results Specific to RQ2

This RQ determines various datasets that have been used in the literature for developing SMP models. The datasets used for SMP are categorized as follows:

- *Public datasets:* In this category UIMS and QUES datasets were found to be

the most widely used datasets for SMP. These datasets were collected by Li and Henry during the three years of maintenance of these two systems. Thus, UIMS and QUES datasets together are called Li&Henry data set [5]. This data set has been used in 67% (SI1, SI5, SI6, SI7, SI8, SI9, SI10, SI11, SI13, SI14, SI15, SI16, SI17, SI18, SI19, SI23, SI25, SI26, SI28, SI29, SI30, SI33, SI34, SI36) of the primary studies selected for this review. Since these datasets are public, the result obtained using these datasets in various studies are easy to verify and replicate.

- *Proprietary Software:* These kinds of datasets include datasets collected from software developed by industrial professionals. Only 11% (SI2, SI3, SI4, SI20) of the selected primary studies have used proprietary software for SMP. Some of the proprietary software projects include File Letter Monitoring (FLM), Student Management System (SMS), Inventory Management System (IM System), Angel Bill Printing (ABP) System.

- *Open-source projects:* Open-source projects are used in 19% of the studies (SI21, SI22, SI24, SI27, SI31, SI32, SI35) selected for this review. Some of the open-source projects include Apache Poi, Apache Rave, Hodoku, ORDrumbox, jTDS, Art of illusion, etc. Most of these open-source projects were downloaded from software repositories: Sourceforge.net and Github.

- *Student projects:* This type of data set includes academic projects developed by students. Only one study (SI12) has dealt with such kind of data set.

Figure 3.3 shows the dataset used and the corresponding percentage of studies in which these datasets are used.

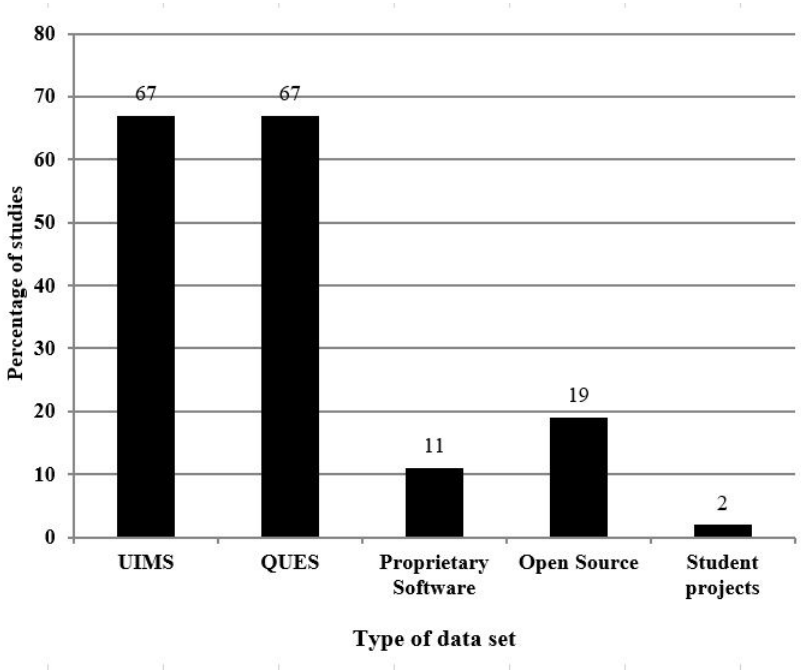


Figure 3.3: Types of Datasets used for SMP

3.3.3 Results Specific to RQ3

Various software metrics have been given in the literature to characterize and quantify various aspects of the software system like size, complexity, modularity, complexity, etc. In the software engineering predictive modeling, the usefulness of these metrics has been very well proved. For SMP, the applicability of these lies in their ability to act as predictors or independent variables. To answer RQ3, we classified the selected primary studies based on software metrics used as a predictor or independent variable. A similar kind of classification is given by Malhotra [138] for software fault prediction. We found selected studies using both static software code metrics and dynamic metrics as independent variables for maintainability prediction. The categories of software metrics based on primary studies are given below:

- *Studies using traditional software metrics as independent variables:* These

studies used traditional (procedural) software metrics as predictor variables given by Halstead [33]. These metrics include size metrics: lines of source code (SLOC), number of unique operands, number of unique operators, etc. Studies using traditional metrics are SI2, SI3.

- *Studies using OO software metrics as independent variables:* This category of studies used OO metrics as independent variables. Such metrics quantify the characteristics of OO systems like modularity, inheritance, polymorphism, etc. Studies using OO metrics are SI1, SI4, SI5, SI6, SI7, SI8, SI9, SI10, SI11, SI12, SI13, SI14, SI15, SI16, SI17, SI18, SI19, SI20, SI21, SI23, SI25, SI26, SI28, SI29, SI30, SI32, SI33, SI34, SI35, SI36.

- *Studies using composite software metrics as independent variables:* In this category, we identify two types of studies as follows: (i) studies using both traditional and OO metrics as predictors (ii) studies using both OO metrics and dynamic metrics as predictors. The first category includes the studies that used OO metrics at the class level and traditional metrics at the method level. Studies using such a combination of metrics are SI22 and SI24. The second category includes studies that used OO metrics and dynamic metrics combined as predictors. Such studies are SI27 and SI31. The distribution of studies according to the kind of metrics used is depicted in Figure 3.4.

  Amongst 83% studies using OO metrics, 86% of studies used the combination of Chidamber&Kemerer (C&K) and Li&Henry metric suite, 7% of studies solely used C&K metrics, and in remaining studies, C&K and Li&Henry metrics are used with another metric suite like Henderson-Seller [86], Lorentz & Kidd [38], and Briand [3]. Studies using a combination of C&K, Li&Henry, and other OO metric suite are SI4 and SI32. Thus, C&K and Li&Henry metric suite have been observed as one of the most popularly used OO metric suite for SMP.

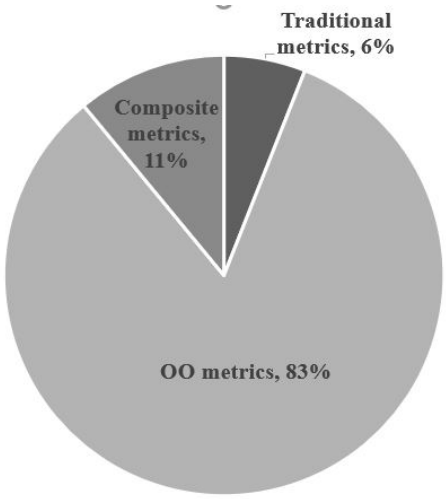


Figure 3.4: Distribution of Studies according to Metrics used

3.3.4 Results Specific to RQ4

To assess the effectiveness of the software maintainability predictor model, it is essential to examine it along with performance evaluators. This section describes the performance evaluators like performance measures, validation methods, and statistical tests used in literature to evaluate the effectiveness of SMP models.

Performance Measures (RQ4.1)

The performance of maintainability prediction models can be judged with the help of various performance measures. In the literature, multiple measures are used by the researchers to evaluate the performance of maintainability prediction models developed using different techniques discussed in Section 3.4.1. The definitions of the most frequently used performance measures, along with the study in which they are used given in Table 3.5.

Table 3.5: Performance Measures used in Literature Studies

Performance Measure	Description	Study Identifier
Mean absolute error (MAE)	It is defined as the mean of the absolute difference between the actual and predicted value of the dependent variable.	SI2, SI3, SI5, SI14, SI24, SI26, SI27, SI31, SI32, SI36
Magnitude of Relative Error (MRE)	It is defined as the ratio of the absolute difference between the actual and predicted value of the dependent variable to the actual value of the dependent variable.	SI6, SI7, SI8, SI11, SI12, SI13, SI15, SI17, SI18, SI19, SI20, SI21, SI23, SI26, SI28, SI29, SI30, SI33, SI35
Mean magnitude of relative error (MMRE)	It is defined as the mean of MRE's over all the data points.	SI6, SI7, SI8, SI9, SI11, SI12, SI13, SI15, SI17, SI18, SI19, SI20, SI21, SI23, SI26, SI28, SI29, SI30, SI33, SI34, SI35, SI36
Maximum Magnitude of relative Error (maxMRE)	It measures the maximum of MRE value over all the data points.	SI7,SI18,SI21,SI33
Root Mean Square Error (RMSE)	It is defined as the square root of the average of the sum of squares of all of the errors.	SI10, SI18, SI24, SI31, SI32, SI36
Pred(30)	It is defined as the percentage of predictions in which MRE is less than or equal to 30%.	SI7, SI8, SI9, SI15, SI17, SI18, SI19, SI20, SI21, SI23, SI33, SI35
Pred(25)	It is defined as the percentage of predictions in which MRE is less than or equal to 25%.	SI7, SI8, SI9, SI15, SI17, SI18, SI19, SI20, SI21, SI32, SI33, SI35
Coefficient of Regression (R-Square)	It measures the closeness of the data point to the fitted regression line.	SI1, SI4, SI5, SI8, SI14, SI15

Figure 3.5 shows the percentage of studies using the performance measure given in Table 3.5. As shown in Figure 3.5, MMRE is the most commonly used performance measure, which is used in 61% of the primary studies.

The other commonly used performance measures are Pred(25) and Pred(30), which are used in 33% and 33% of the studies, respectively. Few performance measures like Adjusted R-square, Sum of squares of residuals, Mean square of residuals, Minimum absolute error, Maximum absolute error, absolute residual error, normalized root

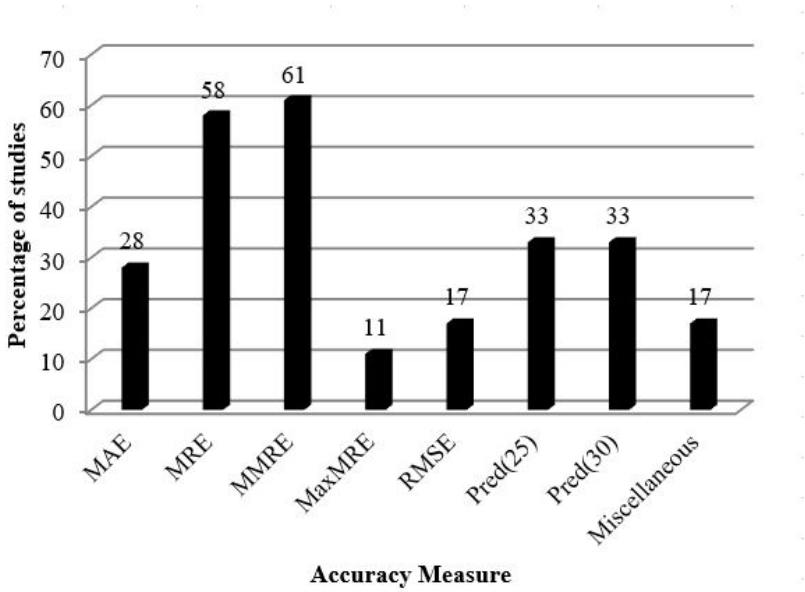


Figure 3.5: Distribution of Studies according to Performance Measures

mean square error, standard error of the mean, True error, an estimate of true errors, Spearmen Rank correlation coefficient, Relative Absolute Error (RAE) are clubbed into the miscellaneous category as these were less commonly used.

Validation Methods (RQ4.2)

The selection of a cross-validation method is essential to develop a prediction model effectively. Cross-validation methods tend to build a prediction model by dividing the available data set into two parts, namely: training data and test data. The training data set is used to train the model, while the test data set is used to evaluate the performance of the developed model.

For SMP, three types of cross-validation methods have been used in literature: K-Fold cross-validation, Hold-out cross-validation, and Leave-one-out. The brief description of these methods, along with the study identifier in which these methods have been used, is given in Table 3.6.

Table 3.6: Description of Cross-validation methods used in Selected Primary Studies

Method	Description	Study Identifier
K-Fold cross validation	In this cross-validation method, the data set in hand is divided into K partitions each of equal size. In each iteration K-1 partitions are used to train the model while one out of K partitions is used to test the model. This process is repeated K times. The commonly used values for K were 5 and 10.	SI5, SI7, SI23, SI24, SI26, SI28, SI29, SI30, SI31, SI32, SI35, SI36
Hold-out cross validation	In Hold-out cross-validation data set is splitted into two partitions; one partition is used for training purpose and another for validation. The commonly employed partitions methods were 60-40, 70-30 and 75-25.	SI15, SI19, SI22, SI27
Leave-one-out cross validation	This validation method involves dividing data set with N observations into two partitions of size N-1 and 1. In each iteration random N-1 data points are used for training and one data point is used for testing.	SI8, SI9, SI10, SI18, SI34

Figure 3.6 presents the distribution of primary studies by the cross-validation menthod used.

Only 21 studies reported the validations methods used. K-Fold cross-validation was used in 12 studies followed by hold-out and leave one out validation, which were used in 4 and 5 studies, respectively.

Statistical Tests (RQ4.3)

The use of the statistical test in predictive modeling strengthens the outcome of the developed models by statistically examining the conclusion drawn regarding the predictions given by the developed model. The use of statistical tests has been observed in 28% of the primary studies selected for this review. The distribution of

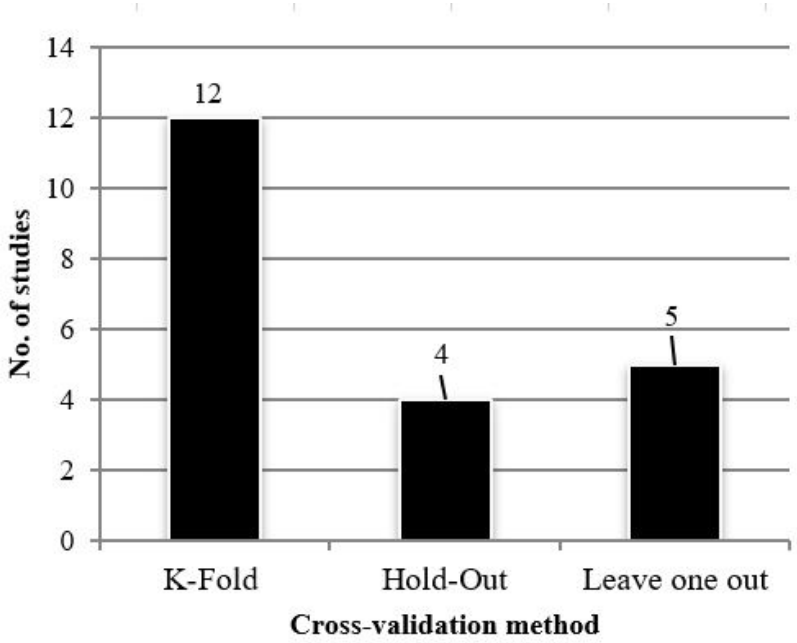


Figure 3.6: Distribution of Studies according to Cross-validation Methods

the statistical test used is depicted in Figure 3.7. Three studies used t-test (SI23, SI30,

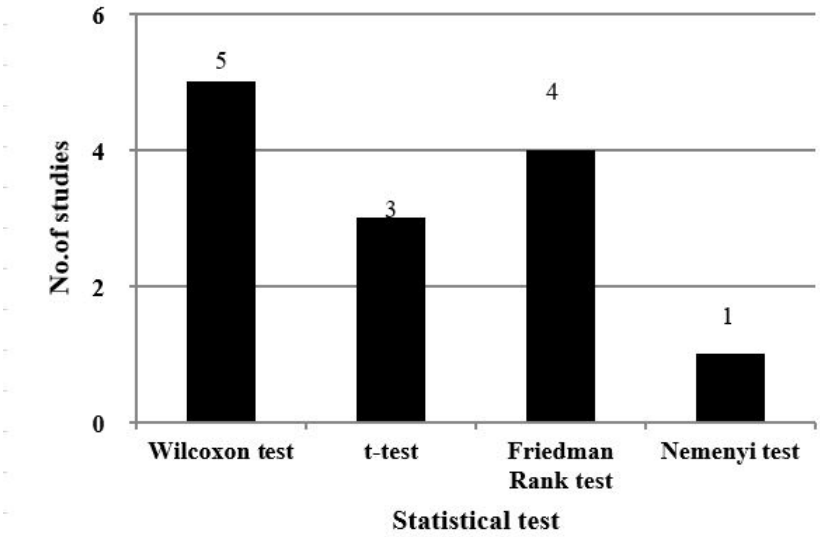


Figure 3.7: Distribution of Studies according to Statistical Tests

S36); five studies used the Wilcoxon test (SI7, SI8, SI18, SI27, SI35) for comparative statistical analysis. Four studies (SI18, SI31, SI32, SI35) used the Friedman ranking test, and the Nemenyi test is used in one study (SI32). It is evident that the use of non-parametric tests such as the Wilcoxon test and Friedman test has been found in more number of studies as compared to parametric t-test; the reason is that parametric test requires data normality of the underlying data set to be true before the application of the test. In studies SI7, SI8, and SI33, the Wilcoxon test has been used to statistically prove the significant difference between the predictive performances of different techniques. In the study SI18, the Wilcoxon test is used as a post-hoc analysis after the Friedman test. Nemenyi the test is also used as a post-hoc analysis after the application of the Friedman test in study SI32.

3.3.5 Results Specific to RQ5

This section presents the predictive capabilities of various techniques used for developing SMP models. To ensure the usefulness of a technique for developing a prediction model, it is extremely important to evaluate its predictive capability. The predictive performance of various techniques used for SMP is evaluated by summarizing the values of the performance measures described in Section 4.4. We report the predictive performance of techniques that have been used in at least two primary studies to get more generalized results. The following subsections describe the predictive performance of ML, statistical, and HB techniques.

Performance of SMP Models developed with ML Techniques (RQ5.1)

To evaluate the performance of ML techniques for SMP, we recorded the MMRE and Pred(25) values of different ML techniques on different data sets. We analyze the predictive performance of only those ML techniques that are used in at least two primary studies so that generalized results can be reported.

Table 3.7: Performance of SMP Models Developed using ML Techniques

Technique	Count	Performance Measure	Minimum	Maximum	Mean	Std. Dev.
GRNN	9	MMRE	0.32	0.54	0.43	0.06
	10	PRED	0.26	0.78	0.59	0.15
ANN	3	MMRE	0.26	1.95	0.93	0.89
	2	PRED	0.15	0.37	0.26	0.15
FFBN	3	MMRE	0.45	0.51	0.48	0.03
MLP	4	MMRE	0.59	1.95	1.16	0.63
	2	PRED	0.13	0.37	0.25	0.16
PNN	7	MMRE	0.37	0.53	0.42	0.05
	8	PRED	0.51	0.73	0.62	0.06
KN	7	MMRE	0.35	0.56	0.46	0.07
	7	PRED	0.48	0.73	0.69	0.08
GMDH	9	MMRE	0.21	0.42	0.33	0.06
	9	PRED	0.51	0.79	0.68	0.08
RT	5	MMRE	0.49	1.53	0.91	0.54
	6	PRED	0.10	0.41	0.26	0.11
M5P	4	MMRE	0.54	1.67	1.00	0.56
	2	PRED	0.28	0.32	0.30	0.02
BN	4	MMRE	0.45	0.97	0.71	0.30
	4	PRED	0.39	0.44	0.41	0.03
SVM	9	MMRE	0.22	0.67	0.45	0.14
	8	PRED	0.31	0.65	0.46	0.10
KSTAR	9	MMRE	0.27	0.77	0.52	0.14
	9	PRED	0.36	0.72	0.53	0.10
Std. Dev. indicates Standard Deviation						

MMRE and Pred (25) are selected for the analysis as these are the most commonly used performance measures, as discussed in Section 3.4.4. We present the descriptive statistics such as minimum, maximum, mean, and standard deviation of performance measures: MMRE and Pred (25) in Table 3.7. A good maintainability prediction model should have low MMRE and high Pred(25) values. The MMRE and Pred values in Table 3.7 are presented after removing the outliers to minimize the prediction bias as it was observed that some techniques perform exceptionally different on some data sets resulting in very biased predictions. To overcome this bias, outlier analysis was

essential, and we used boxplots to gain insights about the outliers. These outliers were removed before reporting the statistics of performance measures MMRE and Pred(25).

Figures 3.8 and 3.9 show the results of outlier analysis done with the help of boxplots for the two prominent performance measures, namely MMRE and Pred(25), respectively.

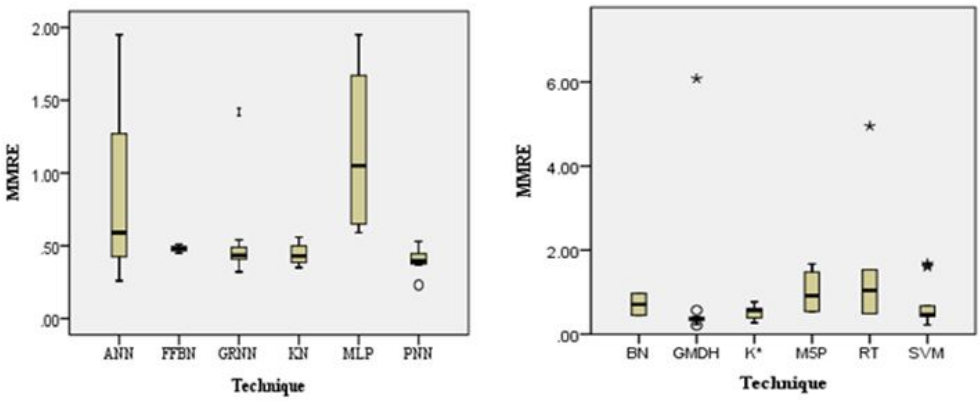


Figure 3.8: Performance of ML techniques in terms of MMRE

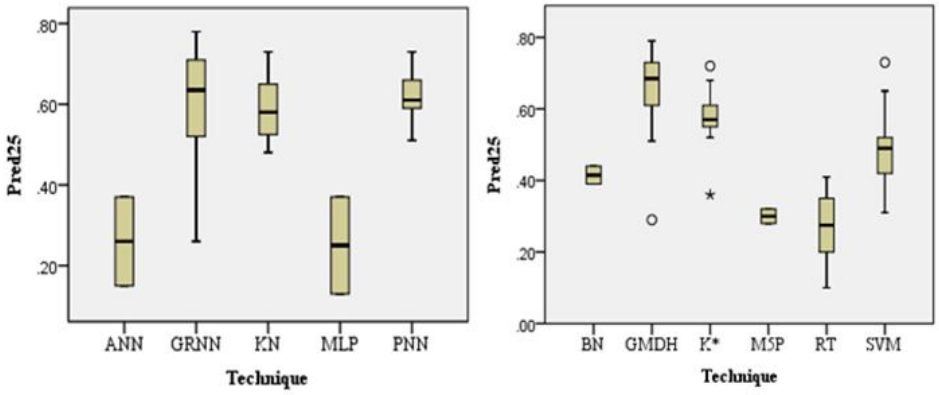


Figure 3.9: Performance of ML techniques in terms of Pred(25)

As shown in Table 3.7, the GMDH technique reported the best mean MMRE of 0.33, followed by PNN and GRNN, which gave mean MMRE of 0.42 and 0.43,

respectively. It is to be noted that the mean MMRE of GMDH, PNN, GRNN, SVM, KN, KSTAR, and FFBN ranged from 0.33 to 0.52. KN has reported the best mean Pred(25) value of 0.69 followed by GMDH with a Pred(25) value of 0.68. Again as shown in Table 3.7, mean Pred(25) values of GMDH, PNN, GRNN, SVM, KN, KSTAR, and FFBN ranged from 0.46 to 0.69. Therefore, it is quite evident that the performance of GMDH, PNN, GRNN, SVM, KN, KSTAR, and FFBN with respect to MMRE and Pred(25) is quite acceptable. MLP performed worst with a mean MMRE of 1.16 and a mean Pred(25) value of 0.25. Thus, the predictive capability of ML techniques: GMDH, PNN, GRNN, SVM, KN, KSTAR, and FFBN, is close to an acceptable level as per Conte et al. [177] and Wen el al. [178].

Performance of SMP Models developed with Statistical Techniques (RQ5.2)

Statistical techniques are used in 14 primary studies (SI1, SI3, SI4, SI7, SI8, SI10, SI17, SI18, SI19, SI21, SI22, SI31, SI32, SI34) to develop software maintainability, prediction models. Out of studies that used statitical techniques for SMP, 11 compared statistical and ML models. To get generalized results about the performance of models developed using statistical techniques, we evaluated the results of statistical techniques, which were used in at least two primary studies. The results were evaluated based on performance measures MMRE and Pred(25). We found LRN and MLR as the prominent statistical techniques used for SMP. The performance of models developed using LRN and MLR techniques in is recorded in Table 3.8 after outlier analysis using boxplots.

Table 3.8: Performance statistics of SMP Models using Statistical Techniques

Technique	Count	Performance Measure	Minimum	Maximum	Mean	Std. Dev.
LRN	7	MMRE	0.4	0.66	0.53	0.09
	7	PRED	0.54	0.63	0.59	0.03
MLR	6	MMRE	0.39	2.70	1.50	1.17
	6	PRED	0.15	0.42	0.28	0.12
Std. Dev. indicates Standard Deviation						

Figure 3.10 shows the result of the outlier analysis of LRN and MLR techniques for performance measures MMRE and Pred(25). It is evident from Table 3.8 that MMRE of prominent statistical techniques ranges from 0.53 to 1.50, and Pred(25) ranges from 0.28 to 0.59. On comparing the performance statistics of statistical techniques (Table 3.8) with ML techniques (Table 3.7), it may be noted that ML techniques: GMDH, KN, GRNN, PNN, SVM, FFBN, and KSTAR, outperform statistical techniques LRN and MLR.

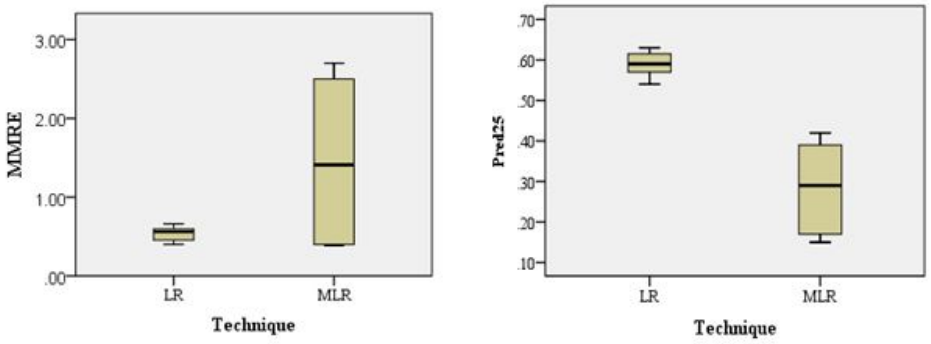


Figure 3.10: Performance of Statistical techniques in terms of MMRE and Pred(25)

Performance of SMP Models developed with HB Techniques (RQ5.3)

The HB techniques such as FPSO, NGA, MFPSO, FLANN-CS, EFRL, EFSL, and NNEP are used in the studies literature studies for developing models to predict software maintainability. For instance, SI21 has applied EFRL, EFSL, and NNEP techniques to develop software maintainability, prediction models. NGA is used is SI26, SI28, and SI29. HB techniques, namely FLANN, Adaptive FLANN, FPSO, and MFPSO, are used in SI30, whereas SI36 applied FLANN, the category of HB techniques, to develop prediction models of maintainability. In total, eight selected primarily studied and used HB techniques for SMP. Unlike ML and statistical techniques, cumulative performance statistics of each HB technique could not be reported as

each study has applied the different techniques to predict maintainability. It has been recorded that MMRE of HB techniques in different studies ranged from 0.22 to 2.1, whereas Pred(25) ranged from 0.19 to 0.64. Thus, by looking at these performance statistics of HB techniques, no valid conclusion about their applicability for SMP could be drawn.

3.3.6 Results Specific to RQ6

To get an insight about which technique is best in terms of its performance on datasets with differing properties, we record the performance (in terms of MMRE) of the best techniques on different datasets from different primary studies and present results in Table 3.9.

Table 3.9: Performance of ML Techniques on different Datasets

Best Technique	Dataset	MMRE	Best Technique	Dataset	MMRE
GRNN	QUES	0.41	KSTAR	Log4j	0.39
GRNN	FLM,EASY	0.54	GMDH	QUES	0.21
GRNN	Drumkit	0.38	GMDH	Jedit	0.38
GRNN	OpenCV	0.42	GMDH	Junit	0.37
GRNN	Abdera	0.45	GMDH	FLM,EASY	0.35
GRNN	Ivy	0.41	GMDH	Drumkit	0.23
GRNN	Log4j	0.32	GMDH	OpenCV	0.37
GRNN	Jedit	0.49	GMDH	Abdera	0.32
GRNN	QUES	0.41	GMDH	Ivy	0.34
FFBN	UIMS	0.51	GMDH	Log4j	0.37
FFBN	QUES	0.48	KN	Drumkit	0.56
FFBN	FLM,EASY	0.45	KN	OpenCV	0.48
PNN	Drumkit	0.38	KN	Abdera	0.37
PNN	Jedit	0.41	KN	Ivy	0.40
PNN	Ivy	0.53	KN	Log4j	0.35
PNN	OpenCV	0.42	KN	Jedit	0.43
PNN	Abdera	0.38	KN	Junit	0.52
PNN	Log4j	0.37	SVM	Junit	0.54
PNN	junit	0.47	SVM	Student Project	0.22
KSTAR	Abdera	0.55	SVM	Log4j	0.39

KSTAR	Jedit	0.56	SVM	Drumkit	0.64
KSTAR	Ivy	0.38	SVM	QUES	0.36
KSTAR	junit	0.61	SVM	Jedit	0.48
KSTAR	UIMS	0.56	SVM	Abdera	0.47
KSTAR	QUES	0.27	SVM	Ivy	0.42
KSTAR	Drumkit	0.67	SVM	OpenCV	0.67
KSTAR	OpenCV	0.77			

From Table 3.9, it is very much clear that the GMDH technique shows lower MMRE on the majority of the diverse datasets as compare to other best techniques: GRNN, PNN, KSTAR, KN, FFBN, and SVM. On the majority of the datasets, MMRE of the GMDH technique ranged from 0.21 to 0.38, which is quite close to the acceptable MMRE as suggested by [177]. Thus, it is quite conclusive that the GMDH technique is emerged as the best technique in terms of its predictive performance irrespective of the datasets compare to other best techniques depicted in Table 3.9.

3.3.7 Results Specific to RQ7

For any empirical study, the threats to the validity of the results of the study are quite important concern as the threats limit the applicability of the results. After the analysis of the results of the empirical study, it is tremendously essential for the researchers to list down the validity threats of the results.

Table 3.10: Classification of Threats to Validity Reported by Primary Studies

Threats to external validity			
S. No.	**Threat**	**Threat description**	**Study Identifier**
1.	Applicability of results across languages.	The study considered the data set extracted for software systems written in a particular programming language.	SI13, SI15, SI20, SI27, SI29, SI30, SI33, SI35, SI36

2.	Dataset of inadequate size.	The size of the software system evaluated in the study is quite low which affects the result generalizability.	SI17, SI18, SI27, SI33
3.	Application of results across different variables.	Independent variables are collected from the software systems that have specific characteristics and results cannot be generalized.	SI20, SI34, SI36
4.	Result bias because techniques used.	The study applied only a few techniques and the effect of other prominent techniques on the same data set need to be judged.	SI15, SI27, SI29
5.	Software system used have specific characteristics.	Dataset is extracted from the software system with specific characteristics, and results cannot be generalized.	SI32, SI36
Threats to internal validity			
6.	Confounding effect of independent variables.	The study does not account for the impact of variables on maintenance other than those selected in that particular study.	SI15, SI29, SI30, SI32
7.	Influence of human factors.	The study does not account that maintenance also depends on external quality attributes like developerâs competence, familiarity with code, etc.	SI21, SI36

Only 39% of selected primary studies (SI13, SI15, SI17, SI18, SI20, SI26, SI27, SI29, SI30, SI32, SI33, SI34, SI35, and SI36) for this systematic review reported the threats to the validity of results. The threats to validity reported by the selected primary studies are classified into two types, namely threats to external validity and threats to internal validity as per classification given by [179]. The classification of threats to validity reported by primary studies is given in Table 3.10. Out of 39% studies that reported threats, the threat to the applicability of results across the different programming languages and small size of data set were reported by the majority of

the studies. Only a few studies (SI18 and SI32) have reported the threat mitigation measures of some of the threats. Study SI18 lessened the threat of small size of the data set by employing leave one out cross-validation. Study SI32 eliminated the threat of generalizability of results by taking seven open-source software systems with different sizes, different characteristics, and maintenance requirements.

3.4 Discussion and Future Directions

This systematic review aimed to investigate the SMP models from various perspectives such as software metrics used, data set used, techniques employed, performance measures used, validation methods employed, and statistical test employed. We evaluated the performance of models developed using various techniques where maintainability is measured as the number of lines of code changed during the maintenance period and described as the continuous dependent variable in terms of the lines of code added, deleted, and modified. After an exhaustive search and going through rigorous quality assessment, 36 studies are selected for this review to answer the eight research questions. These studies have been published in the period from January 1990 to October 2019. The information discovered as a result of this review is discussed as follows:

- RQ1: ML, statistical, and few HB techniques have been applied in developing models for SMP. ML techniques are found to be the most prominent techniques applied for SMP in the studies we analyzed and are divided into Neural Networks, Decision trees, Bayesian Networks, Support Vector Machine, Instance Based Learning, Rule-Based Learning, Ensembles, and Fuzzy Rule Learning.
- RQ2: Data sets used for SMP include public data sets, proprietary software, open-source projects, and student projects used in only one study. Public data

sets UIMS and QUES were found to be the most widely used data sets in the literature.

- RQ3: The majority of the studies (83%) have used OO metrics as the predictor variables; 6% of studies used procedural metrics, and the remaining 11% of the studies have used composite metrics. Among 83% of the studies using OO metrics, 86% have used C&K and Li&Henry metric suite combined. Thus, C&K and Li&Henry metrics emerged as prominent metric suite used for SMP.

- RQ4: The majority of studies have used MMRE and Pred(25) performance measures. Performance measures such MAE, RMSE, and normalized root-mean-square error (NRMSE) have also been used in selected primary studies to evaluate the performance of developed models. K-fold cross-validation with K=10 is employed in most of the studies as a validation method, and leave-one-out validation and hold-out validations are used in a few studies. The use of statistical tests was observed in only 27% of the studies. The nonparametric Wilcoxon test and Friedman rank test were commonly used statistical tests.

- RQ5: Mean MMRE of ML techniques GMDH, KN, GRNN, PNN, SVM, KSTAR, and FFBN ranged from 0.33 to 0.52. Mean Pred(25) values of techniques GMDH, KN, GRNN, PNN, SVM, KSTAR, and FFBN ranged from 0.46 to 0.69. Thus, performance measures MMRE and Pred(25) of these techniques indicate quite an acceptable level of performance of these ML techniques to develop models to predict software maintainability. The LRN and MLR techniques are found to be commonly used statistical techniques for developing maintainability prediction models. Overall mean MMRE values of these techniques ranged from 0.53 to 1.50, and mean Pred(25) values ranged from 0.28 to 0.59. These performance statistics are indicative of the poor performance of statistical techniques. As the use of HB techniques for developing prediction

models for maintainability is concerned, different HB techniques such as FPSO, MFPSO, NNEP, and EFRL are applied in different primary studies. Thus, the cumulative values of MMRE and Pred(25) for HB techniques could not be reported. However, as per values of extracted from different studies, MMRE ranged from 0.22 to 2.1, whereas Pred(25) ranged from 0.19 to 0.64. According to these statistics, no meaningful conclusions about the overall performance of HB techniques for SMP could be drawn.

- RQ6: GMDH technique outperformed all techniques applied for SMP on different data sets with varying properties. On different data sets, the MMRE of this technique ranged from 0.21 to 0.37.

- RQ7: Very few studies have reported threats to the validity of their empirical results, which we classified into threats to internal validity and threats to external validity. It has been observed that the inadequate size of data sets used for experimentation and result bias maybe because of the technique used; data sets extracted from software systems having specific characteristics are of major concern.

After taking into account the result discussions specific to each RQ, We propose the following future directions:

- It has been observed that in the majority of the selected primary studies for this review, public data sets UIMS and QUES have been used. The size of this data set is very small. Future work should focus on carrying out more empirical studies on SMP on industrial data sets, and for this, it is a critical need for the availability of these datasets to the research community. Also, researchers can use varying-sized data sets extracted from open-source software for SMP.

- Majority of studies in the literature have analyzed OO metrics at the class level. More studies should be performed which evaluate OO metrics at the method level.

- This review has revealed that the overall performance of models developed using the ML technique is more as compared to models developed using statistical techniques. Few statistical techniques like PPR gave a good performance but were excluded from analysis due to lack of a uniform framework for experimentation. Thus, apart from exploring more techniques, future work should focus on the development of a uniform experimental scheme for developing prediction models for software maintainability.

- It was observed that a majority of studies used ML techniques and these techniques are effective in the domain of SMP. However, more studies should be conducted which assess and compare the effectiveness of more ML techniques for SMP. Also, researchers should explore the use of ensemble techniques for developing SMP models.

- It has been reflected through this review that few HB techniques were used for SMP. Hence, the generalized results about the applicability of these techniques could not be reported as these techniques were used in very few studies. Future work should focus on applying these techniques on diverse data sets to establish their applicability in SMP.

- Few of the selected primary studies reported the threat of result bias as the software system from which datasets have been extracted had got specific properties. Thus, to reduce this threat, future studies should focus on cross-project validation or inter-project validation.

- The results indicate that a majority of studies did not use any statistical tests for

verifying the obtained results. Hence, future studies should statistically evaluate the significance of their results. The researchers should take into account the various possible threats to validity while designing their experiments to yield effective and reliable results.

- The review has revealed that very few studies have applied ensembles learners to develop prediction models for software maintainability. More studies should be carried out to examine the performance of ensembles for SMP.

Chapter 4

Software Maintainability Prediction using Machine Learning Techniques by Handling Imbalanced Data

4.1 Introduction

Predicting software maintainability is in the initial stages of software development is an impending area in software engineering. It focuses on the design and development of prediction models to forecast software maintainability when the software is in the early development stages. The knowledge about low maintainability classes in advance helps to allocate the limited resources of an organization optimally to these classes. It results in good quality and highly maintainable software developed within the time and budget. In the literature, researchers [5, 8, 13, 57] defined software maintainability in the form of the lines of source code changed during the period of maintenance to correct faults and advocated that the software maintainability has a

strong correlation with the OO metrics describing various software characteristics such as inheritance, coupling, and cohesion. More the changes encountered in a class in the maintenance phase means more maintainability effort is required for that class and vice versa [49]. The ultimate goal of developing these models is to predict those software classes accurately that have low maintainability. The low maintainability classes are critical for any project because these classes must be tested cautiously to decrease the probability of occurrence of faults. Also, such classes should need to be well-documented to augment understandability to carry out future maintenance activities. The dataset used for training the maintainability prediction models should consist of sufficient instances of high and low maintainability classes to train the model effectively. However, in reality, during maintenance, few software classes demand complex interventions, resulting in more changes in the code lines i.e., of low maintainability. Therefore, there is an imbalance among the number of instances of the classes having low (minority class) and high maintainability (majority class), resulting in an imbalanced dataset. It is challenging to train the prediction models to predict the unseen data points of these classes with reasonable accuracy using imbalanced data.

Therefore, this chapter deals with the development of effective software maintainability models by treating imbalanced datasets. Identification of low maintainability classes is crucial as these classes need more attention during the software maintenance and testing phase as such classes are likely to be sources of defects and future advancements. However, with the imbalanced datasets, ML techniques encounter enormous trouble [27, 180, 181], and the prediction models obtain higher prediction accuracy just for the majority class rather than those for both of the classes. SMP models developed using imbalanced data do not have any practical significance as they may misclassify the minority class (low maintainability) instances. Thus, such misclassification may lead to improper resource allocation to the misclassified classes

resulting in poor quality software products.

In the software engineering domain, the imbalanced class problem is addressed to build competent models to predict faulty and change-prone classes [68, 69, 76]. However, the imbalanced data problem has not been addressed for SMP. Therefore, to treat the imbalanced data problem in SMP, in this chapter we handle the imbalanced data using various data resampling techniques, including oversampling, undersampling, and hybrid resampling, before learning the SMP models using ML techniques to improve their performance. ML techniques have been successfully applied in various other domains of software engineering like defect prediction [84, 87, 138], effort prediction [178, 182], and change prediction [183]. Different ML techniques work differently and may yield contrasting results on many software datasets. Thus, it is important to evaluate a number of ML techniques for the task of maintainability prediction. Also, as noted in Chapter 3, more studies are essential which compare the effectiveness of ML techniques for SMP. This chapter has specifically following objectives:

- To construct SMP models to predict low maintainability classes by treating the imbalanced datasets with data resampling techniques.
- To assess the predictive performance of the developed SMP models and validate them statistically.
- To investigate the improvement in the predicting performance of the built SMP models after data resampling.

To achieve the above-specified objectives, the following research questions are addressed in this chapter.

- RQ1: What is the performance of SMP models developed using ML techniques on original imbalanced datasets?

- RQ2: What is the performance of SMP models developed using ML techniques after balancing the datasets with data resampling techniques?
- RQ3: Which data resampling technique improves the performance of the prediction models the most?

In the interest of answering the above research questions, we build up SMP models that use OO metrics as predictors and software maintainability as the outcome. The datasets extracted from eight open-source software packages are used to develop SMP models with the application of ML techniques (C4.5, MLP-CG, RBFNN, IRBFNN, Bagging, AdaBoost, KNN, LR, and KSTAR). The stable performance metrics, Balance and G-Mean, are used in the study to evaluate the predictive performance of the models. Also, the statistical analysis of constructed models has been carried out to strengthen the conclusions.

This chapter is organized as follows: Section 4.2 states the research background. Section 4.3 describes the research methodology. Section 4.4 discusses the results of the chapter in the form of answers to RQ's. Section 4.5 states the key findings of the chapter. The results of this chapter are published in [184].

4.2 Research Background

This section states the research background of this chapter.

4.2.1 Independent and Dependent Variables

In this chapter, OO metrics from C&K [36], QMOOD [41], Henderson-Sellers [86], and Martin [31] metric suites are used as an independent variable for developing the SMP models. The detailed description of the OO metrics is given in Chapter 2 (Section 2.5.1). The dependent variable analyzed in this chapter is “Maintainability“.

4.2.2 Datasets

We used eight open-source software projects (Bcel, Betwixt, Io, Ivy, Jcs, Lang, Log4j, Ode) in this chapter for developing SMP models. The details of the datasets i.e., version analyzed, number of common classes, number of common classes changed and percentage of change (% Change) is given in Chapter 2 (Table 2.4).

4.2.3 Data Resampling

As discussed in Chapter 2 (Section 2.7.4) there is an imbalance in the data points of low maintainability and high maintainability data points for each dataset, therefore, we use data resampling to provide effective training data to ML techniques for developing SMP models. The data resampling techniques modify the training dataset in such a manner that it includes enough quantity of data points of the minority and the majority class [27]. The data resampling techniques applied in this chapter include oversampling, undersampling, and hybrid resampling. In the oversampling techniques, the new data points of the rare or the minority class are produced so that the dataset contains the proportionate number of instances of the minority and majority class. The undersampling techniques work by expelling a few data points of the majority class to make a proportionate dataset. Hybrid resampling combines the oversampling and undersampling strategy [185].

4.2.4 Model Development and Validation

This phase uses ML techniques for model development. The developed models are validated either using ten-fold cross-validation. The performance of the models is assessed using G-Mean and Balance measures. Furthermore, statistical analysis is also performed to evaluate the performance of different data resampling techniques

for handling imbalanced data.

4.2.5 Hypothesis Evaluation using Statistical Tests

This section describes the hypothesis evaluated in this chapter by using statistical tests.

Hypothesis for Friedman Test

The hypothesis for the Freidman test were framed to appraise whether using different data resampling techniques improve the performance SMP models or not. The Friedman test is used to evaluate the null hypothesis (H0, H1) and alternate hypothesis (H2, H3) in RQ2. These hypotheses evaluate the performance of SMP models using G-Mean and Balance.

Null Hypothesis (H0 / H1): There is no significant difference in the predictive performance of SMP models developed with original imbalanced datasets and after applying data resampling techniques performance measures G-Mean and Balance.

Alternate hypothesis (H2 / H3): There is a significant difference in the predictive performance of SMP models developed with original imbalanced datasets and after applying data resampling techniques with respect to performance measures G-Mean and Balance.

Hypothesis for Wilcoxon test

The Wilcoxon test with the Bonferroni correction is used as a post-hoc analyzer to evaluate the pairwise significance difference among the performance of best technique R and other examined data resampling techniques when the Friedman test returned significant results. The hypothesis evaluated with the Wilcoxon test was the null hypothesis (H4 / H5) and alternate hypothesis (H6 / H7).

Null Hypothesis (H4 / H5): SMP models developed using various ML techniques do not give a significantly improved performance with respect to G-Mean and Balance

when data resampling technique R is used as an alternative of other data resampling technique S for handing the imbalanced datasets.

Alternate Hypothesis (H6 / H7): SMP models developed using various ML techniques give a significantly improved performance with respect to G-Mean and Balance when data resampling technique R is used as an alternative of other data resampling technique S for handing the imbalanced datasets.

Here, by data resampling technique S, we mean all other data resampling techniques applied to balance the datasets in the chapter excluding the best-ranked data resampling technique R as per the Friedman test results. For example, if SafeSMOTE (data resampling technique R) is the best-ranked resampling technique as per the results of the Friedman test, then S corresponds to the remaining resampling techniques investigated in the chapter.

4.3 Research Methodology

The ML techniques used in the chapter include C4.5, MLP-CG, RBFNN, IRBFNN, Bagging, AdaBoost, KNN, LR, and KSTAR. These techniques are discussed in detail in Chapter 2 (Section 2.6). For simulation KEEL tool (`http://keel.es/`) was used with default the default parameter settings of the techniques. This section briefly describes the data resampling techniques used in the chapter to handle the imbalanced data.

4.3.1 SMOTE

SMOTE is an oversampling technique. It oversamples the minority class by engendering synthetic data points for it. The artificial data points are created by interpolating the line that joins a randomly picked data point of the minority class and its k-nearer

neighbors. The amount of oversampling depends on neighbors chosen for interpolation [28]. For instance, for an oversampling rate of 100%, one nearer neighbor, n, of a minority data point, is randomly selected. By interpolating n with the minority class data point, a synthetic data point is generated, which is then added to the dataset. For exampling, a training dataset of 10000 instances with 500 data points of the minority class and 9500 data points of majority class, 100% oversampling using SMOTE generates 500 synthetic data points corresponding to the minority class. So, the training dataset after 100% oversampling with SMOTE would have 1000 data points of the minority class, i.e., 500 original minority class data points and 500 synthetic data points.

Let X_i be a minority class data point, R_i be the randomly selected neighbor of X_i, and $rand$ be a random number between 0 and 1, and S_i be the synthetic data point to be generated. D_i is the difference between the selected minority class data point X_i and its randomly selected nearer neighbor R_i. The synthetic minority class data points would be created with the help of the following equations:

$$D_i = R_i - X_i \tag{4.1}$$

$$S_i = X_i + rand * D_i \tag{4.2}$$

The synthetic data points S_i generated using the above equations are added into the original training data.

4.3.2 BSMOTE

Unlike SMOTE, BSMOTE does not create synthetic data points for all the data points of the minority class. It only oversamples the minority class data points that lie at the borderline. To find the borderline minority data points, the nearer neighbors of

the minority data point X_i are determined. If the all nearer neighbors of X_i are the majority class data points X_i is thought-out noise and it is not oversampled i.e., no synthetic data points are created corresponding to it. Also, if in the nearer neighbors of a X_i, the majority class data points are less in number compared to minority class data points, X_i is considered as safe and it is not oversampled. However, if the nearer neighbors of X_i, the majority class data points are more in number compared to the minority class data points, X_i is considered borderline. In this case, the synthetic data points corresponding to X_i are generated in the same manner as SMOTE. So, in this way, for oversampling, the emphasis of the BSMOTE is only on the borderline minority data points [186].

4.3.3 SafeSMOTE

SafeSMOTE is another variant of SMOTE. As BSMOTE synthesizes only the borderline minority data points, the resultant dataset after oversampling still may contain too few data points of the minority class compared to the majority class. Instead of only oversampling the borderline minority data points, SafeSMOTE oversamples the minority class data point X_i as per its Safe-Level ratio. For a minority data point X_i Safe-Level ratio is given as a ratio of Safe-Level of X_i to that of nearer neighbor's Safe-Level [187]. The safe level of a data point is given as the number of minority class data points in its KNN. The focus of the SafeSMOTE technique is to generate synthetic data points in safe regions.

4.3.4 Adasyn

Adasyn adaptively generates synthetic examples corresponding to the minority class data points i.e., unlike SMOTE, Adasyn automatically calculates the number of synthetic samples to be generated. It employs the weighted density distribution to

decide the number of synthetic data points to be created corresponding to each the the minority class example [188]. Density distribution is the measure of weights that are assigned to each the minority class data point by their difficulty level of learning. Many examples are created corresponding to harder-to-learn cases, and a smaller number of instances are generated, corresponding to easy-to-learn examples. The synthetic samples are generated by adopting the same technique as that of SMOTE.

4.3.5 SMOTE-TL

SMOTE-TL technique is an oversampling technique that works in two-phases namely (i) oversampling and (ii) identification and removal of Tomek links [101]. i.e., SMOTE-TL follows a hybrid data resampling approach. In the first phase, the dataset is balanced by introducing synthetic data points into it. The synthetic data points are generated by adopting the SMOTE procedure. All the generated synthetic data points are added to the training dataset. After this phase, the Tomek links from the training dataset are identified and removed. Tomek links are described as follows. Consider two data points X and Y of two different classes. Let the distance between X and Y be $d(X, Y)$. A pair (X, Y) pair is named a Tomek link if there is not a data point Z, such that $d(X, Z) < d(X, Y)$ or $d(Y, Z) < d(X, Y)$. If data points form Tomek link, then either one of these data points would be a noisy or both data points are borderline. After the identification of Tomek links, the data points forming the link are removed from the training dataset.

4.3.6 SMOTE-ENN

SMOTE-ENN is a hybrid data resampling technique. In this technique SMOTE is used to oversample the dataset by creating synthetic data points corresponding to the minority class. On the oversampled dataset, Wilson's edited nearer neighbor algorithm

is applied which removes those data points from the training dataset that satisfy the editing rule [189]. As per the editing rule, if the label of a data point differs from the labels of the majority of its neighbors, such data points are removed from the training dataset.

4.3.7 SPIDER

SPIDER technique preprocesses the imbalanced data in two distinguished phases [190]. The first phase consists of segregating all the data points in the training data into safe and noisy categories for which the KNN algorithm is used. The data points that are correctly classified using KNN are placed into a safe category and are assigned labels safe. The data points that are incorrectly classified by the KNN algorithm are assigned labels noisy. In the second phase, the data points belonging to the minority class are oversampled using three oversampling options namely, weak amplification, weak amplification with relabelling, and strong amplification. In the weak amplification, the minority class data points that are labeled as noisy are oversampled by creating their exact copies. The number of copies to be created for a noisy minority class data points depend upon the number of safe data points in 3-nearer neighbors of noisy the majority class data points. The weak amplification with relabelling extents weak amplification with relabelling step. In this method of amplification, firstly the noisy minority class data points are oversampled using weak amplification. Afterward, noisy data points from the majority class situated in the 3-nearest neighbors of noisy data points from the minority class are relabelled by changing their class from the majority class to the minority class. The strong amplification option considers both safe and noisy data points from the minority class. With strong amplification, firstly the safe data points are amplified using a weak amplification process as discussed above. After this algorithm is switched to treat noisy data points from the minority class. The noisy

data points from the minority class are reclassified using an extended nearer neighbor method i.e., using a 5-nearer neighbor. If a data point is classified correctly using extended neighbors, it is oversampled by adding as many of its copies as there are safe data points in the majority class in 3-nearest neighbors. But, if a data point is incorrectly classified again, it is oversampled by creating a number of copies that are equal to the number of safe data points from the majority class in its 5-nearest neighbors. In this thesis another variant of SPIDER i.e., SPIDER II is also used [191]. Though, SPIDER as discussed above instinctively oversamples the minority class data points without considering the changes that take place in the dataset on account of relabelling. SPIDER II considers the changes made due to the relabelling option. The minority class data points are flagged as safe or noisy and based on those changes. After the identification of noisy examples, SPIDER II oversamples the minority class using the relabelled dataset.

4.3.8 ROS and RUS

ROS balances class distribution by replicating the minority class data points randomly. ROS may increase the occurrence of overfitting as it balances the dataset by creating replicated the data points of the minority class. The balanced dataset after ROS when presented to the classification algorithm, generated rules may be very accurate, but cover the replicated data points [189]. On the other hand, the RUS technique balances the distribution of data points by elimination data points of the majority class randomly. The major limitation of RUS is that this technique can remove possibly useful data points that could be significantly important for training the prediction model [189].

4.3.9 CNN and CNN-TL

CNN is a heuristic undersampling technique that removes only the redundant data points of the majority class. It avoids the limitation of RUS that blindly remove the majority class data points and may suffer potential loss of valuable data points. This technique finds a consistent subset S from the overall training data. The subset S is said to be consistent with T if using S, we can correctly classify the data points of T. The CNN technique works as follows: In the initial step all the minority class data points and a random majority class data point are extracted from T and are placed in set S. After this, the remaining data points of T are classified using the subset S using 1-NN. If a data point is misclassified in due course, it is moved from T to S. This process is repeated for all data points present in set T [192]. CNN-TL is the extended version of CNN. In this approach after finding the consistent subset S from T, all the Tomek links from S are removed [189].

4.3.10 NCL

NCL is an undersampling technique that removes the majority class data points from the imbalanced dataset using the ENN rule. As per ENN rule, if the class of a data point differs from that of the class of at least two of its three nearest neighbors, such data points are removed from the dataset. NCL technique slightly modifies the ENN rule [193]. This algorithm undersamples the training dataset as follows: For each data point D_i in the training set, its three nearest neighbors are determined. If D_i belongs to the majority class and D_i is incorrectly classified using its three nearer neighbors, then D_i is removed. However, if D_i is a minority class data point and it is misclassified by its three nearer neighbors, then the nearer neighbors of D_i belonging to the majority class are removed.

4.3.11 CPM

CPM is an unsupervised way of balancing the imbalanced dataset. The algorithm calls itself recursively [194]. In the initial iteration, two data points one corresponding to minority class and others from the majority class are randomly selected as initial cluster centers. The remaining data points are then partitioned into two clusters, C1 and C2, using the selected centers. The algorithm calls itself recursively until the stopping criteria are not met i.e., the algorithm stops when the class impurity of either of the clusters becomes less than that of its parent's impurity. Here, the impurity of a cluster is defined as the proportion of the minority class data points in the corresponding cluster.

4.4 Results and Analysis

This section discusses the experimental results which evaluate the use of data resampling techniques to deal with imbalanced data for developing SMP models using ML techniques.

4.4.1 Results Specific to RQ1

The predictive capability of various techniques in the domain of SMP is assessed by analyzing the performance of ten-fold cross-validation models developed using them. In this chapter, we first developed SMP models with original imbalanced datasets using ML techniques. Table 4.1 show the predictive performance of ML techniques for SMP models based on G-Mean and Balance. As shown in Table 4.1, ML techniques without applying resampling techniques have inferior performance regarding G-Mean and Balance. On analyzing Table 4.1 it has been noted that in 61% of the cases,

G-Mean values are less than 50%. Similarly, in 66.66% of the cases, the Balance values are less than 50%.

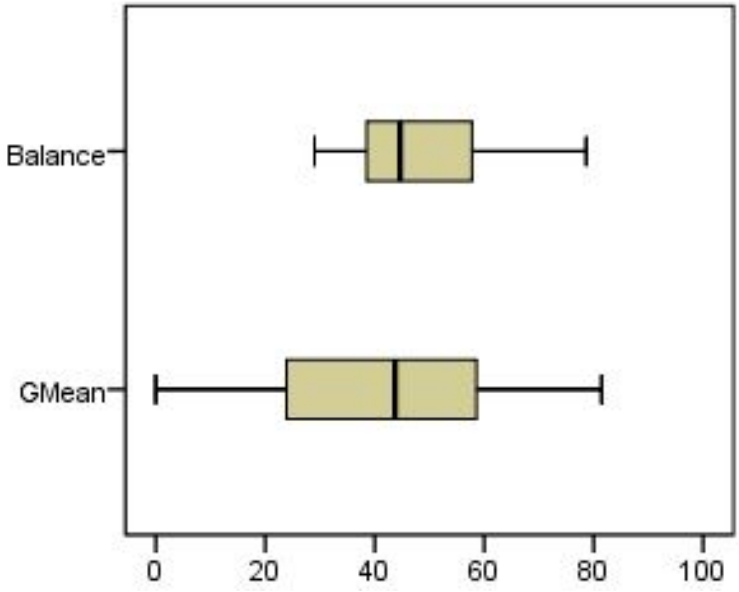


Figure 4.1: Performance of SMP Models on Imbalanced data

Figure 4.1 shows the boxplots for the performance of maintainability prediction models regarding Balance and G-Mean in case of imbalanced data. It is evident from Figure 4.1 that G-Mean values are even 0% for a few of the cases, whereas Balance has 29% as its lowest value. Also, the median of G-Mean and Balance for all ML techniques is approximately 40%. These trends of the poor performance of ML techniques for maintainability prediction are because the datasets are imbalanced in nature and the prediction model is unable to learn the minority class instances properly, i.e., for training the model, very few minority class instances are presented to the classifier. Therefore, such kind of prediction models cannot be utilized for making future predictions for unknown instances.

Table 4.1: G-Mean and Balance Results on Imbalanced Datasets

Dataset	C4.5	AdaBoost	IRBFNN	KNN	KSTAR	Bagging	LR	MLP-CG	RBFNN
G-Mean									
Bcel	46.46	46.46	51.86	22.66	0.00	46.77	23.27	51.52	40.70
Betwixt	18.62	36.59	51.74	35.93	0.00	0.00	46.27	55.30	45.49
Io	29.00	43.37	43.39	57.55	29.29	29.29	44.06	43.34	57.56
Ivy	42.07	41.82	31.86	26.37	41.76	32.68	49.71	49.04	40.75
Jcs	70.94	70.73	69.42	72.76	0.00	66.08	72.98	74.62	76.25
Lang	77.90	62.49	56.83	65.88	0.00	67.38	62.83	64.56	81.46
Log4j	56.69	50.68	69.42	61.18	0.00	0.00	50.61	59.65	41.99
Ode	15.79	15.79	12.28	19.30	0.00	17.54	17.54	24.56	12.28
Balance									
Bcel	44.97	44.97	48.88	33.00	29.29	4.99	33.19	48.84	41.07
Betwixt	31.76	44.50	49.78	39.08	29.29	34.52	44.94	52.50	44.79
Io	29.00	43.37	43.39	57.55	29.29	29.29	44.06	43.34	57.56
Ivy	41.91	41.90	36.76	34.38	41.89	36.87	46.96	46.90	41.71
Jcs	65.89	65.86	65.59	68.4	29.29	60.70	68.44	73.71	73.26
Lang	73.69	57.99	52.81	62.68	29.29	63.11	58.04	60.47	78.66
Log4j	52.79	42.72	65.59	58.61	29.29	58.32	47.79	55.98	42.52
Ode	40.45	40.42	37.95	42.76	29.29	41.69	41.69	46.59	37.95

4.4.2 Results Specific to RQ2

In this section, we assess ML techniques' performance for predicting software maintainability after applying various data resampling techniques to balance the datasets. Tables 4.2 to 4.9 show the performance of SMP models with respect to performance measures G-Mean and Tables 4.10 to 4.17 show the performance of SMP models with respect to performance measures Balance after applying data resampling techniques.

The use of data resampling techniques enhanced the performance of the ML techniques for developing SMP models. For the Bcel dataset, the G-Mean values ranged from 50.32% to 80.14%, and Balance ranged from 50.65% to 79.31%, respectively, for the majority of the cases after data resampling. Betwixt dataset, the G-Mean values ranged from 50.01% to 72.54% and Balance ranged from 50.17% to 72.48%,

respectively, for most of the cases after data resampling. On analyzing the results of SMP modes for the Io dataset after data resampling, we observed that for most of the cases, G-Mean and Balance values ranged from 50.96% to 87.82% and 50.11% to 86.31% respectively. The G-Mean and Balance values ranged from 50.16% to 73.44% and 50.74% to 73.39%, respectively, for most of the cases after data resampling for the Ivy dataset. In the case of the Jcs dataset, the G-Mean and Balance values ranged from 55.17% to 87.07% and 60.54% to 86.97%, respectively, for the majority of the cases after data resampling. The range of G-Mean and Balance values was 60.70% to 85.41% and 60.01% to 83.68%, respectively, for the majority of the cases after data resampling for the Lang dataset. Log4j dataset the G-Mean values ranged from 60.07% to 78.73%, and Balance values ranged from 60.02% to 78.39%, respectively, for the majority of the cases after data resampling. For the Ode dataset, the G-Mean and Balance values were observed in the range from 50.09% to 74.06% and 50.11% to 72.69%, respectively, for most of the cases after data resampling.

Figure 4.2 and Figure 4.3 shows the boxplots for the performance of maintainability prediction models regarding G-Mean and Balance after data resampling. It is evident from Figure 4.2 that G-Mean reaches up to 80% in most of the datasets. Also, Balance reaches up to 70% to 80% in all eight datasets, as shown in Figure 4.3. It is also quite evident from Figure 4.2 and Figure 4.3 that the median of G-Mean and median of Balance is even higher after data resampling. The G-Mean and Balance results showed improvement for all datasets when data resampling techniques were used. The improvement in G-Mean and Balance after data resampling is due to an increase in sensitivity and specificity. When the datasets were imbalanced, SMP models gave lower sensitivity values as models were having a smaller number of instances of the low maintainability class to learn the instances of this class properly. However, the sensitivity increased after data resampling that has increased the G-Mean of SMP models as G-Mean is the geometric mean of specificity and sensitivity. After

data resampling, the rise in sensitivity led to a decrease in the false-positive rate that improved the Balance results.

Table 4.2: G-Mean Results for Bcel Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	60.58	52.97	64.63	52.87	55.30	71.18	66.03	61.05	69.67
BSMOTE	61.26	56.72	60.66	31.54	48.17	73.22	55.02	50.83	76.42
ROS	58.91	55.11	57.12	22.65	73.13	63.42	59.88	57.53	54.63
SafeSMOTE	65.96	66.74	58.86	22.58	75.64	69.81	67.27	59.93	59.53
SMOTE	63.09	64.19	61.96	43.00	60.12	63.09	66.82	61.62	60.93
SMOTE-ENN	62.41	63.20	63.72	37.79	63.46	70.66	70.90	57.32	63.58
SMOTE-TL	67.86	61.73	63.64	50.46	70.26	70.22	68.12	66.82	63.39
SPIDER	61.16	51.77	59.74	22.65	33.33	69.46	51.60	59.84	52.95
SPIDER II	63.64	56.97	64.63	22.65	40.04	67.85	56.06	54.63	59.32
CNN	73.84	65.49	61.16	53.82	58.49	75.75	51.86	65.22	73.77
CNN-TL	68.56	65.81	66.94	44.29	65.96	69.92	62.30	56.62	67.66
CPM	54.50	51.09	53.54	37.50	22.69	43.57	21.09	53.36	54.54
NCL	65.49	64.95	60.96	52.26	23.57	56.72	39.84	50.31	46.69
RUS	80.13	79.55	65.63	58.32	70.41	75.99	60.47	47.06	69.50
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.3: G-Mean Results for Betwixt Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	52.32	62.17	53.58	55.70	55.19	60.91	69.51	57.29	71.87
BSMOTE	40.60	51.48	44.81	46.52	36.50	54.18	61.25	52.32	54.60
ROS	43.07	47.54	54.31	35.93	46.45	58.16	66.36	45.10	71.82
SafeSMOTE	54.74	59.07	62.24	39.75	52.54	68.77	70.28	68.18	58.26
SMOTE	59.04	52.17	49.46	55.19	31.77	62.65	71.25	54.68	64.04
SMOTE-ENN	58.87	55.91	54.37	60.55	25.87	62.35	72.54	54.68	70.28
SMOTE-TL	58.32	55.02	59.66	60.21	37.44	65.74	68.18	53.12	64.49
SPIDER	50.01	44.93	53.11	43.78	26.96	47.66	65.29	56.37	65.07
SPIDER II	53.67	54.46	48.35	50.28	25.14	62.40	68.10	24.37	62.70
CNN	50.98	58.89	54.99	45.43	25.34	47.16	51.35	48.07	49.80
CNN-TL	65.16	61.46	63.05	60.83	50.33	54.81	56.67	34.04	60.13
CPM	42.66	46.27	57.37	40.05	17.78	50.31	50.88	0.00	57.59
NCL	66.51	68.77	59.56	64.05	19.20	71.11	61.58	34.83	55.85
RUS	57.51	68.77	72.34	64.49	51.91	69.40	58.06	34.83	56.40
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.4: G-Mean Results for Io Dataset after Data Resampling

Res. Tech.	**C4.5**	**AB**	**IRBFNN**	**KNN**	**KSTAR**	**BG**	**LR**	**MLP-CG**	**RBFNN**
Adasyn	66.26	43.29	52.30	72.24	42.90	59.55	71.71	42.11	62.85
BSMOTE	53.96	43.77	43.77	69.51	44.25	62.31	53.14	43.48	61.63
ROS	53.37	43.68	41.40	78.64	58.41	61.63	51.57	43.39	77.27
SafeSMOTE	70.11	65.47	65.14	69.66	69.75	70.29	75.54	63.68	71.36
SMOTE	59.83	53.26	42.11	72.24	59.27	53.26	71.01	52.90	58.98
SMOTE-ENN	52.30	53.14	61.77	73.79	42.31	66.89	81.79	52.66	58.55
SMOTE-TL	66.42	53.26	60.81	73.28	48.29	53.26	86.31	60.53	57.39
SPIDER	60.94	53.49	52.54	76.64	52.54	53.49	52.90	53.14	67.83
SPIDER II	61.22	53.49	52.30	62.44	52.54	53.49	52.54	52.66	59.83
CNN	57.39	50.96	59.55	65.14	30.67	50.96	63.51	40.17	76.70
CNN-TL	78.22	73.96	57.69	64.69	30.67	73.96	80.65	38.35	77.57
CPM	75.93	64.98	65.14	66.10	30.74	64.98	65.47	55.15	65.79
NCL	62.44	61.77	62.04	67.98	31.62	61.77	52.42	61.36	87.82
RUS	65.84	57.10	63.01	78.64	71.12	57.10	64.69	37.96	66.80
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.5: G-Mean Results for Ivy Dataset after Data Resampling

Res. Tech.	**C4.5**	**AB**	**IRBFNN**	**KNN**	**KSTAR**	**BG**	**LR**	**MLP-CG**	**RBFNN**
Adasyn	56.43	35.82	35.58	58.19	41.76	59.36	72.95	65.77	65.37
BSMOTE	27.11	41.01	40.75	39.12	25.37	36.68	70.78	59.02	55.82
ROS	56.54	52.19	62.97	35.99	45.88	60.26	69.63	61.22	67.60
SafeSMOTE	67.45	60.90	60.13	39.38	47.88	64.94	65.18	56.00	72.90
SMOTE	63.72	54.76	62.36	46.18	38.92	59.39	72.95	63.41	72.29
SMOTE-ENN	56.08	51.14	50.90	49.83	57.03	50.32	66.69	60.61	67.58
SMOTE-TL	56.43	58.52	48.14	57.35	60.97	54.95	73.44	63.89	66.94
SPIDER	57.63	45.20	56.72	47.46	47.15	48.75	60.92	64.57	59.78
SPIDER II	57.72	54.76	52.76	47.07	43.65	54.59	59.30	62.54	69.77
CNN	40.17	56.33	58.40	55.22	24.99	51.14	61.12	49.49	54.36
CNN-TL	58.40	45.40	61.61	59.67	58.75	67.16	54.94	49.74	57.87
CPM	26.18	50.16	42.49	54.47	57.46	40.57	52.04	61.637	49.49
NCL	50.82	55.61	67.44	62.74	26.72	51.47	58.44	49.66	51.21
RUS	63.62	62.68	58.86	64.85	64.22	62.43	71.83	52.54	63.25
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.6: G-Mean Results for Jcs Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	74.62	76.44	71.34	77.47	66.17	80.03	80.44	74.97	80.52
BSMOTE	64.88	78.47	66.90	76.74	72.79	79.21	80.36	77.21	73.51
ROS	72.98	74.68	71.83	72.76	69.92	76.74	83.94	62.74	73.03
SafeSMOTE	80.14	80.89	82.55	74.91	81.02	81.62	77.54	78.63	78.94
SMOTE	77.46	77.96	60.16	80.85	72.52	83.37	79.17	68.60	78.33
SMOTE-ENN	80.30	82.84	70.94	73.54	62.83	84.22	77.26	65.68	76.67
SMOTE-TL	79.21	80.58	77.21	75.31	69.97	81.45	80.03	66.57	77.28
SPIDER	79.84	77.71	75.28	75.52	26.89	79.06	76.44	76.40	85.24
SPIDER II	81.11	76.95	81.17	81.65	55.17	85.35	79.21	76.41	79.49
CNN	75.77	77.21	70.25	62.48	60.35	78.72	75.31	10.84	80.03
CNN-TL	83.67	77.26	76.67	56.09	63.85	80.44	75.02	10.84	78.63
CPM	71.10	68.54	63.45	71.58	49.55	73.98	67.16	63.09	66.96
NCL	87.07	84.95	83.08	74.48	27.13	82.79	78.66	78.01	80.89
RUS	74.08	74.10	72.21	74.56	73.63	78.32	75.32	10.84	75.92
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.7: G-Mean Results for Lang Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	70.74	68.76	67.58	67.82	70.36	81.41	77.38	65.12	81.41
BSMOTE	78.85	67.57	67.01	68.92	46.28	79.29	77.97	69.15	85.41
ROS	73.93	64.92	75.27	66.45	67.18	77.97	76.18	55.13	80.78
SafeSMOTE	80.55	82.04	74.45	66.45	72.92	83.01	77.33	75.95	78.85
SMOTE	82.77	72.58	52.57	69.87	55.91	83.98	77.56	69.65	80.30
SMOTE-ENN	77.75	74.55	73.09	69.33	59.10	82.65	80.06	66.80	79.58
SMOTE-TL	79.09	80.79	73.81	68.99	74.25	83.01	79.09	68.54	80.63
SPIDER	83.26	74.14	75.34	65.31	19.25	83.12	73.30	77.33	79.42
SPIDER II	76.86	73.09	72.67	72.25	37.58	82.42	76.40	77.63	79.58
CNN	81.48	69.43	68.32	61.98	44.37	81.25	73.17	61.33	74.08
CNN-TL	76.18	69.25	73.30	60.70	61.95	72.96	65.31	61.02	69.99
CPM	78.89	67.58	68.03	61.58	35.05	79.34	72.78	58.64	73.21
NCL	79.51	75.99	66.26	66.80	19.20	74.14	75.99	70.77	81.01
RUS	76.88	73.04	76.63	66.49	62.78	80.79	68.54	60.22	77.42
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.8: G-Mean Results for Log4j Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	71.29	62.75	58.79	72.35	53.29	68.96	69.42	68.15	62.65
BSMOTE	59.65	61.66	57.24	73.48	51.96	61.3	67.83	62.32	66.7
ROS	67.83	54.71	61.66	62.75	64.8	59.87	68.7	62.75	65.35
SafeSMOTE	71.13	69.1	64.77	62.36	70.32	72.35	78.73	70.16	69.2
SMOTE	63.87	64.89	61.27	72.48	47.78	67.42	73.82	72.59	69.18
SMOTE-ENN	68.05	73.11	71.53	71.17	42.79	71.53	75.42	73.11	65.8
SMOTE-TL	69.47	71.65	64.71	73.59	60.84	64.52	73.27	70.25	69.84
SPIDER	67.85	58.85	62.52	65.77	21.76	66.09	55.3	62.04	55.93
SPIDER-II	65.13	63.67	65.69	64.28	37.59	64.38	63.85	67.1	60.76
CNN	55.58	67.57	61.55	68.63	52.48	64.28	66.03	63.71	60.07
CNN-TL	70.42	68.98	73.87	70.04	66.67	72.22	66.67	62.64	52.36
CPM	47.52	59.12	59.03	64.51	54.69	57.5	65.42	51.22	57.5
NCL	62.32	71.31	59.56	72.6	21.79	69.42	65.5	71.64	57.5
RUS	72.98	72.97	70.69	66.09	72.19	69.29	66.05	71.41	54.08
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.9: G-Mean Results for Ode Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	50.59	61.36	51	48.42	50.31	61.69	69.48	59.25	71.24
BSMOTE	44.71	51.69	49.41	52.33	43.29	42.89	64.97	49.64	59.65
ROS	55.36	51.36	61.42	42.53	65.06	57.17	72.28	52.49	72.04
SafeSMOTE	69.63	60.66	65.2	42.48	70.29	65.68	72.94	63.33	67.49
SMOTE	56.03	53.1	70.58	60.62	47.18	61.74	71.98	61.73	69.16
SMOTE-ENN	70.71	65.3	74.06	60.97	51.43	65.2	70.71	65.66	71.42
SMOTE-TL	71.76	65.3	70.67	64.26	60.86	63.68	71.76	70.43	72.52
SPIDER	58.3	43.08	58.07	42.45	48.87	53.32	58.3	55.83	66.14
SPIDER-II	62.63	51.2	61.23	45.76	18.72	49.51	62.59	45.49	59.49
CNN	51.69	50.47	45.97	53.41	64.43	55.61	51.69	50.11	66.98
CNN-TL	64.26	52.81	61.53	54.54	13.21	65.89	64.26	55.17	65.41
CPM	50.84	50.09	50.4	46.51	18.69	52.02	50.84	52.18	55.71
NCL	50.46	53.71	56.58	56.23	65.3	61.23	50.46	62.06	45.05
RUS	68.53	66.99	66.69	61.23	65.3	70.71	68.53	59.75	66.91
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

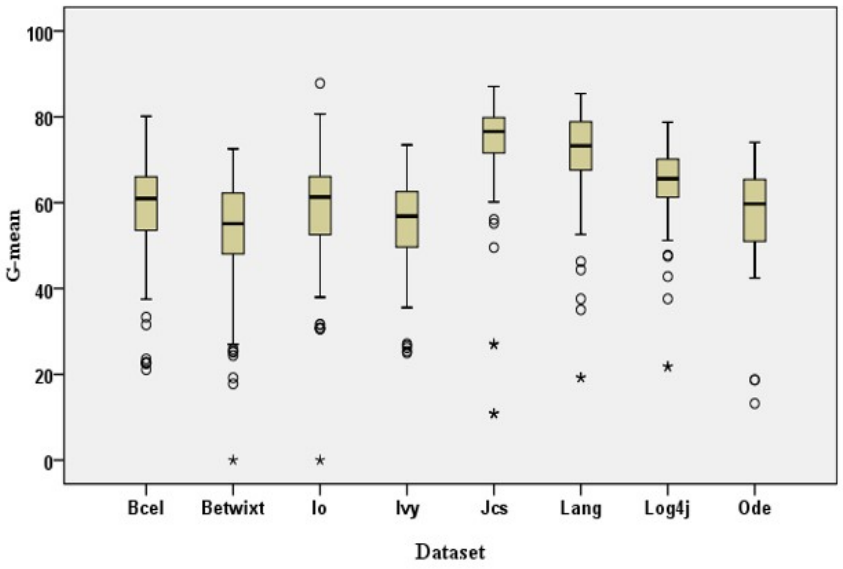


Figure 4.2: Box-plots for G-Mean Results after Data Resampling

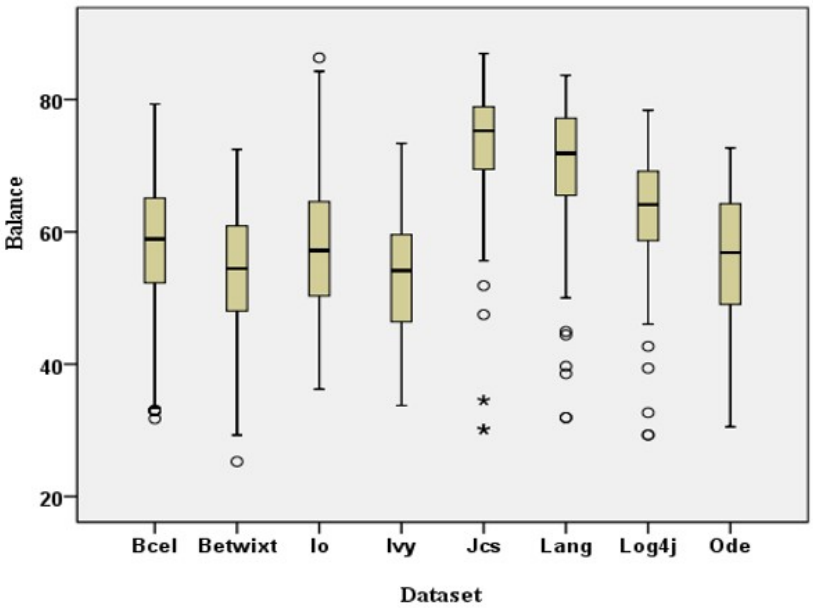


Figure 4.3: Box-plots for Balance Results after Data Resampling

Table 4.10: Balance Results for Bcel Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	58.83	51.55	62.78	51.5	52.5	69.96	65.08	59.09	68.89
BSMOTE	56.72	52.8	56.62	36.71	47.62	68.48	52.41	48.69	72.32
ROS	56.12	52.44	56.45	33	73.12	60.15	58.41	55.53	52.28
SafeSMOTE	65.78	65.59	58.51	32.96	74.91	69.71	66.94	59.42	58.19
SMOTE	60.03	60.38	59.56	43.74	58.56	60.03	65.65	59.39	59.03
SMOTE-ENN	59.76	60.07	63.27	40.21	62.58	67.77	69.78	55.42	63.15
SMOTE-TL	66.33	59.45	62.18	49.99	70.23	69.3	67.67	65.65	62.02
SPIDER	56.7	48.87	56.4	33	37.15	64.56	48.85	56.43	50.65
SPIDER-II	60.22	52.82	62.78	33	41.01	64.2	52.69	52.28	56.27
CNN	71.47	64.68	59.15	53.64	55.96	74.46	50.92	65.11	73.09
CNN-TL	67.92	64.95	66.53	44.36	65.78	69.7	62.21	56.38	66.94
CPM	54.51	51.1	52.97	39.19	33.02	44.24	31.75	53.36	54.5
NCL	60.64	60.55	56.67	51.16	33.32	52.8	40.98	48.55	44.99
RUS	78.85	79.31	64.79	58.04	70.27	75.3	60.36	47.23	69.43
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.11: Balance Results for Betwixt Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	51.19	60.77	52.73	54.83	54.46	59.91	69.43	56.58	71.84
BSMOTE	41.86	49.68	44.58	46.34	39.34	52.11	59.20	51.19	52.27
ROS	43.78	46.84	54.30	39.08	46.58	56.41	65.94	45.45	70.93
SafeSMOTE	54.46	58.49	61.98	41.46	52.52	68.73	70.28	68.16	58.05
SMOTE	57.86	51.11	48.74	54.46	36.81	61.86	71.02	54.07	64.03
SMOTE-ENN	57.74	54.13	53.25	59.64	34.17	60.89	72.48	54.07	70.28
SMOTE-TL	57.87	54.33	58.27	59.85	39.87	65.41	68.14	52.80	64.48
SPIDER	49.03	44.62	52.39	44.15	34.51	46.89	62.45	55.30	63.57
SPIDER II	52.79	52.22	48.05	49.17	33.71	59.75	66.47	25.27	61.11
CNN	50.35	58.34	53.63	45.71	33.85	46.67	49.64	48.14	49.72
CNN-TL	64.25	61.11	62.98	59.24	49.8	54.69	56.42	37.95	59.74
CPM	43.54	46.5	57.04	40.6	31.12	50.17	50.86	29.29	57.58
NCL	64.41	67.85	58.20	63.54	31.91	69.38	59.37	38.28	55.42
RUS	57.21	67.85	72.30	64.46	51.9	68.83	58.05	38.28	55.87
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.12: Balance Results for Io Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	63.61	43.26	50.11	70.25	43.15	56.82	71.71	42.87	61.65
BSMOTE	50.46	43.35	43.35	64.56	43.41	57.52	53.14	43.30	57.42
ROS	50.37	43.34	42.54	78.61	56.32	57.42	51.57	43.28	76.38
SafeSMOTE	68.96	63.23	63.06	64.58	68.72	69.09	75.53	62.20	69.76
SMOTE	56.93	50.35	42.87	70.25	56.71	50.35	71.01	50.27	56.59
SMOTE-ENN	50.11	50.33	57.45	70.97	42.95	63.87	81.79	50.21	56.39
SMOTE-TL	63.68	50.35	57.24	70.76	48.06	50.35	86.31	57.16	55.78
SPIDER	57.27	50.39	50.18	71.68	50.18	50.39	52.89	50.33	64.20
SPIDER II	57.34	50.39	50.11	57.54	50.18	50.39	52.54	50.21	56.93
CNN	55.78	49.60	56.82	63.06	36.22	49.60	63.51	41.81	75.97
CNN-TL	78.16	73.76	55.94	64.54	36.22	73.76	80.64	39.69	77.48
CPM	75.39	62.98	63.06	63.54	36.24	62.98	65.46	54.32	63.39
NCL	57.54	57.45	57.49	64.24	36.36	57.45	52.420	57.37	84.28
RUS	65.58	55.61	61.76	78.61	71.11	55.61	64.69	40.00	66.41
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.13: Balance Results for Ivy Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	55.03	38.96	38.86	56.95	41.89	57.68	72.63	64.46	65.33
BSMOTE	34.64	41.77	41.71	41.04	33.97	39.25	69.59	55.73	54.64
ROS	53.94	49.38	62.76	39.03	44.43	56.74	67.86	58.63	67.60
SafeSMOTE	67.05	59.54	59.58	41.17	45.77	63.89	64.68	55.36	72.56
SMOTE	61.10	51.78	61.88	46.40	40.94	56.47	72.63	60.95	72.21
SMOTE-ENN	53.76	49.14	49.06	49.40	55.38	48.86	65.88	58.35	67.16
SMOTE-TL	55.03	56.12	46.72	56.87	60.26	53.25	73.39	62.33	66.07
SPIDER	54.27	44.34	54.01	46.51	46.39	46.85	56.89	61.44	57.92
SPIDER II	54.3	51.78	51.09	46.36	43.89	51.74	56.43	59.12	68.88
CNN	41.52	54.96	57.09	54.73	33.74	49.14	58.59	49.20	53.63
CNN-TL	58.29	45.74	61.23	59.59	58.64	67.16	54.76	48.69	57.87
CPM	34.28	48.79	43.34	53.00	55.61	41.65	50.74	61.46	48.49
NCL	49.04	53.56	64.07	61.58	34.34	49.23	54.44	48.57	49.16
RUS	63.62	62.66	58.09	64.6	64.14	62.27	71.73	52.48	63.21
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.14: Balance Results for Jcs Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	73.71	74.86	69.51	76.61	65.19	79.32	80.42	74.96	79.32
BSMOTE	60.54	75.82	63.00	73.41	70.23	76.07	78.25	76.42	70.52
ROS	68.44	70.89	69.77	68.40	69.41	73.41	83.78	61.13	72.51
SafeSMOTE	80.11	80.69	82.40	70.95	80.69	81.62	77.35	78.5	78.83
SMOTE	75.39	75.62	57.36	79.87	71.34	83.28	79.14	67.72	78.17
SMOTE-ENN	79.51	82.22	67.88	71.97	58.04	84.02	77.10	64.82	76.03
SMOTE-TL	78.72	79.69	76.42	74.95	69.45	81.16	79.93	66.57	77.11
SPIDER	78.02	75.51	72.87	72.98	34.51	77.61	74.86	75.82	85.24
SPIDER II	80.04	75.14	80.92	80.35	51.87	84.92	78.72	76.33	78.92
CNN	73.08	75.27	67.59	62.42	57.43	75.91	74.95	29.29	79.32
CNN-TL	82.77	77.10	76.03	55.64	63.64	80.42	74.92	29.29	78.50
CPM	65.91	65.31	60.16	69.64	47.49	70.69	65.87	62.07	64.64
NCL	86.97	84.94	83.02	74.25	34.53	82.75	78.28	78.01	80.69
RUS	74.09	74.10	71.85	74.00	73.50	78.17	75.24	29.29	75.88
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.15: Balance Results for Lang Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	69.17	65.39	64.93	67.61	67.64	81.41	77.20	62.38	81.41
BSMOTE	75.95	63.15	63.03	66.94	44.94	76.09	75.62	65.51	83.68
ROS	70.67	60.55	74.15	62.87	64.74	75.62	74.71	52.45	79.82
SafeSMOTE	80.39	81.62	74.22	62.87	71.6	82.34	76.51	74.58	78.43
SMOTE	82.17	68.36	50.01	68.61	52.65	82.96	76.67	68.47	79.51
SMOTE-ENN	75.52	70.86	70.36	67.16	55.30	80.86	79.34	65.63	78.99
SMOTE-TL	78.63	78.43	72.13	68.00	72.37	82.34	78.63	68.51	80.62
SPIDER	82.50	70.74	72.90	62.46	31.91	81.06	70.45	76.51	77.80
SPIDER II	75.09	70.36	70.18	69.98	39.67	80.75	74.84	77.42	78.99
CNN	80.26	68.32	67.51	60.65	44.42	80.12	72.63	60.81	74.08
CNN-TL	76.14	68.66	73.3	60.01	61.31	72.92	65.15	60.34	68.18
CPM	73.80	67.4	68.02	60.38	38.56	78.81	72.76	58.64	72.98
NCL	76.15	73.16	62.81	65.63	31.91	70.74	73.16	67.81	78.51
RUS	76.76	73.03	76.53	66.49	61.16	80.61	68.51	60.12	77.18
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.16: Balance Results for Log4j Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	70.73	60.21	56.66	71.96	50.83	68.04	69.25	66.93	61.95
BSMOTE	55.98	57.69	54.17	72.87	49.34	57.61	66.08	59.05	63.80
ROS	66.08	52.31	60.79	60.21	63.52	57.14	68.66	60.21	63.17
SafeSMOTE	71.09	68.58	64.71	60.02	69.54	71.96	78.39	70.03	68.21
SMOTE	61.60	62.07	58.65	72.47	46.05	65.05	73.82	71.08	67.59
SMOTE-ENN	66.20	70.49	69.70	70.63	42.69	69.70	75.41	70.49	65.18
SMOTE-TL	68.88	70.48	62.82	73.51	57.48	63.22	72.96	68.94	69.17
SPIDER	65.25	55.76	60.86	63.38	32.66	62.5	52.5	58.96	53.71
SPIDER II	63.06	61.5	65.58	61.8	39.39	61.85	62.31	64.89	60.09
CNN	53.55	66.52	59.61	68.46	50.56	61.8	65.37	63.66	59.54
CNN-TL	69.03	68.78	73.81	69.06	66.67	71.82	66.63	62.06	52.35
CPM	46.91	57.54	58.66	64.46	53.66	56.48	65.16	51.21	57.29
NCL	59.05	68.71	59.53	72.17	32.66	65.86	62.3	69.77	55.24
RUS	72.98	72.63	70.69	65.9	72.08	69.25	65.99	71.19	54
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

Table 4.17: Balance Results for Ode Dataset after Data Resampling

Res. Tech.	C4.5	AB	IRBFNN	KNN	KSTAR	BG	LR	MLP-CG	RBFNN
Adasyn	49.38	58.38	48.87	47.82	49.24	59.22	69.47	57.87	71.23
BSMOTE	44.06	49.02	47.65	50.04	42.90	42.84	62.68	47.70	56.23
ROS	52.52	48.96	61.27	42.76	62.72	53.85	71.90	50.79	71.89
SafeSMOTE	68.92	58.7	64.8	42.75	68.47	64.16	72.69	62.52	67.33
SMOTE	54.13	51.03	67.20	58.68	46.25	59.24	71.83	59.82	68.98
SMOTE-ENN	70.66	63.91	70.90	59.38	48.97	63.85	70.66	64.92	70.61
SMOTE-TL	71.63	63.91	67.23	63.79	58.19	62.78	71.63	70.28	72.43
SPIDER	54.05	42.87	55.67	42.74	47.86	50.27	54.05	52.64	63.2
SPIDER II	58.78	48.92	60.32	45.11	46.62	47.67	58.77	45.03	56.19
CNN	49.02	49.72	46.07	52.83	31.77	52.58	49.02	50.11	66.86
CNN-TL	63.79	52.8	61.37	54.54	64.27	65.37	63.79	54.75	65.4
CPM	48.83	49.48	49.95	46.73	30.53	49.95	48.83	52.04	53.97
NCL	47.85	51.23	53.68	54.23	31.77	57.52	47.85	58.62	44.12
RUS	68.41	66.99	66.69	61.22	63.91	70.54	68.41	59.69	66.89
Res. Tech. indicates resampling technique, AB indicates AdaBoost, BG indicates Bagging									

4.4.3 Results Specific to RQ3

To assess the performance of data resampling techniques used in this chapter, we perform the Friedman test with respect to performance metrics G-Mean and Balance for all eight datasets used in the study along with the scenario when no data resampling is used. We evaluated the hypothesis stated in Section 4.2.5. The hypothesis for Friedman and Wilcoxon test is evaluated at a confidence level of 95% (α = 0.05) by extracting the values of the performance metrics G-Mean and Balance of all datasets used in this chapter. Table 4.18 shows the Friedman test ranks for G-Mean and Balance. The higher the rank obtained by the resampling technique, the better would be that technique.

Table 4.18: Friedman Ranking based on G-Mean and Balance

Based on G-Mean		**Based on Balance**	
Technique	**Mean Rank**	**Technique**	**Mean Rank**
SafeSMOTE	11.23	SafeSMOTE	11.35
SMOTE-TL	11.08	SMOTE-TL	11.27
SMOTE-ENN	10.66	SMOTE-ENN	10.44
RUS	10	RUS	9.69
SMOTE	9.49	SMOTE	9.42
Adasyn	8.94	CNN-TL	9.1
CNN-TL	8.65	Adasyn	9.00
NCL	8.45	NCL	7.8
SPIDER II	7.6	SPIDER II	7.29
ROS	7.22	ROS	7.06
SPIDER	7.16	SPIDER	6.47
CNN	6.62	CNN	6.33
BSMOTE	6.42	BSMOTE	5.83
CPM	4.64	CPM	4.72
No-Resampling	2.47	No-Resampling	4.24

On conducting the Friedman test for different data resampling techniques with respect to G-Mean measure on all eight datasets used in the study, the p-value obtained is 0.00 ($p < 0.05$), which means the results of the Friedman test are significant. It

is evident from Table 4.18 that SafeSMOTE achieves the best rank with respect to G-Mean. The worst rank was obtained for no resampling situation. Similarly, on conducting the Friedman test for different data resampling techniques for Balance measure on all eight datasets used in this chapter, the p-value obtained was 0.00 ($p - value < 0.05$), which means again the results of the Friedman test are significant with respect to Balance, the mean rank obtained after the Friedman test for different data resampling techniques along with no resampling scenario are shown in Table 4.18. Again, SafeSMOTE yielded the best rank with respect to Balance measure after the Friedman test is applied, and the worst rank is obtained for the no resampling situation. As the test statistics of the Friedman test are significant for both G-Mean and Balance, this leads to rejection of the null hypothesis H0 and H1 and acceptance of the alternate hypothesis H2 and H3. Therefore, in this way, we observe a significant improvement in the performance of SMP models developed after applying data resampling techniques on imbalanced datasets. It is observed that the enhanced version of SMOTE, namely SafeSMOTE and hybrid resampling techniques, SMOTE-TL, SMOTE-ENN, are amongst four ranked techniques as per ranking obtained after the Friedman test with respect to G-Mean and Balance measures. The SafeSMOTE technique emerges as the best technique to improve the performance of prediction models.

To further extend our analysis, i.e., to get insight into whether the SafeSMOTE technique is statistically better than other resampling techniques used in the study or not, we apply Wilcoxon signed-rank test at 95% level of confidence ($\alpha = 0.05$) by doing the Bonferroni correction. The set of hypothesis evaluated using the Wilcoxon test is stated in Section 4.2.5. Using the Wilcoxon signed-rank test, a pair-wise comparison amongst the SafeSMOTE technique and other resampling techniques is computed with respect to G-Mean and Balance measures of all ML techniques for all datasets. The test statistics of Wilcoxon signed-rank are reported in Table 4.19 both for G-Mean and Balance.

Table 4.19: Wilcoxon Test Results

Techniques Examined	**G-Mean**	**Balance**
SafeSMOTE vs. SMOTE-TL	NS (p-value = 0.0.404)	NS (p-value = 0.287)
SafeSMOTE vs. RUS	S+ (p-value = 0.161)	S+ (p-value = 0.375)
SafeSMOTE vs. SMOTE-ENN	NS (p-value = 0.079)	NS (p-value = 0.013)
SafeSMOTE vs. SMOTE	S+ (p-value = 0.001)	S+ (p-value = 0.000)
SafeSMOTE vs. CNN-TL	S+ (p-value = 0.003)	S+ (p-value = 0.000)
SafeSMOTE vs. Adasyn	S+ (p-value = 0.000)	S+ (p-value = 0.000)
SafeSMOTE vs. NCL	S+ (p-value = 0.000)	S+ (p-value = 0.000)
SafeSMOTE vs. SPIDER II	S+ (p-value = 0.000)	S+ (p-value = 0.000)
SafeSMOTE vs. ROS	S+ (p-value = 0.000)	S+ (p-value = 0.000)
SafeSMOTE vs. SPIDER	S+ (p-value = 0.000)	S+ (p-value = 0.000)
SafeSMOTE vs. CNN	S+ (p-value = 0.000)	S+ (p-value = 0.000)
SafeSMOTE vs. BSMOTE	S+ (p-value = 0.000)	S+ (p-value = 0.000)
SafeSMOTE vs. CPM	S+ (p-value = 0.000)	S+ (p-value = 0.000)
SafeSMOTE vs.No-Resampling	S+ (p-value = 0.000)	S+ (p-value = 0.000)

In Table 4.19, S+ means a significant difference in the performance of two corresponding pairs of resampling techniques, and NS signifies that there is no significant difference. The results depict that SafeSMOTE significantly outperforms than Adasyn, SMOTE, BSMOTE, SPIDER, SPIDER II, ROS, CNN, CNN-TL, CPM, NCL, and no resampling by both G-Mean and Balance. Also, the test results depict that SafeSMOTE does not significantly outperform compared to SMOTE-TL, SMOTE-ENN, and RUS. The performance of SMOTE-TL, and SMOTE-ENN is comparable with SafeSMOTE.

4.5 Discussion

In this chapter we have conducted an empirical evaluation for SMP using the software system written in Java language. The datasets corresponding to these software systems were imbalanced. The data resampling techniques, including oversampling, undersampling, and hybrid resampling are applied in this chapter to deal with the

imbalanced data. SMP models are developed using ML techniques. The results of the chapter show that the performance of ML techniques significantly improved after data resampling. We observed that that G-Mean values were greater than 50 in 85%, 72%, 84%, 73%, 95%, 95%, 95%, and 83% of the cases respectively for Bcel, Betwixt, Io, Ivy, Jcs, Lang, Log4j, and Ode datasets after data resampling. Similarly, Balance values are higher than 50 in 81%, 68%, 83%, 66%, 95%, 95%, 94%, and 87% of the cases for Bcel, Betwixt, Io, Ivy, Jcs, Lang, Log4j, and Ode datasets respectively after data resampling. The performance of the SMP models showed an improvement of 31.71% for Bcel, 50.61% for Betwxit, 41.31% for Io, 36.19% for Ivy, 8% for Jcs, 35% for Lang, 29.42% for Log4j and 78.08% for Ode dataset in terms of G-Mean after data resampling. Also, in terms of Balance performance measure, the SMP models showed an improvement of 31.01% for Bcel, 22.38% for Betwxit, 31.88% for Io, 29.25% for Ivy, 14% for Jcs, 18.84% for Lang, 21.48% for Log4j and 40.60% for Ode dataset after data resampling.

SMP models developed after resampling with SafeSMOTE performed well on all the datasets. The SafeSMOTE technique improved the performance of the models in terms of G-Mean and Balance. According to statistical analysis carried out with the Friedman test, the SafeSMOTE technique achieved the highest rank whereas the no resampling situation has attained the worst rank. These results show the competence of SafeSMOTE technique to deal with the imbalanced data.

The pair-wise comparison of the performance of SafeSMOTE with all other resampling techniques used in this chapter indicates that the performance of SafeSMOTE was better than all other resampling techniques except SMOTE-TL and SMOTE-ENN. The performance of two other variants of SMOTE namely SMOTE-TL and SMOTE-ENN was comparable with the top-ranked technique (i.e., SafeSMOTE).

The superiority of the SafeSMOTE technique over other techniques is because this technique does not create the same number of synthetic instances for each the

minority instance; instead, it emphasizes on the instances that fall in the safe-region and discounts the instances that are noise. The SafeSMOTE technique's superiority denotes that a data resampling technique should properly use a strategy for generating the synthetic instances that evade noise and redundancy.

Chapter 5

Analysis of Ensemble Techniques for Imbalance Data Problem

5.1 Introduction

The imbalanced data problem in software maintainability arises when one class which usually refers to the concept of interest i.e., low maintainability class or positive class is underrepresented in the training dataset. Alternatively stated, the number of negative instances i.e., high maintainability class instances overwhelm the number of low maintainability class instances in the imbalanced dataset. Prediction models developed using ML techniques with an imbalanced dataset gives suboptimal classification results as shown in the previous chapter, i.e., majority class instances are predicted with good accuracy but on the other hand minority class instances are often misclassified as they are treated as noise during the learning process [74]. The results of the previous chapter confirm that the performance of the models developed using ML is significantly improved after data balancing with resampling techniques. Considering the importance of imbalanced data issues, there is still an active need to explore

other methods to treat it so that efficient models for maintainability prediction can be developed. In the last chapter, the data resampling techniques are used that are called a data level solution to deal with the imbalanced data problem. The data resampling techniques tend to balance the imbalanced datasets and lower down the skewed data distribution before learning the prediction model [28, 186–188].

Another approach to deal with imbalanced data is the use of ensemble learning techniques. The ensemble learning techniques train the multiple base classifiers and combine their prediction to obtain a single final output class label [25]. Ensemble techniques combine multiple classifiers to make a single consolidated decision. The ensembles improve the performance for a classification task because by combining multiple classifiers the error of a single classifier will be compensated by the other individual classifiers. Ensemble techniques have been widely used in diverse domains.

In the domain of software engineering, ensembles are used to improve the performance of software defect and change prediction models. Although ensembles have gained popularity in predictive modeling due to their improved performance, they are not able to tackle the imbalanced data on their own. Therefore, in the framework of imbalanced data, ensembles are specifically designed by incorporating data resampling techniques and are called as Ensembles for Imbalanced Data Problem (EIDP). EIDP is the hybridization of data resampling techniques and ensembles, where data resampling techniques resolve the skewness in the imbalanced training data set and after that balanced data, is fed to multiple classifiers comprising an ensemble [195–197].

EIDP techniques are explored by a few of the researchers in the literature to develop prediction models to predict defects in software classes. Wang and Zhou [198] combined three data resampling techniques with ensembles to develop software defect prediction models. Wang and Yao [74] compared ML techniques, classic ensembles, and EIDP techniques to develop software defect prediction models using imbalanced datasets. Yohannes et al. [199] examined and compared prominent ensembles tech-

niques and ensembles combined with data balancing techniques for software change prediction and advocated that data balancing combined with ensembles improves the performance of models. Therefore, as noticed that though EIDP seems to be very capable of dealing with the imbalanced data in their concrete framework, still they are being explored very little literature. Also, the use of EIDP techniques for SMP has not been explored in literature.

Therefore, this chapter deals with efficiently learning from imbalanced data using EIDP techniques for developing SMP models to correctly predict low maintainability classes. In this chapter, we compare and analyze three types of EIDP techniques. These are Bagging-based ensembles, Boosting-based ensembles, and hybrid ensembles. Bagging-based ensembles are a hybridization of data resampling and Bagging, Boosting-based ensembles are a hybridization of data resampling and boosting and whereas hybrid ensembles combine data resampling, Bagging, and boosting in a single technique. Apart from this, the chapter also develops SMP classic ensembles (AdaBoost and Bagging) and compares their performance with EIDP techniques. We explore the following research questions in this chapter.

- RQ1: Are the classic ensembles are capable of predicting low maintainability classes in imbalanced SMP datasets?
- RQ2: How is the performance of Bagging-based ensembles, Boosting-based ensembles, and hybrid ensembles for predicting low maintainability classes in imbalanced SMP datasets?
- RQ3: Which method is the best EIDP technique for SMP in an imbalanced data framework?

RQ1 aims to assess the predictive capability of classic ensembles techniques for the development of SMP models using imbalanced data. The predictive capability of

each classic ensemble techniques is compared with EIDP techniques. We investigate RQ2 to assess the predictive performance of EIDP techniques for the development of SMP models. RQ3 aims to determine the best techniques by conducting statistical analysis on Balance and G-Mean to determine the technique exhibiting the best predictive capabilities. Moreover, we make pairwise assessments of all the investigated techniques with the technique that exhibits the best performance. This helps in evaluating which pairs of techniques are statistically significantly different than that of the best performing technique. This chapter is organized as follows: Section 5.2 describes be ensembles techniques used in this chapter, Section 5.3 presents the research background. Section 5.4 presents the results and analysis of RQ's. Section 5.5 presents the discussion. The results of this chapter are published in [200, 201].

5.2 An Overview of Ensembles for Imbalanced Data Problem

EIDP techniques are the hybridization of data resampling techniques and classical ensembles. The data resampling techniques provide balanced data distribution to ensembles for the learning process. Three categories of EIDP techniques are investigated in this chapter namely (i) Boosting-based ensembles (ii) Bagging-based ensembles (ii) Hybrid ensembles. Figure 5.1 describes the ensemble techniques investigated in each category.

5.2.1 Boosting-based Ensembles

This category includes the ensembles in which data resampling techniques discussed in Chapter 4 are hybridized with AdaBoost. Boosting improves the performance of weak learners by concentrating on hard instances that are tough to classify. A series

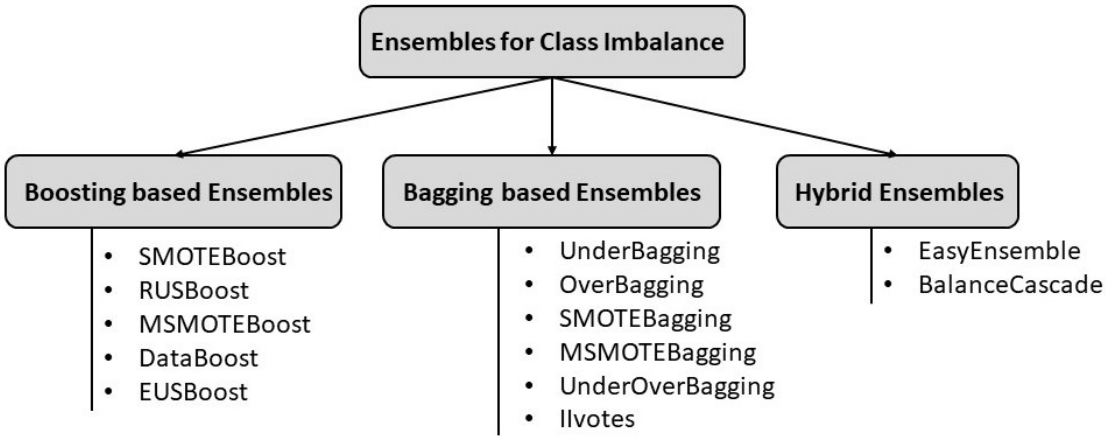


Figure 5.1: Ensembles to Address Imbalanced Data Problem

of learners are produced and their collective output is produced using weighted voting as the outcome of the model. In each step of boosting, the training data points are re-weighted and carefully chosen based on the performance of earlier learners in the training process. Throughout each of iteration, boosting tries to yield a new learner that is better able to predict instances for which the preceding learner's performance was poor. The Boosting-based techniques investigated in this chapter are SMOTEBoost, MSMOTEBoost, RUSBoost, DataBoost, and EUSBoost. A brief description of these techniques is presented below.

- ***SMOTEBoost***

  This technique combines SMOTE and AdaBoost as one single approach to handle imbalanced data. In SMOTEBoost, SMOTE is used to generate synthetic data points corresponding to minority class to improve the accuracy of the prediction model over the minority classes, and boosting is used so that the accuracy of the model over the entire training dataset also not sacrificed [202]. The aim of combining SMOTE with boosting is twofold, first minority class data points are learned in a better way by the model, and secondly, the classifier is not

only provided the minority class data points that were incorrectly classified in the previous iteration but also oversampling with the help of SMOTE provides more number of minority data points to the classifier. In this way, SMOTEBoost tends to improve the accuracy of the model towards minority class i.e., to improve the true positive rate of the model. The classic AdaBoost algorithm gives equal weights to all incorrectly classified data points. If the dataset is imbalanced, AdaBoost algorithm would select data points from the pool of data that primarily belong to the majority class, then in the successive iteration of AdaBoost, the selection of data points may still be from the skewed data. Although AdaBoost decreases the bias in the final ensemble, in case of imbalanced data, it might not hold and the successive iterations may still be biased towards majority class. In SMOTEBoost, SMOTE is introduced in each iteration of boosting that will empower each classifier to be able to select more of the minority class data points and learn a wider and better decision boundary for the minority class. Combining SMOTE in each iteration of AdaBoost, we are predominantly improving the probability of selecting a greater number of minority class data points that were dominated by the majority class due to imbalanced data. Also, introducing SMOTE in AdaBoost rounds increases the diversity in the classifiers of the ensemble learner, as SMOTE generates a different set of synthetic data points in each AdaBoost round.

- ***RUSBoost***

  This technique combines random undersampling and AdaBoost as one technique [203]. It performs just like SMOTEBoost, instead of SMOTE, RUS is applied in each round of boosting that randomly removes the majority class instances. As compare to SMOTEBoost, RUSBoost is a simple and faster ensemble to deal with imbalanced data problem as in each round of boosting, a lesser number of

instances are fed to the classifiers. Also, the main limitation of RUS, which is the loss of data, is significantly overcome by merging it with boosting. While certain data may be absent during a given round of boosting, it will probably be included when training of models during other iterations. So, in this in RUSBoost, RUS helps to balance the skewed data, while AdaBoost improves the performance of the classifiers using the balanced data.

- ***MSMOTEBoost***

  This is a hybridization of MSMOTE and AdaBoost. To deal with the imbalanced data, modified synthetic minority oversampling technique (MSMOTE) is used to generate synthetic data points in each round of AdaBoost [204]. MSMOTE is a modified version of SMOTE. Unlike SMOTE that creates synthetic data points corresponding to each minority class data point, MSMOTE categorizes the minority data points into three types, secure, border, and noisy. The secure, noisy, and border data points are is determined as follows. Let X be a minority class data point. If the label of X is the same as that of its k nearer neighbors, X would secure. If the class label of X is completely different from that of its k nearer neighbors, X would be noisy and if X is neither noisy nor secure, it is considered as a border data point. During the creation of synthetic data points from the minority class, MSMOTE ignores the noisy data points of the minority class. The synthetic data points corresponding to secure minority data points are created by selecting any of its k nearer neighbors, whereas, for border data points, synthetic data points are created by selecting the only nearest neighbors. The procedure for creating synthetic data points is the same as that of SMOTE as discussed in Chapter 4.

- ***DataBoost***

  This technique is somewhat different than of the above discussed Boosting-based

ensemble techniques. It associates data generation and boosting to enhance the accuracy of both the minority and majority classes. The Databoost algorithm works in three stages. In the first stage, all the instances in the training dataset are allotted an identical weight. This dataset is then used for training the first set of classifiers. In the second stage, the hard instances are identified which are called seed instances. Corresponding to each of these seed instances, a set of synthetic instances is created. In the next stage, the synthetic instances generated in the second stage are combined with the original training data [205]. In this way, the class distribution and the total weights of both of the classes are re-balanced. The algorithm repeats until classifier's error rate in the current iteration is worse than a threshold value of 0.5.

- ***EUSBoost***

  This technique combines evolutionary undersampling with AdaBoost [206]. The main objective is to improve the accuracy of the base classifier while maintaining diversity. The algorithm tends to obtain a useful subset of the original training. To so do, it starts randomly undersampling multiple subsets of the original training data set. The subsets are evolved until the presently best-undersampled dataset cannot be improved further. The original training dataset is denoted as T and it consists of N data points. The search space associated with T of the EUS algorithm is established by using the subsets of T. Afterward, the chromosomes characterize the subsets of T. This is done using a binary representation. A chromosome comprises of genes (one for each data point of T) with two possible states: 0 and 1. If the gene is 1, then its related data point is contained within the subset of T characterized by the chromosome. If it is 0, then this means that a data point is not included. Let ST denoted the subset of T and that coded by a chromosome. The fitness function of ST in

EUS is defined as follows:

$$fitness(ST) = \alpha * Class_{rate} + (1 - \alpha) * PR \tag{5.1}$$

Here, $Class_{rate}$ denotes the classification rate which is defined as the percentage of correctly classified data points from T using ST when kNN classification is used. PR denotes the percentage of reduction of data points in T on account of undersampling and it is given as:

$$PR = (\frac{|T - ST|}{|T|}) * 100 \tag{5.2}$$

In the equation of fitness function $\alpha = 0.5$ is considered. After this, the algorithm fed the best-improved dataset inside the loop of AdaBoost.

5.2.2 Bagging-based Ensembles

Bagging-based ensembles are the hybridization of Bagging and data resampling techniques. The hybridization of data resampling techniques with Bagging is usually easier as compare to their integration with boosting techniques as Bagging techniques do not require any kind of weight updating. The imbalanced data is dealt with in Bagging-based ensembles by creating bootstrap replicas. The bootstrap replicas are then used as input to the classifiers in the ensemble. The techniques evaluated in this category are UnderBagging, OverBagging, SMOTEBagging, MSMOTEBagging, UnderOverBagging, and IIVotes.

- ***UnderBagging***

  UnderBagging combines random undersampling with Bagging to construct a robust ensemble method [198]. A classifier in the ensemble is iteratively

constructed using a subset of a balanced training dataset that is generated using undersampling the majority class instances. Multiple classifiers are trained using these balanced subsets of the dataset. After construction of the individual classifiers, when an unseen instance is presented to the ensemble, the majority voting on the predictions of the individual classifiers is done to output the class label of the unseen instance.

- ***OverBagging***

  OverBagging combines random oversampling with Bagging to construct a robust ensemble method [198]. A classifier in the ensemble is iteratively constructed using the subsets of balanced training dataset that are generated using oversampling the minority class instances. Multiple classifiers are trained using these balanced subsets of the dataset. After construction of the individual classifiers, when an unseen instance is presented to the ensemble, the majority voting on the predictions of the individual classifiers is done to output the class label of the unseen instance.

- ***UnderBagging to OverBagging***

  This technique follows a different strategy than that of UnderBagging and OverBagging [198]. This technique uses both undersampling and oversampling to create diversity in the sampled data. In each iteration, a resampling rate is set that defines the number of instances selected from each class.

- ***IIvotes***

  IIvote aggregates SPIDER technique with an adaptive ensemble of classifiers. As ensembles can adapt to data points that are hard to learn in subsequent iterations. Such difficult data points from the majority class could be particularly significant when the training dataset is imbalanced data [207]. In IIvotes, the

SPIDER technique is incorporated inside the ensemble framework to developed a classification model to be more focused on minority class. IIvotes generate a new training set by importance sampling so that the newly generated training set will encompass nearly the same quantity of correctly and incorrectly classified data points. In importance sampling, a data point is selected randomly with all data points of equal probability of being chosen. Then this data point is classified by ensemble constituting the classifiers that have not learned that data point. If a data point is incorrectly classified then it is placed into the new training set T_i. Otherwise, it is sampled into T_i with probability $\frac{E(i)}{1-E(i)}$, where $E(i)$ denotes a generalization error. The sampling process is carried until n data points are chosen. Each T_i is balanced using SPIDER. The SPIDER technique is described in detail in Chapter 4.

- ***SMOTEBagging and MSMOTEBagging***

  SMOTEBagging combines SMOTE and Bagging to create ensembles to handle imbalanced data. The available training dataset is divided into k subsets. In each subset, synthetic instances are generated using SMOTE [198]. Each subset is oversampled by creating synthetic instances into it according to the resampling rate. The resampling rate defines the number of synthetic instances to be created in that subset. MSMOTEBagging integrates MSMOTE technique with Bagging. In MSMOTEBagging, the subset construct process same as used in SMOTEBagging except for the minority class instances is generated by the MSMOTE technique [198].

5.2.3 Hybrid Ensembles

Hybrid ensembles carry double ensemble learning where the base classifiers are ensembles rather than single classifiers. The hybrid ensembles investigated in this chapter are EasyEnsemble and BalanceCascade.

- ***EasyEnsemble***

  Drummond and Holte [208] advocated that undersampling is an effective approach to deal with the imbalanced data problem. But, the disadvantage of undersampling is that many valuable data points belonging to the majority class are thrown away. EasyEnsemble tries to explore the majority class data points that are overlooked by undersampling [209]. The working of the EasyEnsemble technique is described as follows. Let us consider a training dataset T with M minority class data points and N majority class data points. The undersampling method randomly selects N_T majority class data points from N where $|N_T| < |N|$. Generally, in undersampling, $|N_T| = |M|$ and for a highly imbalanced dataset, $|N_T|$ is far less than $|N|$. EasyEnsemble, sample multiple subsets $N_1, N_2, ..., N_M$ from N. For each subset, N_j $(1 \leq j \leq M)$ a classifier C_i (i.e., AdaBoost) is trained using N_j and all of M. The outcomes of all AdaBoost classifiers are then voted and the final output is returned.

- ***BalanceCascade***

  BalanceCascade trains multiple classifiers in a sequential manner. consider a training dataset T with M minority class data points and N majority class data points [209]. The undersampling method randomly selects N_T majority class data points from N where $|N_T| < |N|$. BalanceCascade samples multiple subsets $N_1, N_2, ..., N_M$ from N. For each subset, N_j $(1 \leq j \leq M)$ the first classifier C_i (i.e., AdaBoost) is trained using N_j and all of M. After C_i is

trained, all the data points D_i that are correctly classified by C_i are removed from N. Thus, size of N keep on reducing after classifier C_i is trained. The final classifier would the conjunction of all $(C_i)_{i=1,2....T}$.

5.3 Research Background

This section gives an overview of independent and dependent variables, model development, and evaluation procedure followed in this chapter.

5.3.1 Independent and Dependent Variables

Eighteen OO metrics are used as independent variables in this chapter. These metrics consist of six metrics from C&K metrics suite (DIT, NOC, RFC, WMC, CBO, and LCOM), five metrics from the QMOOD metrics suite (MOA, DAM, MFA, CAM, and NPM), afferent and efferent coupling (Ca & Ce) metrics along with AMC, SLOC, LCOM3, IC, and CBM metrics. Maintainability is the dependent variable. The detailed explanation of the variables of this chapter may be referred from Chapter 2.

5.3.2 Empirical Data Collection and Preprocessing

The data is collected from eight Apache open-source systems described in Chapter 2. The same datasets were used in Chapter 4 for developing SMP models using ML techniques. The data collection process is discussed in Chapter 2 in detail. After data collection, pre-processing was performed to remove outliers from each dataset.

5.3.3 Prediction Model Development and Evaluation

Next, we developed SMP models using ten-fold cross-validation to predict software maintainability. Considering the essence to deal with imbalanced datasets, prediction models were developed using EIDP techniques. The detailed description of these techniques is given in Section 5.2. Apart from developing models using EIDP techniques, classic ensembles AdaBoost and Bagging are also investigated in this chapter. G-Mean and Balance metrics are used to evaluate the predictive performance of the developed models.

5.3.4 Statistical Analysis and Hypothesis Evaluation

The results of the chapter are statistically evaluated using two non-parametric tests (i.e. Friedman test and Wilcoxon signed-rank test). RQ1 assesses the capabilities of classic ensembles AdaBoost and Bagging for developing SMP models. Wilcoxon test is used in RQ1 for determining the difference in the performance of AdaBoost and Bagging from base learner C4.5. Furthermore, a comprehensive statistical investigation is performed in RQ3 to determine the best ensemble technique in the category of Bagging-based, Boosting-based, and Hybrid ensembles with the aid of the Friedman test. The Wilcoxon signed-rank test is conducted in RQ3 to determine pairwise differences between the capabilities of the best ensemble technique in each category with all other techniques investigated in that category using G-Mean and Balance.

5.4 Results and Analysis

This section describes the answers to the RQ's of the chapter and discusses the obtained results

5.4.1 Answer Specific to RQ1

We first developed SMP models using classic ensembles AdaBoost [104] and Bagging [105]. The performance of constructed models is evaluated using G-Mean and Balance. The performance of the developed model is considered good if G-Mean and Balance are equal to greater than 50%. The reason behind choosing this threshold value of 50% for G-Mean and Balance is as follows: for a class-imbalanced dataset, it is challenging to develop a model with high Balance and G-Mean. A high Balance value means the model is giving TP rate equal to 1 (correctly predicting all low maintainability classes) and an FP rate equal to 0. However, for imbalanced SMP datasets, the model is unable to effectively learn the characteristics of low and high maintainability class data points appropriately. That is why a Balance threshold of 50% is taken for designating the model accurate. Similarly, we consider the G-Mean threshold of 50% as accurate. We also compare and analyze the performance of models developed using classic ensembles with the base classifier C4.5. The performance of the developed models is depicted in Table 5.1.

Table 5.1: Performance of SMP Models Developed using Classic Ensembles and Base Classifier

Dataset	**G-Mean**			**Balance**		
	C4.5	AdaBoost	Bagging	C4.5	AdaBoost	Bagging
Betwixt	26.79	52.39	26.95	34.5	49.97	34.51
Bcel	46.46	52.45	52.36	44.97	48.93	48.92
Io	0.00	62.30	42.55	29.29	57.52	42.15
Ivy	42.01	37.07	26.68	41.91	39.33	34.34
Jcs	70.94	52.45	52.37	65.89	48.93	48.92
Lang	77.89	53.29	67.56	73.69	50.15	63.15
Log4j	56.69	57.49	52.56	52.79	54.24	49.44
Ode	39.44	45.04	39.52	40.45	44.12	40.45
Avg.	45.03	51.56	45.06	47.93	49.14	45.23
Avg. indicates average						

As described in Table 5.1, that G-Mean values for models developed using the AdaBoost ensemble is either less than 50% or slightly above 50% except for two datasets: Apache Io and Apache Log4j. Similarily, G-Mean values for Bagging ensemble have been reported either less than 50% or slightly above 50% on all datasets except Apache Lang. Also, the Balance values of SMP models developed using the AdaBoost ensemble are reported either less than 50% or slightly above 50% except for the Apache Lang dataset. In the same way, the Balance values for SMP models developed using Bagging are less than 50% for all datasets except for Apache Lang. Also, it evident from Table 5.1 that the mean G-Mean values attained by AdaBoost and Bagging on all eight datasets are 51.56% and 45.06%, respectively. Similarly, the mean Balance values on all datasets used in the chpater were 49.14%, 45.23% for AdaBoost, and Bagging, respectively. We further compare the performance of AdaBoost and Bagging with C4.5. The average performance of SMP models developed using C4.5 on eight datasets under investigation in this chapter was 45.03% and 47.93% concerning G-Mean and Balance respectively. Therefore, it is evident from Table 5.1 that AdaBoost improves the performance of SMP models over the C4.5 to a minimal extent but the predictive performance of SMP models built with Bagging and C4.5 is approximately the same.

Table 5.2: Wilcoxon Test Results for Classic Ensembles and Base classifier

Technique pair	Using G-Mean	Using Balance
C4.5-AdaBoost	0.484	0.674
C4.5-Bagging	0.674	0.449

To further analyze, if there exists a significant difference in the performance of SMP models developed using classic ensembles and base classifiers, we test the following hypothesis.

- Null hypothesis (H0): There is a significant difference in the performance of

SMP models developed using a base classifier and classic ensembles: AdaBoost and Bagging for imbalanced SMP datasets.

- Alternate Hypothesis (Ha): There is no significant difference in the performance of SMP models developed using a base classifier and classic ensembles: AdaBoost and Bagging for imbalanced SMP datasets.

To validate the above hypothesis, we do a pair-wise comparison on the performance of the base classifier and individual ensembles used in the chapter using the Wilcoxon test concerning G-Mean and Balance at a level of significance, $\alpha = 0.05$. The results of the Wilcoxon signed-rank test are given in Table 5.2. As shown in Table 5.2, for both of the pairs of classic ensembles and base classifier, the p-values are greater than 0.05 which means of the Wilcoxon test are not significant. Therefore, we reject the H0 which says that there is a substantial difference in the performance of SMP models developed using a base classifier and classic ensembles. The alternate hypothesis (Ha) is accepted and leads to the conclusion that the performance of SMP models developed using classic ensembles does not improve statistically over the SMP models developed using base classifiers for imbalanced datasets.

Analysis of RQ1

Ensemble techniques combine multiple classifiers to make a single consolidated decision. The ensembles improve the performance for a classification task because by combining multiple classifiers the error of a single classifier will be taken care of by the other individual classifiers. However, on analyzing the results of RQ1, we found that, for imbalanced datasets, classic ensembles (AdaBoost and Bagging) are not competent techniques to develop SMP models i.e., the performance of AdaBoost and Bagging does not improve statistically over their base learner, C4.5 classifier.

5.4.2 Answer Specific to RQ2

To answer this RQ, we developed SMP models using EIDP. The three types of EIDP have investigated in this chapter are Bagging-based ensembles, Boosting-based ensembles, and hybrid ensembles. Tables 5.3 and 5.4 show the results of SMP models developed using Bagging-based ensembles with respect to G-Mean and Balance. Table 5.5 shows the G-Mean and Balance results for SMP models developed using the Boosting-based ensembles.

Table 5.3: G-Mean Results for Models Developed using Bagging-based Ensembles

Technique	**Dataset**							
	Bcel	**Betwixt**	**Io**	**Ivy**	**Jcs**	**Lang**	**Log4j**	**Ode**
UnderBagging	72.62	66.46	79.28	71.95	72.62	81.48	70.84	69.89
OverBagging	67.35	52.02	53.26	48.37	67.62	78.19	64.49	51.27
SMOTEBagging	77.38	67.35	65.63	55.92	77.38	69.15	63.74	65.43
MSMOTEBagging	51.27	68.85	62.44	56.17	72.14	76.63	66.80	57.24
IIvotes	61.27	51.61	62.04	40.88	61.27	79.42	70.31	42.81
UnderOverBagging	74.37	64.85	59.69	58.52	74.37	81.48	70.14	62.25

Table 5.4: Balance Results for Models Developed using Bagging-based Ensembles

Technique	**Dataset**							
	Bcel	**Betwixt**	**Io**	**Ivy**	**Jcs**	**Lang**	**Log4j**	**Ode**
UnderBagging	72.19	66.08	79.27	71.52	72.19	80.26	70.83	69.87
OverBagging	66.00	51.02	50.35	46.77	64.13	75.71	61.89	48.94
SMOTEBagging	76.94	66.00	63.31	54.71	76.94	65.51	60.61	63.5
MSMOTEBagging	48.79	66.90	57.54	53.8	68.26	73.38	63.84	53.86
IIvotes	56.72	49.73	57.49	41.74	56.72	77.8	68.23	42.82
UnderOverBagging	71.69	64.18	56.88	56.12	71.69	80.26	68.86	59.46

Table 5.5: G-Mean and Balance Results for Models Developed using Boosting-based Ensembles

Technique	**G-Mean**							
	Bcel	**Betwixt**	**Io**	**Ivy**	**Jcs**	**Lang**	**Log4j**	**Ode**
DataBoost	60.59	41.01	31.62	45.27	60.59	71.74	70.86	48.51

EUSBoost	75.03	68.76	80.54	62.11	60.59	68.96	60.66	69.42
SMOTEBoost	67.62	62.52	68.14	51.22	67.62	78.96	68.98	58.24
RUSBoost	75.03	66.26	71.01	51.12	75.03	81.01	73.98	69.42
MSMOTEBoosting	51.27	68.85	62.58	40.17	51.27	71.38	64.18	50.05
Technique	**Balance**							
	Bcel	**Betwixt**	**Io**	**Ivy**	**Jcs**	**Lang**	**Log4j**	**Ode**
DataBoost	59.96	42.02	36.36	44.36	59.96	71.44	70.32	46.57
EUSBoost	73.99	66.84	80.54	61.67	59.96	65.45	57.42	68.75
SMOTEBoost	64.13	61.00	64.29	49.16	64.13	77.55	68.78	54.99
RUSBoost	73.99	66.17	69.55	50.75	73.99	79.97	73.24	68.75
MSMOTEBoosting	48.79	66.9	57.55	41.52	48.79	68.03	61.75	47.79

In Table 5.6, G-Mean and Balance results for SMP models developed using hybrid ensembles are reported. On analyzing Table 5.3, we found that G-Mean values for Bagging-based ensembles ranged from 40.88% to 85.65%. In 95.31% of the cases, G-Mean values are more than 50%. The G-Mean values of the models developed using the UnderBagging technique are ranged from 66.46% to 81.48%.

Table 5.6: G-Mean and Balance Results for Models Developed using Hybrid Ensembles

Technique	**G-Mean**							
	Bcel	**Betwixt**	**Io**	**Ivy**	**Jcs**	**Lang**	**Log4j**	**Ode**
BalanceCascade	71.87	60.33	84.09	70.30	71.87	71.98	61.30	65.09
EasyEnsemble	67.23	60.33	84.09	74.22	67.23	70.27	67.34	64.79
Technique	**Balance**							
	Bcel	**Betwixt**	**Io**	**Ivy**	**Jcs**	**Lang**	**Log4j**	**Ode**
BalanceCascade	71.87	60.17	83.28	70.10	71.87	71.65	57.61	65.09
EasyEnsemble	67.22	60.17	83.28	73.50	67.22	70.14	67.03	64.79

For OverBagging the range of G-Mean values was 48.37%-78.19% over all the datasets. The G-Mean results are in the range of 51.27% to 76.63% and 40.88% to 79.42% for SMOTEBagging and MSMOTEBagging techniques respectively. The SMP models developed using the UnderOverBagging technique gave G-Mean results in the range of 58.52%-81.48%. On evaluating the values of G-Mean on individual

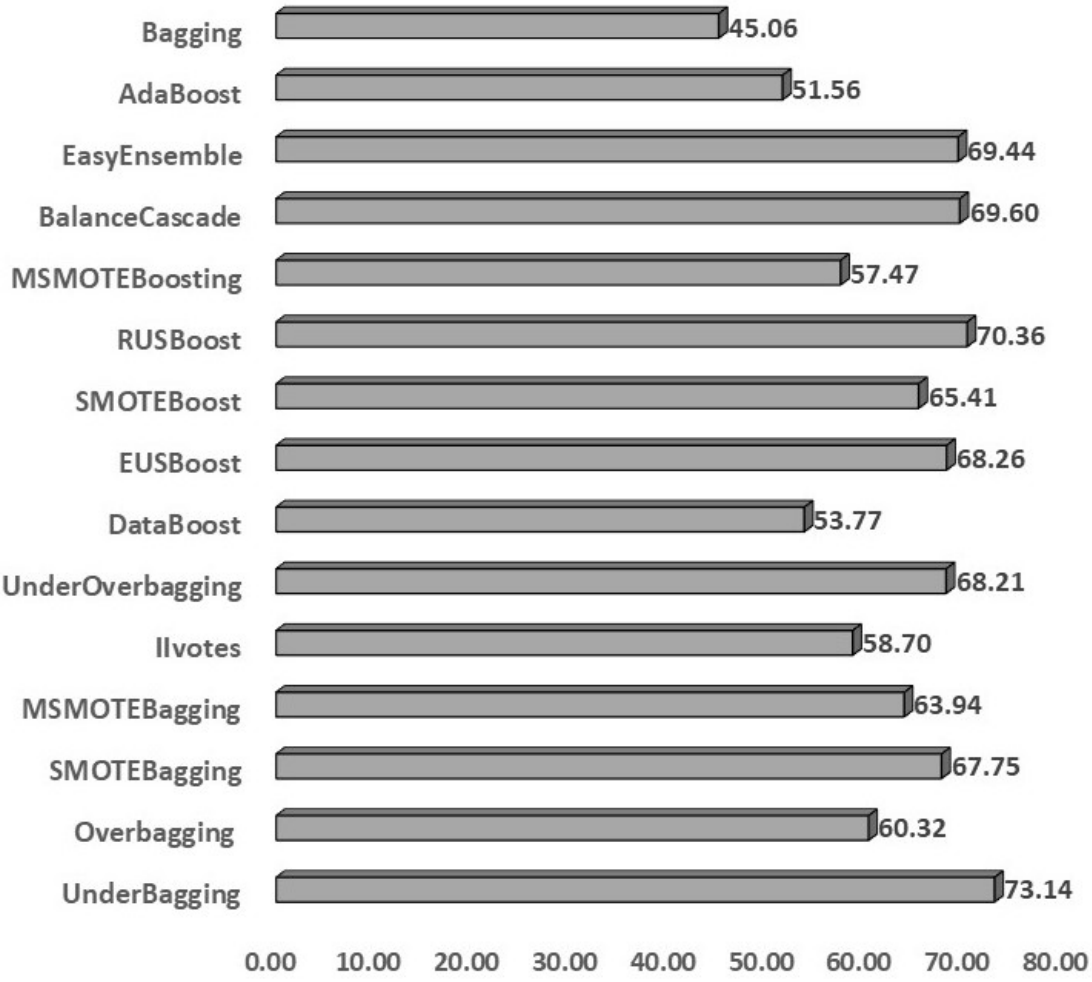


Figure 5.2: Average G-Mean of different Ensemble Techniques

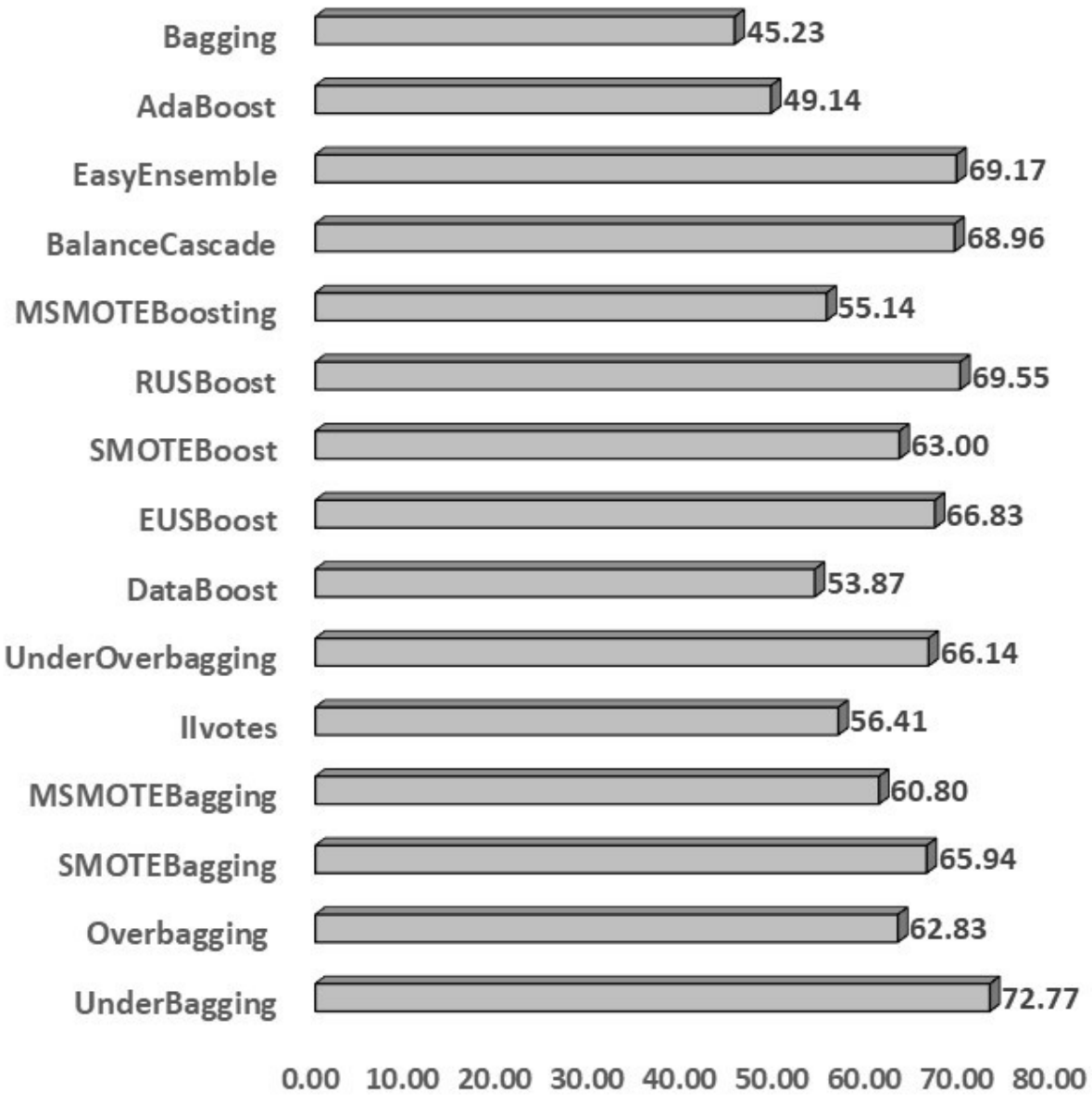


Figure 5.3: Average Balance of different Ensemble Techniques

datasets, it was noticed that all Bagging-based ensembles except SMOTEBagging gave the best G-Mean results on the Apache Lang dataset. The best G-Mean performance of the SMOTEBagging technique was observed on the Apache Jcs dataset. The Balance results of the SMP models developed using all Bagging-based ensembles ranged from 41.74% to 85.14%. In 90.62% of the cases, the Balance results are greater than 50%. All Bagging techniques except SMOTEBagging obtained the best Balance results on the Apache Lang dataset. The underBagging technique obtained the best Balance result on the Apache IO dataset. The performance of the SMOTEBagging technique was best on the Apache Jcs dataset in terms of Balance.

Similarly, on analyzing the performance of the Boosting-based ensembles, we found that G-Mean results (Table 5.5) of these techniques ranged from 31.62% to 81.01% over all the datasets. In 88% of the cases, Balance results were greater than 50%. The Balance results of DataBoost techniques were in the range of 31.62% to 71.74%. The EUSBoost technique obtained Balance values ranging from 60.59% to 80.54%. The Balance results for SMOTEBoost and MSMOTEBoosting techniques were in the range of 51.22%-78.96% and 40.17%-71.38%, respectively. The performance of RUSBoost was in the range of 51.12%-81.01% in terms of Balance. Again, it was observed that all Boosting-based ensembles except RUSBoost performed best on the Apache Lang dataset in terms of G-Mean.

On analyzing the performance of Boosting-based ensembles in terms of Balance (Table 5.5), we found that the performance of SMP models on all datasets under investigation in the chapter ranged from 36.36% to 80.54% and to further analyze; it was observed that in 77.50% of the cases Balance values were greater than 50%. On further analysis, we found that the performance of Boosting-based ensembles: DataBoost, SMOTEBoost, MSMOTEBoost, and RUSBoost was best on the Apache Lang dataset. The EUSBoost performance in terms of Balance was best on the Apache IO dataset. There were two hybrid ensembles, namely, BalanceCascade and

EasyEnsemble analyzed in this chapter. The performance of SMP models developed using BalanceCascade and EasyEnsembles was in the range of 60.33%-84.09% and 57.61%-83.28% for G-Mean and Balance, respectively (Table 5.6).

Figures 5.2 and 5.3 show the comparison of the performance of the SMP models developed using Bagging-based ensembles, Boosting-based and hybrid ensembles with the classic ensembles in terms of average G-Mean and Balance. According to Figure 5.2, the average G-Mean values of the SMP models developed using AdaBoost and boosting were 45.06% and 51.06%, respectively. The average G-Mean values of various Bagging-based ensembles ranged from 58.70% to 73.14%. Except for IIvotes, the average G-Mean of all Bagging-based ensembles was greater than 60%. It is quite clear from Figure 5.2 that the average G-Mean results of models developed using various Boosting-based ensembles are very good. The average G-Mean values of various Boosting-based ensembles ranged from 53.77% to 70.36%. Also, it can be seen from Figure 5.2 that the hybrid ensembles also gave an exceptionally good performance. The average G-Mean performance of both of the hybrid ensembles is nearly 70%. It can be seen from Figure 5.3 that classic ensembles gave an average Balance value of merely 49.14% and 45.23%, but the EIDP gave very good results. It is quite clear from Figure 5.3 that the majority of the Bagging-based ensembles and Boosting-based ensembles gave average Balance values greater than 60%. Also, average Balance results for both of the hybrid ensembles used in the chapter are quite near to 70%.

Analysis of RQ2

On analyzing the performance of various EIDP techniques according to G-Mean and Balance values, it is evident that the SMP models built using all of such techniques are better than those developed by the classic ensembles. This is because the EIDP incorporates the advantages of both the data resampling and constituent ensembles to perform effectively for imbalanced data.

5.4.3 Answer Specific to RQ3

To figure out whether the different EIDP techniques significantly improve the performance of SMP models built using an imbalanced dataset, we used the Friedman test. The smaller the rank acquired by the given technique, the better that technique is considered. We find the best techniques among each category of EIDP techniques and examine whether those techniques significantly improve the performance of SMP models over other techniques in that category.

5.4.3.1 *Statistical Analysis for Different Bagging-based Ensembles*

To analyze the best techniques among the Bagging-based ensembles, we evaluate the following hypothesis by conducting the Friedman test. We compare the 10 techniques (8 Bagging-based ensembles and 2 classic ensembles) for performance metrics: G-Mean and Balance.

- *Null Hypothesis* $(H0_1)$: SMP models built using Bagging-based ensembles: UnderBagging, OverBagging, SMOTEBagging, MSMOTEBagging, IIvotes, UnderOverBagging, and classic ensembles: AdaBoost and Bagging do not show a significant difference in terms of G-Mean.

- *Alternate Hypothesis* (Ha_1): SMP models built using Bagging-based ensembles techniques: UnderBagging, OverBagging, SMOTEBagging, MSMOTEBagging, IIvotes, UnderOverBagging, and classic ensembles techniques: AdaBoost and Bagging show a significant difference when the performance of the model is analyzed using G-Mean.

- *Null Hypothesis* $(H0_2)$: SMP models built using Bagging-based ensembles techniques: UnderBagging, OverBagging, SMOTEBagging, MSMOTEBagging, IIvotes, UnderOverBagging, and classic ensembles: AdaBoost and Bagging

techniques do not show a significant difference when the performance of the model is analyzed using Balance.

- *Alternate Hypothesis* (Ha_2): SMP models built using Bagging-based ensembles techniques: UnderBagging, OverBagging, SMOTEBagging, MSMOTEBagging, IIvotes, UnderOverBagging, and classic ensembles techniques: AdaBoost and Bagging show a significant difference when the performance of the model is analyzed using Balance.

Table 5.7: Friedman Test Results for Bagging-based Ensembles

Technique	**Mean Rank using G-Mean**	**Mean Rank using Balance**
UnderBagging	1.81	1.44
UnderOverBagging	2.94	3.06
SMOTEBagging	3.00	3.00
MSMOTEBagging	3.88	4.00
OverBagging	5.25	5.00
IIVotes	5.25	5.13
AdaBoost	6.25	6.50
Bagging	7.63	7.88

The UnderBagging technique got the best rank followed by UnderOverBagging techniques using G-Mean. The p-value obtained after the Friedman test was 0.00. As the results of the test are significant at $\alpha = 0.05$, the null hypothesis $H0_1$ is rejected. This implies that the different techniques behave differently when SMP models developed using these techniques are analyzed using G-Mean. Table 5.7 also shows the mean ranks obtained by different Bagging-based ensembles and classic ensembles after the Freidman test when SMP models are evaluated using Balance. According to Table 5.7, the best rank is obtained by the UnderBagging technique followed by UnderOverBagging techniques for Balance. The worst ranks are obtained by classic ensembles. The p-value obtained was 0.00, which indicates that the results of the Friedman test are significant and thus the null hypothesis $H0_2$ is also rejected.

This implies that the techniques examined using the Friedman test behave differently when the SMP models developed using these techniques are evaluated based on Balance. The Friedman test gave significant results, stating UnderBagging as the best technique, and we further carry out post-hoc analysis using the Wilcoxon test to evaluate the pairwise differences amongst the performance of the different techniques with respect G-Mean and Balance. We would evaluate the following hypothesis using the Wilcoxon test at $\alpha = 0.05$.

- *Null Hypothesis* $(H0_3)$: The UnderBagging technique and all other Bagging-based ensembles and classic ensembles do not show a significant difference in the performance when SMP models are analyzed using G-Mean.
- *Alternate Hypothesis* (Ha_3): The UnderBagging technique and all other Bagging-based ensembles and classic ensembles show a significant difference in the performance when SMP models are analyzed using G-Mean.
- *Null Hypothesis* $(H0_4)$: The UnderBagging technique and all other Bagging-based ensembles and classic ensembles do not show a significant difference in the performance when SMP models are analyzed using Balance.
- *Alternate Hypothesis* (Ha_4): The UnderBagging technique and all other Bagging-based ensembles and classic ensembles show a significant difference in the performance when SMP models are analyzed using Balance.

Table 5.8 reports the pair-wise comparison with the help of the Wilcoxon test between the performance of the UnderBagging technique with all other Bagging-based ensembles and classic ensembles based on G-Mean and Balance.

Table 5.8: Wilcoxon Signed Rank Test Results for Bagging-based Ensembles

Technique Pair	**Using G-Mean**	**Using Balance**
UnderBagging Vs. OverBagging	Sig+	Sig+
UnderBagging Vs. SMOTEBagging	Not Sig+	Not Sig+
UnderBagging Vs. MSMOTEBagging	Sig+	Sig+
UnderBagging Vs. IIvotes	Sig+	Sig+
UnderBagging Vs. UnderOverBagging	Not Sig+	Not Sig+
UnderBagging Vs. AdaBoost	Sig+	Sig+
UnderBagging Vs. Bagging	Sig+	Sig+

As shown in Table 5.8, the UnderBagging technique significantly outperforms ($p-value < 0.05$) all other Bagging-based ensembles and classic ensembles respect to performance measures G-Mean and Balance over all the datasets except for three cases. These three cases are SMOTEBagging and UnderOverBagging. Therefore, except for SMOTEBagging, and UnderOverBagging, for all other techniques, we accept the alternate hypothesis: Ha_3 and Ha_4 and state that the UnderBagging technique significantly outperformed all other Bagging-based ensembles techniques except for SMOTEBagging, and UnderOverBagging.

5.4.3.2 *Statistical Analysis for Different Bootsing-based Ensembles*

We evaluate the following hypothesis by conducting the Friedman test. The objective of the Friedman test is to find the best techniques among Boosting-based ensembles. We compare the seven techniques (five Boosting-based ensembles and two classic ensembles) with respect to G-Mean and Balance.

- *Null Hypothesis* $(H0_5)$: SMP models developed using Boosting-based ensembles techniques RUSBoost, EUSBoost, SMOTEBoost, MSMOTEBoost, DataBoost, and classic ensembles techniques AdaBoost and Bagging do not show a significant difference when G-Mean is used to evaluate the performance of the models.

- *Alternate Hypothesis* (Ha_5): SMP models developed using Boosting-based ensembles RUSBoost, EUSBoost, SMOTEBoost, MSMOTEBoost, DataBoost, and classic ensembles techniques AdaBoost and Bagging show a significant difference when G-Mean is used to evaluate the performance of the models.

- *Null Hypothesis* $(H0_6)$: SMP models developed using Boosting-based ensembles techniques: RUSBoost, EUSBoost, SMOTEBoost, MSMOTEBoost, DataBoost, and classic ensembles AdaBoost and Bagging techniques do not show a significant difference when Balance is used to evaluate the performance of the models.

- *Alternate Hypothesis* (Ha_6): SMP models developed using Boosting-based ensembles techniques: RUSBoost, EUSBoost, SMOTEBoost, MSMOTEBoost, DataBoost, and classic ensembles techniques: AdaBoost and Bagging show a significant difference Balance is used to evaluate the performance of the models.

The above hypothesis is evaluated by the Friedman test at $\alpha = 0.05$. The Friedman test is conducted by taking the G-Mean and Balance results achieved by all the Boosting-based ensembles techniques and classic ensembles techniques on all datasets in this chapter. Table 5.9 reports the mean rank obtained by various techniques after applying the Friedman test on G-Mean and Balance.

Table 5.9: Friedman Test results for Boosting-based Ensembles

Technique	**Mean Rank using G-Mean**	**Mean Rank using Balance**
RUSBoost	1.75	1.63
EUSBoost	2.56	2.56
SMOTEBoost	2.75	2.88
MSMOTEBoost	4.38	4.38
DataBoost	4.44	4.44
AdaBoost	5.63	5.63
Bagging	6.50	6.50

As reported in Table 5.9 that the best rank was secured by RUSBoost followed by EUSBoost when the performance of SMP models is evaluated by Balance and G-Mean. In both of the cases, the p-value obtained after the Friedman test was 0.00. As the results of the test are significant at $\alpha = 0.05$, the null hypothesis $H0_5$ and $H0_6$ are rejected. The rejection of $H0_5$ and $H0_6$ leads to the conclusion that different techniques behaved differently when SMP models developed using these techniques are evaluated by Balance and G-Mean. As the results of the Friedman test result are significant, stating RUSBoost as the best technique, we further carry out Wilcoxon signed-rank test as the post-hoc analysis to evaluate the pairwise differences amongst the performance of the different Boosting-based ensembles and classic ensembles. The following hypothesis are evaluated for this purpose.

- *Null Hypothesis* $(H0_7)$: There is no significant difference amongst the performance of the RUSBoost technique, all other Boosting-based ensembles and classic ensembles when SMP models are evaluated using G-Mean.

- *Alternate Hypothesis* (Ha_7): There is a significant difference amongst the performance of the RUSBoost technique, all other Boosting-based ensembles and classic ensembles when SMP models are evaluated using G-Mean.

- *Null Hypothesis* $(H0_8)$: There is no significant difference in the performance of the RUSBoost technique, all other Boosting-based ensembles and classic ensembles when SMP models are evaluated using Balance.

- *Alternate Hypothesis* (Ha_8): There is a significant difference amongst the performance of the RUSBoost technique, all other Boosting-based ensembles and classic ensembles when SMP models are evaluated using G-Mean.

Table 5.10 reports the pair-wise comparison with the help of the Wilcoxon signed-rank test between the performance of the RUSBoost technique with all other Boosting-

based ensembles and classic ensembles with respect to performance measures G-Mean and Balance.

Table 5.10: Wilcoxon Signed Rank Test Results for Boosting-based Ensembles

Technique Pair	**Using G-Mean**	**Using Balance**
RUSBoost Vs. EUSBoost	Not Sig+	Not Sig+
RUSBoost Vs. SMOTEBoost	Sig+	Sig+
RUSBoost Vs. MSMOTEBoost	Sig+	Sig+
RUSBoost Vs. DataBoost	Sig+	Sig+
RUSBoost Vs. AdaBoost	Sig+	Sig+
RUSBoost Vs. Bagging	Sig+	Sig+

As shown in Table 5.10, the RUSBoost technique significantly outperforms $(p - value < 0.05)$ all other Boosting-based ensembles and classic ensembles respect to performance measures G-Mean and Balance over all the datasets except for EUSBoost. Therefore, except for EUSBoost, for all other techniques, we accept the alternate hypothesis Ha_7 and Ha_8. We report that the RUSBoost technique significantly outperformed all other Boosting-based ensembles and classic ensembles.

5.4.3.3 Statistical Analysis of Hybrid Ensembles

To evaluate the best techniques among the hybrid ensembles, we would evaluate the following hypothesis with the help of the Friedman test. Here, we compare four techniques (two hybrid ensembles and two classic ensemble techniques).

- *Null Hypothesis* $(H0_9)$: SMP models built using hybrid ensembles BalanceCascade, EasyEnsemble, and classic ensembles techniques: AdaBoost and Bagging do not show a significant difference when the performance of the model is analyzed using G-Mean.
- *Alternate Hypothesis* (Ha_9): SMP models built using hybrid ensembles BalanceCascade, EasyEnsemble, and classic ensembles techniques AdaBoost and

Bagging show a significant difference when the performance of the model is analyzed using G-Mean.

- *Null Hypothesis* ($H0_{10}$): SMP models built using hybrid ensembles BalanceCascade, EasyEnsemble, and classic ensembles techniques AdaBoost and Bagging do not show a significant difference when the performance of the model is analyzed using Balance.

- *Alternate Hypothesis* (Ha_{10}): SMP models built using hybrid ensembles techniques BalanceCascade, EasyEnsemble, and classic ensembles techniques AdaBoost and Bagging show a significant difference when the performance of the model is analyzed using Balance.

We evaluated the above hypothesis by applying the Friedman test at $\alpha = 0.05$. Table 5.11 reports the mean rank obtained by hybrid ensembles and classic ensembles after the Friedman test according to G-Mean and Balance. Rank one was obtained by the BalanceCascade technique followed by the EasyEnsemble technique both for G-Mean and Balance. The test statistic obtained after the Friedman test was 0.00.

Table 5.11: Friedman Test Results for Hybrid Ensembles

Technique	**Mean Rank using G-Mean**	**Mean Rank using Balance**
BalanceCascade	1.38	1.38
EasyEnsemble	1.63	1.63
AdaBoost	3.13	3.13
Bagging	3.88	3.88

As the results of the test are significant at $\alpha = 0.05$, the null hypothesis, $H0_9$, and $H0_{10}$ are rejected which means that different techniques behave differently when SMP models developed using these techniques are analyzed using G-Mean and Balance. As the results of the Friedman test result are significant, BalanceCascade has emerged

as the best technique, we further carry out post-hoc analysis using the Wilcoxon signed-rank test to evaluate the pairwise differences amongst the performance of the different techniques with respect to the performance measures G-Mean and Balance. We would evaluate the following hypothesis using the Wilcoxon signed-rank test at a level of significance, $\alpha = 0.05$.

- *Null Hypothesis* $(H0_{11})$: There is no significant difference in the performance of BalanceCascade and EasyEnsemble and classic ensembles when SMP models are evaluated using G-Mean.
- *Alternate Hypothesis* (Ha_{11}): There is a significant difference amongst the performance of BalanceCascade and EasyEnsemble and classic ensembles when SMP models are evaluated using G-Mean.
- *Null Hypothesis* $(H0_{12})$: There is no significant difference in the performance of BalanceCascade and EasyEnsemble and classic ensembles when SMP models are evaluated using Balance.
- *Alternate Hypothesis* (Ha_{12}): There is a significant difference amongst the performance of BalanceCascade and EasyEnsemble and classic ensembles when SMP models are evaluated using Balance.

Table 5.12 reports the pair-wise comparison with the help of Wilcoxon signed-rank test between the performance of BalanceCascade technique with EasyEnsemble and classic ensembles with respect to performance measures G-Mean and Balance. As per Table 5.12, no significant difference is noted in the predictive performance of hybrid ensembles, BalanceCascade and EasyEnsemble $(p - value < 0.05)$ both for G-Mean and Balance-null hypothesis $H0_{11}$ and $H0_{12}$ are accepted) but BalanceCascade techniques are significantly better than that of both of the classic ensembles.

Table 5.12: Wilcoxon Signed Rank Test Results for Hybrid Ensembles

Technique Pair	**Using G-Mean**	**Using Balance**
BalanceCascade Vs. EasyEnsemble	Not Sig+	Not Sig+
BalanceCascade Vs. AdaBoost	Sig+	Sig+
BalanceCascade Vs. Bagging	Sig+	Sig+

Analysis of RQ3

On statistically examining the three categories of EIDP techniques, we found that for imbalanced data, EIDP techniques are better in terms of their predictive performance as compared to classic ensembles. It is discovered that UnderBagging techniques performed best in the category of Bagging-based ensembles and RUSBoost techniques performed best in the Boosting-based ensembles. This is the positive synergy between the random undersampling and ensembles that have contributed toward the excellent performance of RUSBoost and UnderBagging. The performance of both hybrid ensembles is almost the same. Further, in the category of Bagging-based ensembles, the performance of SMOTEBagging is comparable with the UnderBagging technique. In the group of Boosting-based ensembles, the performance of EUSBoost is comparable to the RUSBoost technique.

5.5 Discussion

In this chapter we evaluated whether the use of ensemble methodology aggregated with data resampling i.e., EIDP techniques obtain effective results for predicting low maintainability classes effectively. The results indicate improved performance of the EDIP over the classic ensemble learners. The average G-Mean of AdaBoost and Bagging was observed to be 51.56 and 45.06 respectively. However, the average G-Mean performance of Bagging-based EDIP techniques were observed in the range of 60.32-73.14.

Similarly, the performance of Bagging-based EDIP techniques was in the range of 60.80-72.77 in terms of average Balance whereas the classis ensemble techniques AdaBoost and Bagging gave average Balance of 49.14 and 45.23 respectively. The Boosting-based EIDP techniques gave average G-Mean performance of SMP models in the range of 53.77 to 70.36 and average Balance in the range of 53.77-70.36. Both of hybrid EDIP techniques used in this chapter (EasyEnsemble and BalanceCascade) improved the performance of SMP models in terms of G-Mean and Balance. Both of these hybrid EDIP techniques gave average G-Mean and Balance of nearly 70.00.

The improvement in the results was statistically evaluated using Friedman test and was found to be significant. The reason for favourable results of the EDIP techniques was the diversity introduced due to constituent data resampling techniques. Their diversity would result in an improved SMP models by predicting higher number of correct low maintainability classes. The Wilcoxon test results depicted an improvement in G-Mean and Balance performance measures using EDIP techniques over the classic ensemble techniques for developing SMP models from the imbalanced datasets.

Chapter 6

Empirical Evaluation of Search-Based Techniques for Software Maintainability Prediction with Imbalanced Data

6.1 Introduction

Various prediction models have been developed in the past literature to forecast the maintainability in the earlier phases of software development by using the historical datasets and different ML techniques [13, 13, 20, 56, 57]. From the last few years, in addition to ML techniques, SB techniques are gaining popularity in predictive modelling. In software engineering domain SB techniques have been applied to develop effective prediction models to predict the defective and change-prone classes [17, 210]. These techniques aid the developers and project managers in determining

optimal solutions for constructing effective prediction models. SB techniques search the entire search solution space and give an optimal or nearly optimal solution for the problem in hand. The prediction models developed using SB techniques have delivered better performance than that of ML techniques for various prediction problems. In SB techniques, the search process is directed by a fitness function that establishes the suitability of a solution [211].

Harman and Jones [212] and Harman [213] encouraged the application of SB techniques in the software engineering predictive modeling domain as these techniques are effective in handling constraints and conflicts. Furthermore, these techniques are also effective in dealing with noisy, somewhat inaccurate, and inadequate datasets. According to Harman and Jones [212] robustness and simple problem-solving approach are other prime advantages of the SB technique. These techniques conduct the global search competently by avoiding getting stuck in local optima. The study by Harman and Clark [214] has advocated that the evaluation metrics such as the accuracy, specificity, sensitivity can be used as fitness functions by SB techniques and henceforth can be used to develop prediction models for software. Therefore, keeping in mind the recently acknowledged association amongst the SB techniques and predictive modeling, this chapter evaluates the effectiveness of SB techniques for developing prediction models for software maintainability.

Also, according to systematic literature review conducted by Malhotra et al. [215], SB techniques exhibited effective results for developing predictive models in software engineering domain, however there are very few studies that have evaluated these techniques for SMP. Therefore, more empirical studies are required, which evaluate and compare the effectiveness of SB techniques for SMP. In this chapter, we compare the performance of SB techniques with ML techniques. The performance of fourteen SB techniques (BIOHEL, CHC, CPSO, GGA, GA-ADI, GA-Int, IGA, LWPSO, PBIL, MPLCS, UCS, XCS, SGA, SSMA) and ML techniques (C4.5, AdaBoost, Bagging,

C4.5, PART, RIPPER, SLIPPER, BNGE, EACH, RISE, KNN, LR, KSTAR, PUBLIC, CHI-RW) is appraised on eight open-source datasets used in the previous chapter for developing models. The imbalanced datasets before the application of the SB and ML are balanced by applying same data resampling techniques. To deal with the non-deterministic nature nature of SB techniques and provide unbiased predictions, we carried out several executions of SB techniques. In this chapter, we used performance measures G-Mean and Balance to assess the performance of SMP models. Also, the performance the models is statistically evaluated with the help of Friedman and the Wilcoxon signed-rank test. The following research questions are answered in this chapter.

- RQ1: What is the comparative performance of SMP models developed with SB and ML techniques with imbalanced data?
- RQ2: Does the performance of maintainability prediction models developed using SB techniques improve after employing data resampling methods and to what extent the performance of the models improves?
- RQ3: Which data resampling method best improves the performance of the SMP models developed from SB and ML techniques?
- RQ4: What is the comparative performance of ML and SB techniques after data resampling for developing SMP models?

This chapter is organized as follows: Section 6.2 states the experimental design and framework for conducting the empirical experiment. Section 6.3 states the results and analysis of the chapter. Finally, Section 6.4 states the discussion of the results. The results of this chapter are published in [216].

6.2 Elements of Experimental Design

This section discusses the various experimental design elements. The experiment evaluates the performance of SB techniques for developing SMP models. The performance of models developed by SB techniques is also compared with the models developed using ML techniques. Before developing the models using SB and ML techniques, the imbalanced datasets are balanced with the help data resampling techniques used in Chapter 4. This section explains the datasets and variables, and performance measures which are used to validate the experimental results.

6.2.1 Dependent and Independent Variables

In this chapter independent and dependent variables used are the same as that used in Chapter 4 and Chapter 5. However, to incorporate the correlation of independent variables, CFS is applied to select the best predictors out of independent variables in the datasets. CFS is described in Chapter 2. This chapter focuses on predicting whether the maintainability of a class is low or high i.e., maintainability is a binary dependent variable in this chapter.

6.2.2 Datasets

In this chapter, the same datasets have been used, which were used in Chapter 4 and Chapter 5 (i.e., Bcel, Betwixt, Io, Ivy, Jcs, lang, Log4j, Ode).

6.2.3 Data Resampling Techniques

The data resampling techniques investigated in Chapter 4 are considered in this chapter for balancing the datasets. These techniques are SMOTE, Adasyn, SafeSMOTE,

BSMOTE, SMOTE-ENN, SMOTE-TL, SPIDER, SPIDER II, ROS, RUS, NCL, and CNN.

6.2.4 Model Development and Evaluation

Model training and validation are two imperative processes in developing a prediction model. The training process includes a classification technique (SB or ML) to recognize classification rules which can successfully differentiate between low maintainability and high maintainability classes. The set of these rules formulate a prediction model. The validation process tests these rules to predict low maintainability and high maintainability classes. Training of models in this chapter has been done using fourteen SB techniques and fourteen ML techniques. The fitness functions and parameters used for each of these techniques are described in Chapter 2. Ten-fold cross-validation is employed as a validation method of each of the investigated classification techniques. The performance of the models is assessed using G-Mean and Balance. According to Ali et al. [217], it is essential to perform multiple iterations to effectively handle the stochastic nature of SB techniques. Hence, in this chapter, we developed SMP models using 10 runs of SB techniques and evaluated the median values of the performance measures attained over 10 runs. A similar practice has been followed by many researchers in the literature of search-based software engineering [210, 213, 218]. Also, the developed models are statistically evaluated using the Friedman test and Wilcoxon signed-rank test. For experimental simulation, the KEEL tool has been used. The default parameter settings of the KEEL tool are used for each technique. The parameter of each technique is stated in Chapter 2. According to Arcuri and Fraser [219], the use of default parameter settings should be used as the parameter tuning is an expensive procedure. Arcuri and Fraser [219] also advocated that parameter tuning may always not produce significant improvement in results in

all models. Therefore, we used the default parameter setting of each of the SB and ML techniques for developing prediction models.

6.3 Results and Analysis

This section describes the results of the chapter.

6.3.1 CFS Results

The results on each dataset after application of the CFS method are shown in Table 6.1. The WMC metric was selected in five datasets. In six out of eight datasets SLOC metric was selected for developing SMP models. The LCOM and CAM metrics were selected by five datasets. The RFC metric was selected by Betwixt and Ode datasets. The NOC and DIT metrics were not selected for any of the datasets that show less use of the inheritance metrics (NOC and DIT) while developing models for predicting software maintainability. As the selected metrics are dependent on characteristics of datasets and change statistics due to which the features selected with CFS are different for each dataset.

6.3.2 Results Specific to RQ1

To answer this RQ, we developed SMP models by using the imbalanced datasets. The ML and SB techniques described were used to create the models by providing the datasets using ten-fold cross-validation. The performance of developed models is assessed using G-Mean and Balance metrics (shown in Table 6.2 and Table 6.3). In Table 6.2 and Table 6.3, the first fourteen rows show the G-Mean results of SMP models developed using SB techniques and the next fourteen rows depict the performance of models developed using the ML technique. For SB techniques, we

developed models by running each technique ten times to take care of the scholastic nature of these techniques and reported the median values for the performance metrics G-Mean and Balance.

For each of 14 ML techniques on 8 datasets, 14 $\times$ 8 =112 models were developed. For each of 14 SB techniques on 8 datasets by running each SB technique 10 times, 14 $\times$ 8 $\times$ 10 = 1120 models were developed. Thus, in this experiment, we assessed the performance of 1232 (112+1120) SMP models. On analyzing the G-Mean results (Table 6.2) it was observed that G-Mean values are less than 50 in 42.85% of the cases for the Bcel dataset, 85.71% of the cases for Betwixt dataset, 53.57% of the cases for the Io dataset, 92.85% of the cases for Ivy dataset, 17.85% of the cases for Jcs dataset 7.14% of the cases for Lang dataset, 57.14% of the cases for Log4j dataset and 85.71% of the cases of Ode dataset. Similarly, the performance analysis on Balance results (Table 6.3) it was discovered that Balance values were less than 50 in 50% of the cases for the Bcel dataset, 89.28% of the cases for the Betwixt dataset, 53.57% of the cases for Io dataset, 92.85% of the cases for Ivy dataset, 25% of the cases for Jcs dataset 10.72% of the cases for Lang dataset, 51.14% of the cases for Log4j dataset and 100% of the cases of Ode dataset. The median of G-Mean and Balance of the SMP models developed on imbalanced datasets are reported in Figure 6.1 and Figure 6.2.

As shown in Figure 6.1 median G-Mean values for all datasets except Jcs and Lang are very low for models developed with both SB and ML techniques. Except for Bcel, Jcs, and Lang datasets, for remaining datasets used in this chapter median G-Mean values were low. A similar trend prevails for median Balance values in case of imbalanced datasets. For most of the datasets, the median Balance was just close to 50 for models developed using SB techniques, and neither of SB and ML techniques performed well in terms of Balance measure when the datasets are imbalanced (Figure 6.2). Therefore, it can be seen that the performance of SMP models developed from

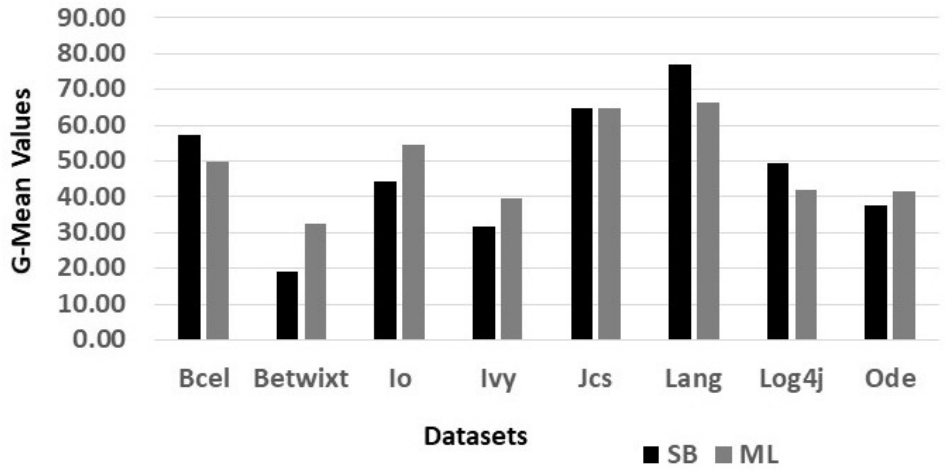


Figure 6.1: Median G-Mean values of SMP Models Developed For Imbalanced Datasets

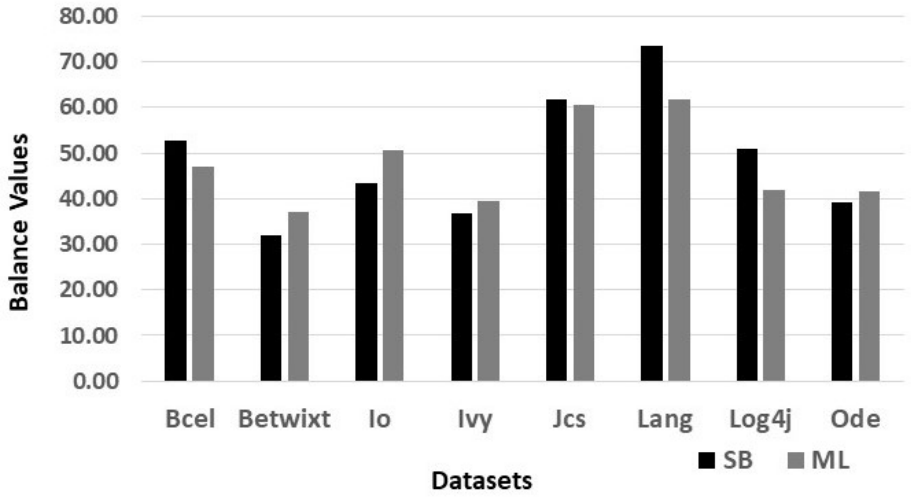


Figure 6.2: Median Balance values of SMP Models Developed For Imbalanced Datasets

the imbalanced dataset is very poor. The reason for the poor performance is the fact that the datasets have very few instances of the low maintainability classes due to which the models have not competently been able to learn the instances of low maintainability classes and resulted in very low true positive rate.

Table 6.1: CFS Results

Dataset	**Selected Features**
Bcel	WMC, CBO
Betwixt	WMC, CBO, RFC, LCOM, SLOC, CAM
IO	NPM, LCOM3
Ivy	WMC, NPM, SLOC, CAM
Jcs	WMC, LCOM3, SLOC, CAM
Lang	NPM LCOM3, SLOC
Log4j	NPM LCOM3, SLOC, DAM, CAM
Ode	WMC, CBO, RFC, LCOM, SLOC, CAM, MOA, Ca, Ce

Table 6.2: G-Mean Results of SMP Models on Imbalanced Datasets

Technique	**Dataset**							
	Bcel	**Betwixt**	**Io**	**Ivy**	**Jcs**	**Lang**	**Log4j**	**Ode**
BIOHEL	61.67	35.74	46.54	40.5	67.11	76.42	58.67	44.25
CHC	52.29	0	31.62	18.87	63.1	79.07	21.61	37.37
CPSO	70.69	52.75	73.11	44.36	57.92	79.31	53.21	39.95
GGA	46.84	0	40.74	18.79	77.22	79.29	45.78	41.66
GA-Int	66.14	0	54.66	26.69	74.91	67.94	58.68	29.58
GA-ADI	46.84	26.78	0	37.41	65.28	75.37	54.94	37.42
IGA	46.69	40.7	43.96	31.76	68.54	60.26	44.67	48.51
LWPSO	67.54	56.37	77.08	31.91	40.69	77.63	44.67	51.85
MPLCS	57.18	33.03	44.53	32.68	69.86	77.9	57.18	32.41
PBIL	57.09	0	31.62	18.84	79.95	79.51	40.54	37.42
SGA	61.67	18.93	31.62	33.09	63.86	79.29	44.48	34.92
SSMA	57.09	18.93	31.49	26.98	52.31	65.27	54.55	29.79
UCS	40.76	19.11	54.66	32.49	52.48	65.45	56.69	41.63
XCS	23.57	0	44.63	26.73	42.65	60.22	27.23	26.47
AdaBoost	57.18	75.11	62.44	49.19	64.07	70.12	58.43	48.57
Bagging	57.28	0	0	37.58	65.68	79.72	50.54	41.63
C4.5	33.17	0	0	32.64	62.5	76.84	30.69	41.73
PART	0	0	31.42	0	26.98	50.25	37.37	0

RIPPER	58.82	60.41	70.65	56.63	78.8	76.63	65.56	51.33
SLIPPER	66.14	36.96	54.43	41.51	71.63	65.45	61.93	48.54
CHI-RW	0	0	0	26.73	0	39.12	22.06	19.38
KNN	52.29	44.12	54.54	60.45	65.61	72.78	32.85	50.05
KSTAR	46.92	0	54.66	26.61	58.3	62.49	46.1	0
LR	33.23	33.18	54.66	46.09	67.88	57.59	34.31	27.47
PUBLIC	23.53	0	0	0	39.85	74.34	0	29.62
BNGE	38.91	41.73	54.19	47.18	64.27	67.01	60	47.18
EACH	60.7	32.79	74.75	32.68	80.71	27.15	48.57	32.11
RISE	57.09	51.08	62.44	47.97	64.96	58.94	26.76	52.02

Table 6.3: Balance Results of SMP Models on Imbalanced Datasets

Technique	**Dataset**							
	Bcel	**Betwixt**	**Io**	**Ivy**	**Jcs**	**Lang**	**Log4j**	**Ode**
BIOHEL	56.76	38.98	44.97	41.63	63.05	73.31	55.7	43.96
CHC	48.92	29.29	36.36	31.81	63.05	76.02	32.64	39.21
CPSO	70.23	51.43	70.68	44.14	54.98	79.23	50.8	41.35
GGA	45	29.28	41.07	31.81	73.54	76.09	44.42	41.69
GA-Int	60.7	29.28	50.5	34.34	70.95	63.22	54.47	35.49
GA-ADI	45	34.49	29.29	39.37	60.61	71.04	51.14	39.21
IGA	44.99	41.9	43.38	36.73	65.31	57.4	44.23	46.57
LWPSO	65.9	54.36	76.24	36.77	41.9	77.42	44.23	49.89
MPLCS	52.84	37.13	43.43	36.87	65.7	73.69	52.84	36.73
PBIL	52.83	29.29	36.36	31.81	76.25	76.15	41.07	39.21
SGA	56.76	31.87	36.36	37.14	60.29	76.09	52.84	37.97
SSMA	52.83	31.87	36.36	34.52	49.95	60.61	51.1	35.6
UCS	41.07	31.9	50.5	36.86	49.99	60.63	52.79	41.69
XCS	33.22	29.29	43.43	34.34	42.37	55.45	34.59	34.25
AdaBoost	52.84	71.4	57.54	46.92	60.35	65.76	54.44	46.58
Bagging	52.85	29.26	29.29	39.39	60.66	76.2	47.78	41.69
C4.5	37.14	29.28	29.29	36.86	58	73.44	36.02	41.69
PART	29.29	29.29	36.35	29.29	34.52	47.59	39.37	29.29
RIPPER	56.1	58.73	69.32	53.97	77.46	74.97	63.28	48.95
SLIPPER	60.7	39.51	50.49	41.86	68.11	60.63	57.75	46.58
CHI-RW	29.29	29.29	29.29	34.34	29.28	40.17	32.74	31.96
KNN	48.92	44.31	50.5	56.78	62.58	68.4	37.36	47.79
KSTAR	45	29.29	50.5	34.34	55.1	57.99	44.44	29.29
LR	37.14	37.14	50.5	44.44	63.5	52.86	37.7	32.25

PUBLIC	33.22	29.29	29.29	29.29	40.98	70.8	29.29	35.49
BNGE	40.72	42.23	50.48	39.35	60.4	63.03	56.05	45.39
EACH	57.86	37.11	74.44	36.87	80.34	34.53	47.34	36.6
RISE	52.83	49.53	57.54	46.26	62.31	55.26	34.55	49.07

6.3.3 Results Specific to RQ2

In the second experiment conducted in this chapter, we developed SMP models by applying data resampling techniques on the imbalanced datasets. We developed the prediction models using ten-fold cross-validation and evaluated the performance with the help of G-Mean and Balance metrics. On each dataset, 28 learning techniques (14 ML techniques and 14 SB techniques) in conjunction with 12 data resampling techniques are used to develop the prediction models for software maintainability. For each dataset, using 14 ML techniques over 12 data resampling techniques, $14 \times 12 = 168$ combinations are explored. As we have used 8 datasets in this chapter, therefore in the case of ML techniques, 8 datasets $\times$ 168 combinations = 1344 models are assessed to formulate our results. To take care of the stochastic nature of SB techniques, we run each of these techniques ten times and reported the median values of G-Mean and Balance performance metrics. For each dataset, using 14 SB techniques over 12 data resampling techniques and 10 runs, $14 \times 12 \times 10 = 1680$ combinations are explored. For 8 datasets in the case of SB techniques, 8 datasets $\times$ 1680 combinations = 16800 models are developed, and median values of G-Mean and Balance performance metrics are reported. G-Mean results of the SMP models developed after data resampling are presented in Table 6.4-6.11 for Bcel, Betwixt, Io, Ivy, Jcs, Lang, Log4j, and Ode datasets. Tables 6.12-6.19 show the Balance performance metric results for all eight datasets used in the chapter. On analyzing Tables 6.4-6.11, it was noted that G-Mean values are higher than 50 in 88.99% of the cases for the Bcel dataset, 67.26% of the cases for Betwixt dataset, 93.15% of the cases for the Io dataset, 77.68% of

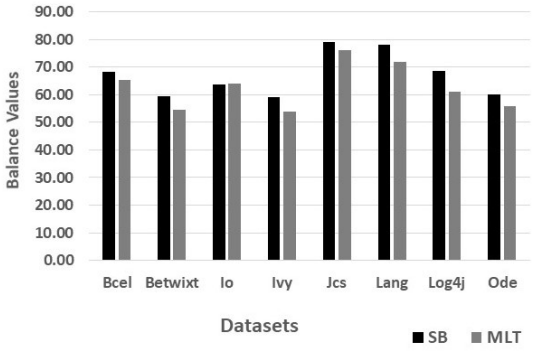


Figure 6.3: Median Balance values of SMP Models Developed with SB and ML Techniques after Data Resampling

the cases for Ivy dataset, 96.13% of the cases for Jcs dataset, 93.75% of the cases for Lang dataset, 92.55% of the cases for Log4j dataset, and 80.65% of the cases for Ode dataset. The analysis of Balance results presented in Tables 6.12-6.19, for SMP models after data resampling revealed that Balance values are greater than 50 in 87.89% of the cases for the Bcel dataset, 67.26% of the cases for Betwixt dataset, 92.86% of the cases for Io dataset, 71.43% of the cases for Ivy dataset, 94.34% of the cases for Jcs dataset, 91.36% of the cases for Lang dataset, 92.69% of the cases for Log4j dataset, and 78.86% of the cases for Ode dataset. Figure 6.3 and 6.4 shows the median of G-Mean and Balance results of SMP models developed using SB and ML techniques after employing data resampling techniques. It is evident from Figures 6.3 and 6.4 that there is a substantial positive enhancement in the performance of prediction models after data resampling as compare to the situation when prediction models were developed from the imbalanced datasets (Figure 6.1 and Figure 6.2). Also, Figures 6.3 and 6.4 indicate that the performance of models developed using SB techniques after data resampling is better than that of ML techniques both with respect to G-Mean and Balance performance measures.

Table 6.4: G-Mean Results of SMP Models after Data Resampling on Bcel Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	73.75	75.66	72.04	78.08	71.16	69.92	69.91	68.9	68.67	69.36	75.79	75.74
CHC	71.01	75.4	70.43	79.42	73.72	69.92	73.92	69.24	72.75	72.02	76.04	74.77
CPSO	65.16	69.28	66.77	68.32	65.96	69.47	65.6	66	64.78	47.16	69.58	67.74
GGA	70.71	76.04	74.22	79.71	76.45	69.81	70.64	69.01	75.91	71.53	75.91	74.49
GA-Int	77.54	75.4	76.64	79.57	77.54	69.92	77.13	69.47	75.79	68.9	75.79	74.68
GA-ADI	79.16	75.27	73.76	79.16	79.16	70.04	75.79	69.92	65.19	42.54	65.82	79.42
IGA	73.92	74.89	76.62	74.24	72.53	65.92	73.53	60.56	66.68	66.56	46.16	74.77
LWPSO	65.15	69.36	68.12	67.93	66.88	65.76	66.28	68.74	75.12	54.35	67.51	60.59
MPLCS	79.16	75.79	77.54	79.16	79.16	70.04	79.16	65.71	68.78	65.28	65.82	79.85
PBIL	72.04	75.27	72.27	79.42	74.24	69.92	73.92	69.24	75.79	66.56	75.91	74.77
SGA	70.86	75.53	71.18	79.42	72.39	69.92	74.35	72.5	72.62	72.75	76.3	76.49
SSMA	66.19	74.63	65.94	81.07	77.41	61.67	48.36	73.47	75.4	72.75	76.3	76.49
UCS	40.76	40.76	40.76	40.76	40.76	40.76	40.76	40.76	40.7	33.28	33.28	33.28
XCS	67.27	72.14	63.15	63.42	55.07	69.92	58.46	57.37	61.77	52.03	69.47	70.59
AdaBoost	73.6	75.53	59.18	77.27	74.37	73.47	80.87	65.5	69.01	56.24	76.04	77.18
Bagging	73.29	75.27	69.11	76.72	75.61	73.47	74.35	65.39	68.32	70.79	76.17	76.04
C4.5	72.82	75.27	68.83	77.13	74.5	65.71	72.53	65.5	71.65	55.12	72.75	79.13
PART	75.27	75.27	74.63	61.47	64.74	0	75.05	0.00	0	0	0	0
RIPPER	0	33.28	0	23.46	0	73.47	52.37	55.48	57.08	53.29	56.38	80.99
SLIPPER	71.45	72.02	75.27	75.53	75.27	73.47	76.86	72.62	73.11	67.27	79.56	76.49
CHI-RW	76.55	57.37	61.87	61.67	61.67	46.99	76.17	33.23	33.23	64	33.23	61.57
KNN	61.87	57.28	40.5	52.29	57.28	61.97	79.95	92.8	65.17	51.32	76.04	40.95
KSTAR	71.16	72.02	55.13	75.75	70.5	69.92	72.91	69.01	68.78	47.44	65.5	75.13
LR	77.82	0	77.95	80.14	80.28	70.04	76.79	69.7	73.23	33.23	69.7	76.04
PUBLIC	75.24	75.27	62.39	79.13	70.22	70.04	70.22	72.5	72.26	57.64	69.47	74.11
BNGE	68.9	68.78	63.27	68.11	66.11	73.59	66.76	62.64	64.75	35.87	68.2	75.44
EACH	60.7	60.7	60.7	60.7	60.7	60.7	60.7	60.7	60.7	60.7	60.7	60.7
RISE	67.13	72.02	55.36	79.13	70.09	69.92	84.19	72.75	72.5	55.69	72.5	57.09
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.5: G-Mean Results of SMP Models after Data Resampling on Betwixt Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	50.08	51.61	40.05	62.85	52.62	56.93	59.63	46.78	47.28	47.25	70.69	62.13
CHC	72.31	55.45	47.64	69.22	69.66	70.67	65.95	69.22	68.77	0	69.95	71.19
CPSO	65.43	67.58	40.58	65.9	64.07	71.03	59.9	61.46	62.17	63.58	65.07	68.4
GGA	63.86	40.28	46.8	67.94	66.04	62.21	65.03	67.22	60.41	32.79	67.92	70.51
GA-Int	65.24	40.49	46.09	67.73	64.49	64.71	64.73	50.82	61.91	51.22	70.69	67.98
GA-ADI	61.46	43.43	45.07	64.28	65.33	63.21	59.9	50.68	54.04	52.01	70.14	66.04
IGA	58	57.84	42.49	64.12	59.3	57.29	61.28	51.58	47.89	19.02	61.46	68.63
LWPSO	65.43	66.4	41.48	63.78	67.05	58.65	51.88	66.78	60.73	69.17	62.88	64.91
MPLCS	63.78	40.17	40.82	65.67	62.46	58.13	58.93	54.32	53.33	40.91	65.98	51.7
PBIL	62.24	55.57	44.32	69.91	64.7	67.46	66.72	61.74	58.48	19.02	63.23	68.63
SGA	64.91	47.91	42.56	67.7	67.94	62.44	65.95	56.37	61.58	42.03	60.91	72.09
SSMA	64.7	51.61	43.19	67.46	68.4	68.97	66.5	59.27	58.32	42.03	60.91	72.09
UCS	32.95	32.95	24.65	32.95	32.95	32.95	33.03	33.03	27.03	32.56	32.56	32.56
XCS	58.58	40.49	35.63	65.72	66.1	68.63	53.18	48.16	47.16	57.29	65.07	54.72
AdaBoost	58	53.9	32.1	59.22	47.5	57.47	63.86	50.14	46.52	26.36	71.7	61.69
Bagging	67.17	36.81	28.34	68.65	61.69	58.73	59.94	43.54	45.36	43.66	73.29	68.41
C4.5	60.52	44.58	31.44	45.34	61.41	55.12	64.71	39.54	44.7	32.64	68.29	66.26
PART	57.65	49.32	0	60.55	26.9	53.82	26.07	0	0	26.65	18.98	59.93
RIPPER	45.88	49.01	32.01	63.64	54.21	55.01	57.02	61.46	58.7	58.13	68.63	60.73
SLIPPER	57.82	54.6	41.51	67.98	62.07	58.32	64.45	50.14	46.4	52.06	70.48	57.91
CHI-RW	70.9	66.86	22.94	60.08	69.31	71.09	69.61	19.63	50.3	0	34	69.81
KNN	58.7	66.85	6.73	43.54	67.34	60.91	62.35	50.68	55.3	54.51	63.4	68.4
KSTAR	56.58	43.19	33.82	47.89	45.75	48.21	48.94	31.44	37.44	25.41	32.64	62.56
LR	71.39	30.64	22.17	71.87	72.1	72.1	68.48	56.21	63.96	41.63	65.25	70.05
PUBLIC	68.79	51.48	28.57	62.56	68.18	62.35	64.41	57.65	62.7	0	41.63	16.28
BNGE	63.21	63.05	27.95	52.9	65.33	59.33	61.47	50.82	54.46	60.24	68.18	63.01
EACH	32.79	32.79	30.8	32.79	32.79	32.79	32.79	32.79	32.79	32.79	32.79	32.79
RISE	55.3	50.41	32.22	63.82	61.3	60.52	61	47.41	53.33	60.33	69.24	60.37
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.6: G-Mean Results of SMP Models after Data Resampling on Io Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	59.69	53.73	51.34	67.36	52.54	61.08	74.13	78.96	58.27	72.76	60.94	80.34
CHC	70.47	68.6	68.95	71.01	71.18	70.83	71.18	54.43	53.26	44.34	61.63	73.56
CPSO	74.25	78.59	78.08	79.71	73.16	80.13	60.25	79.89	84.32	69.03	88.47	84.02
GGA	70.83	61.9	62.85	64.98	72.06	66.89	65.3	54.08	61.08	53.37	54.19	71.13
GA-Int	64.33	61.77	53.96	71.01	72.06	65.63	58.84	62.58	54.19	68.29	61.9	78.22
GA-ADI	63.51	68.91	69.26	76.7	70.29	64.66	71.18	54.31	53.96	66.74	60.81	74.36
IGA	76.12	61.08	70.85	70.83	64.33	69.39	69.57	61.49	53.02	62.31	31.15	75.34
LWPSO	78.51	84.02	77.56	78.75	69.03	76.7	0	40.58	78.43	65.59	79.28	75.31
MPLCS	63.35	61.9	61.62	76.5	65.47	72.06	76.31	54.43	53.84	78.78	61.36	79.07
PBIL	70.65	60.94	68.06	64.98	65.3	71.54	71.54	54.08	53.02	62.31	53.73	75.34
SGA	65.14	69.21	81.69	70.29	71.54	72.24	71.36	54.43	53.26	54.08	61.63	72.36
SSMA	64	68.14	68.42	68.84	63.68	67.92	62.17	54.31	62.17	54.08	61.63	72.36
UCS	44.34	44.44	49.5	44.44	44.44	44.44	54.66	54.54	44.53	62.58	62.58	62.58
XCS	80.96	61.22	59.69	72.24	72.41	72.24	71.89	47.04	54.31	72.06	54.08	75.34
AdaBoost	59.13	61.49	60.94	70.83	60.67	68.75	67.98	53.73	61.22	69.06	61.08	89.1
Bagging	65.3	62.04	59.27	67.74	65.63	66.26	72.41	62.44	53.37	80.62	60.94	77.46
C4.5	71.54	62.31	59.83	66.8	66.89	59.97	67.05	53.96	60.94	68.75	61.08	79.92
PART	80.96	53.61	80.13	60.46	73.76	68.11	66.86	31.49	59.41	23.99	61.22	0
RIPPER	75.98	62.44	62.85	44.53	62.71	75.34	62.04	71.01	70.83	73.16	70.11	83.41
SLIPPER	72.06	61.9	67.05	69.93	72.06	59.83	66.1	62.44	61.77	78.96	61.63	78.86
CHI-RW	78.32	56.39	75.41	75.73	72.82	70.94	73.08	47.04	47.04	46.54	47.04	76.57
KNN	51.57	61.77	54.54	83.14	59.83	59.27	71.71	54.54	53.02	79.71	61.08	71.71
KSTAR	65.94	62.31	50.46	71.13	69.36	68.6	68.45	53.96	52.9	59.69	54.43	69.75
LR	85.21	68.91	85.87	77.65	79.71	81.17	79.49	69.51	74.98	0	69.51	85.87
PUBLIC	75.54	61.08	53.26	65.85	65.14	71.54	72.06	44.53	58.98	0	62.44	85.01
BNGE	52.66	61.63	53.96	62.68	61.08	68.6	74.13	53.96	53.37	73.96	61.08	79.49
EACH	74.75	74.75	74.75	74.75	74.75	74.75	80.16	74.75	74.75	74.75	74.75	74.75
RISE	60.39	94	61.63	72.41	60.39	61.08	70.71	53.84	53.61	78.4	68.14	80.34
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.7: G-Mean Results of SMP Models after Data Resampling on Ivy Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	66.37	53.81	50.86	69.77	58.03	61.22	64.1	55.14	50.16	58.19	60.61	67.84
CHC	73.79	51.63	69.69	68.76	62.97	62.04	65.5	49.49	55.02	32.4	54.68	69.66
CPSO	68.75	69.62	70.36	62.69	68.76	70.5	47.86	45.47	61.68	61.23	45.98	70.1
GGA	63.72	53.8	64.53	67.71	61.21	55.89	50.54	52.03	53.72	37.13	51.47	73.53
GA-Int	70.76	49.64	64.28	68.47	61.73	59.94	66.57	81.04	57.9	44.36	57.9	66.93
GA-ADI	73.08	53.32	64.41	69.48	62.61	62.98	61.55	52.35	51.79	54.76	54.85	71.07
IGA	66.57	60	58.78	65.04	57.94	53.3	58.88	58.62	51.41	37.35	54.42	68.91
LWPSO	71.84	67.69	68.03	60.13	63.68	60	45.82	62.24	66.19	49.43	43.29	67.92
MPLCS	65.45	63.66	50.35	67.32	65.05	63.72	52.1	52.35	51.63	41.2	54.16	71.63
PBIL	68.62	54.19	62.9	66.66	65.89	58.52	62.7	55.27	50.57	37.35	51.39	68.91
SGA	66.57	55.71	57.57	62.39	64.09	63.23	59.52	55.02	51.31	36.39	43.65	69.51
SSMA	68.76	53.41	68.87	59.67	64	61.8	62.04	58.35	60.35	36.39	43.65	69.51
UCS	26.69	26.69	26.63	26.61	26.69	32.59	70.3	32.59	32.59	37.58	37.58	37.58
XCS	71.68	53.99	62.75	69.77	66.37	67.1	71.17	41.82	45.68	48.07	52.67	72.59
AdaBoost	67.76	54.5	51.71	72.41	57	59.1	58.13	48.6	54.07	37.52	53.99	66.93
Bagging	67.07	53.99	62.44	74.98	63.08	63.08	71.52	54.68	57.27	45.41	54.76	36.73
C4.5	66.19	56.54	50.82	70.07	64.46	61.51	25.1	51.63	50.82	81.05	54.5	68.02
PART	62.25	55.98	62.44	69.2	60.2	63.22	55.55	18.73	26.25	0	0	60.92
RIPPER	51.31	36.68	44.24	60.07	63.33	64.36	56.54	66.57	66.72	69.36	60.57	66.19
SLIPPER	74.1	50.9	62.74	70.27	69.15	67.07	64.68	58.26	53.81	23.45	56.08	73.84
CHI-RW	68.42	64.34	60.92	60.3	64	61.63	69.54	26.73	26.73	26.73	26.73	60.61
KNN	65.13	56.12	60.63	60.63	61.09	55.67	63.1	63.03	57.18	61.45	65.02	66.93
KSTAR	62.15	47.07	62.15	67.84	62.87	59.88	65.89	37.19	51.14	44.93	75.63	36.73
LR	67.18	67.32	67.69	68.1	68.75	70.34	70.33	55.78	65.81	58.8	55.61	66.93
PUBLIC	71.95	51.5	40.69	73.99	68.72	60.25	61.86	51.31	32.55	0	37.41	36.73
BNGE	64.46	47.76	41.26	63.51	56.72	53.29	56.12	52.11	54.25	59.57	56.72	36.73
EACH	32.68	32.68	32.68	32.68	32.68	32.68	32.68	27.49	32.68	27.18	32.68	36.73
RISE	56.72	50.98	45.27	67.69	52.84	52.67	56.12	48.67	56.91	52.92	60.4	36.73
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.8: G-Mean Results of SMP Models after Data Resampling on Jcs Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	77.09	75.77	78.92	77.26	75.52	78.27	78.94	79.87	81.11	65.48	82.01	75.03
CHC	86.47	82.57	78.64	81.94	83.76	83.76	85.86	79.49	85.83	71.4	86.41	82.21
CPSO	79.21	66.96	80.97	73.56	74.1	78.01	71.93	70.62	73.3	58.75	81.02	73.56
GGA	84.95	75.03	81.54	81.62	84.95	81.03	82.85	79.49	84.65	85.29	43.69	84.35
GA-Int	84.22	72.79	85.71	79.54	78.1	83.16	81.33	74.29	81.45	68.34	81.73	80.89
GA-ADI	83.65	75.15	75.73	81.62	78.66	83.37	83.16	74.04	83.37	59.5	84.95	77.71
IGA	77.54	80.1	73.58	74.68	74.75	81.64	81.33	76.03	78.88	88.19	79.21	80.32
LWPSO	80.52	57.92	78.77	73.92	74.68	75.2	76.34	72.93	78.63	60.91	82.76	78.94
MPLCS	83.16	76.44	83.88	79.49	75.15	80.32	81.92	79.87	79.76	66.04	34.91	78.59
PBIL	82.55	79.84	82.83	82.21	83.16	83.76	84.64	77.74	85.83	88.19	85.24	80.32
SGA	83.76	77.71	89.09	80.03	82.5	81.62	83.76	79.76	83.65	81.38	81.45	100
SSMA	83.39	82.45	76.82	80.32	85.56	82.85	81.64	78.72	83.46	81.38	81.45	100
UCS	67.53	67.53	66.35	67.53	67.53	67.53	67.53	67.79	70.71	53.3	53.3	53.3
XCS	85.53	78.39	65.61	85.56	81	82.5	82.85	79.58	81	34.19	84.79	79.31
AdaBoost	80.03	75.28	67.87	78.66	79.06	81.45	77.74	75.41	77.47	70.11	80.03	73.92
Bagging	79.21	78.8	76.95	78.88	78.94	80.03	76.67	69.07	79.76	65.18	75.93	73.92
C4.5	80.89	70.48	68.09	78.88	80.03	79.21	79.49	76.94	76.4	73.03	70.85	75.26
PART	66.5	33.14	32.44	75.77	37.8	27.07	48.46	0	19.13	73.3	0	71.09
RIPPER	71.17	73.98	71.63	73.03	72.07	79.46	78.27	78.66	79.21	59.34	84.35	71.09
SLIPPER	73.28	75.03	72.55	78.1	79.06	78.01	77.74	79.76	79.49	68.88	84.95	74.68
CHI-RW	82.71	84.34	83.37	83.69	82.71	82.39	82.39	82.11	85.35	77.69	84.02	80.05
KNN	66.73	70.25	65.61	65.61	70.85	80.89	79.46	74.04	71.09	55.31	84.35	71.09
KSTAR	69.11	66.96	73.92	78.59	72.52	70.11	69.61	77.74	82.21	64.13	78.54	73.92
LR	82.25	85.24	83.46	83.16	83.46	84.64	84.02	81.92	84.22	43.08	81.38	75.26
PUBLIC	78.01	75.67	72.79	74.68	76.41	74.48	73.56	80.44	76.98	38.49	84.06	81.17
BNGE	76.44	76.95	75.28	76.4	77.74	80.89	80.03	78.54	80.03	69.66	83.94	73.92
EACH	80.71	80.71	80.71	80.71	80.71	80.71	80.71	80.71	80.71	80.71	80.71	71.09
RISE	66.5	66.96	70.71	70.01	66.96	77.26	76.41	74.36	78.66	61.57	82.25	73.92
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.9: G-Mean Results of SMP Models after Data Resampling on Lang Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	77.09	79.29	74.76	79.03	58.94	76.84	79.42	75.13	80.78	68.29	81.25	76.63
CHC	79.09	83.58	81.01	83.5	82.89	80.79	81.72	81.46	83.35	79.51	82.89	79.82
CPSO	76.05	82.28	82.18	79.58	79.82	83.21	74.65	79.58	78.78	67.75	81.91	81.15
GGA	77.38	81.68	79.38	80.06	62.22	80.79	79.88	81.46	83.58	79.72	81.24	82.53
GA-Int	80.3	81.24	79.65	82.89	78.41	84.93	82.42	78.85	82.89	70.91	78.5	82.77
GA-ADI	81.79	79.29	82.54	82.65	80.79	82.19	84.7	80.57	82.42	73.17	82.19	83.01
IGA	72.7	77.3	72.45	75.77	78.27	78.12	68.92	73.51	70.19	77.48	70.77	78.85
LWPSO	74.42	76.88	78.77	77.49	79.8	80.01	76.18	75.75	74.65	66.44	72.17	73.98
MPLCS	78.36	81.24	77.3	83.74	80.33	83.12	80.33	83.12	83.12	68.3	79.42	84.97
PBIL	79.29	83.58	71.61	80.11	69.4	81.24	81.72	83.12	79.07	77.48	82.65	78.85
SGA	77.87	81.24	79.38	81.25	79.07	82.65	77.08	81.68	82.89	77.27	49.85	78.61
SSMA	79.03	81.46	82.42	83.74	77.53	82.42	81.48	84.93	83.26	77.27	82.89	78.61
UCS	65.27	65.27	69.42	65.27	62.66	65.27	62.66	65.27	65.45	67.94	67.94	67.94
XCS	79.82	81.46	79.59	82.42	80.33	80.79	82.19	78.63	82.65	65.56	79.88	80.63
AdaBoost	72.7	74.75	63.48	82.53	78.63	75.99	77.08	75.56	75.13	75.42	77.75	67.94
Bagging	75.04	79.07	75.13	83.26	81.01	82.89	80.78	66.61	79.42	68.99	79.88	75.04
C4.5	76.62	83.35	72.03	83.26	84.46	79.88	73.82	79.42	78.5	71.34	79.65	75.04
PART	27.49	83.35	53.44	65.62	80.79	42.47	48.37	32.81	27	57.03	0	44.76
RIPPER	69.73	59.42	67.75	61.98	65.09	79.29	76.2	77.56	79.09	89.22	78.61	78.19
SLIPPER	75.49	76.84	73.3	79.88	80.11	76.2	73.81	77.75	77.08	66.26	76.18	79.54
CHI-RW	77.9	77.58	81.17	76.8	82.31	81.4	75.78	51.75	51.75	69.16	51.75	85.09
KNN	72.47	76.63	72.78	72.78	75.99	77.08	76.18	74.55	67.98	64.11	79.19	67.94
KSTAR	66.38	69.54	65.97	80.54	71.58	77.05	80.56	76.63	71.45	68.79	79.07	75.04
LR	85.85	86.69	85.46	86.37	85.21	84.22	81.93	81.46	80.55	81.48	82.89	67.94
PUBLIC	78.12	83.58	76.18	85.64	82.65	82.89	82.19	73.93	72.49	30.19	79.07	75.95
BNGE	65.12	72.18	64.2	77.09	71.38	76.2	78.27	32.85	70.69	64.62	76.63	67.94
EACH	27.15	27.15	27.15	27.15	27.15	27.15	27.15	27.15	23.62	27.15	27.15	75.04
RISE	62.2	64.56	57.95	76.13	74.14	78.63	81.01	70.36	69.73	61.17	76.4	67.94
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.10: G-Mean Results of SMP Models after Data Resampling on Log4j Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	63.36	57.32	63.86	67.33	66.49	65.87	64.8	67.74	70.25	63.9	79.77	71.42
CHC	74.23	76.7	79.77	75.86	77.88	75.28	75.74	62.61	67.53	60.08	72.1	73.69
CPSO	62.99	74.01	55.77	73.56	68.84	71.17	66.43	70.14	62.87	65.35	67.27	75.42
GGA	69.23	71.87	67.08	74.36	70.45	72.22	77.01	62.42	71.53	47.29	74.45	36.68
GA-Int	70.58	71.98	71.28	74.91	72.47	73.13	77.07	62.98	62.85	91.05	72.24	70.86
GA-ADI	68.16	67.61	69.09	73.85	72.16	75.54	74.76	65.8	63.77	62.63	70.97	69.65
IGA	68.89	73.23	70.73	69.51	71.42	67.88	69.02	66.82	69.29	55.63	63.25	76.45
LWPSO	68.39	71.45	66.1	74.36	69.4	73.76	67.49	60.53	68.97	67.33	67.76	67.71
MPLCS	73.37	71.87	66.92	71.29	70.95	71.05	69.96	60.75	67.74	56.85	67.21	67.08
PBIL	68.73	72.65	73.47	75.32	74.23	71.54	75.56	65.7	68.06	55.63	68.77	76.45
SGA	65.67	67.94	72.34	73.11	70.29	72.22	75	66.39	72.66	55.88	68.94	74.3
SSMA	69.91	70.37	69.18	74.23	71.17	77.05	74.09	69.42	65.66	55.88	68.94	74.3
UCS	54.79	54.79	54.43	54.08	54.79	58.52	54.79	54.94	54.94	56.61	56.61	56.61
XCS	72.91	65.7	65.85	73.36	69.63	68.61	72.6	57.13	66.91	62.85	69.74	71.01
AdaBoost	64.8	61.12	60.2	64.2	61.47	61.47	64.6	63.35	65.45	61.93	67.94	56.61
Bagging	69.47	64.32	64.28	69.68	63.15	67.72	72.03	56.89	59.36	61.39	69.01	56.61
C4.5	70.19	60.66	63.67	71.4	64.88	61.65	65.69	63.84	66.99	58.09	61.47	67.95
PART	59.29	55.17	66.41	60.17	63.64	59.95	53.97	21.73	37.48	56.37	21.79	50.53
RIPPER	64.82	57.92	60.57	71.3	59.92	70.87	63.67	65.69	65.34	52.17	71.93	62.31
SLIPPER	63.64	63.93	64.69	67.33	64.92	67.1	67.61	55.4	65.5	63.42	63.98	63.96
CHI-RW	70.94	77.8	76.32	74.51	75.78	76.8	72.58	30.76	57.23	29.71	51.89	70.89
KNN	69.15	66.48	44	44	70.03	74.23	73.13	17.02	81.57	49.3	60.71	62.31
KSTAR	48.72	61.93	72.28	73.98	54.16	54.39	63.74	56.61	61.21	69.26	54.55	70.89
LR	72.66	74.23	74	72.48	74.29	75.74	74.52	59.47	72.65	47.5	73	62.31
PUBLIC	67.1	70.95	71.05	67.19	66.41	69.1	73.27	55.13	54.88	0	61.57	86.57
BNGE	61.47	62.23	64.42	67.61	62.36	66.23	64.35	61.21	65.5	63.56	69.66	70.89
EACH	48.57	48.57	44.3	48.57	48.57	48.57	48.57	48.57	48.57	48.57	48.57	74.3
RISE	56.95	60.57	57.68	67.27	60.5	65.13	64.35	68.38	64.71	60.43	69.96	74.3
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.11: G-Mean Results of SMP Models after Data Resampling on Ode Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	63.78	54.62	58.32	65.68	62.4	67.39	65.35	52.28	61.69	38.57	55.33	64.91
CHC	69.21	58.03	71.17	71.05	64.25	68.6	69.77	55.65	61.74	48.93	61.99	71.18
CPSO	51.59	62.78	62.39	58.03	55.1	56.15	73.16	57.91	63.23	62.42	55.91	57.26
GGA	59.7	59.38	63.27	68.11	66.07	65.52	65.26	60.96	55.9	50.71	57.61	71.75
GA-Int	70.24	57.95	67.82	71.1	70.19	69.89	66.18	43.48	58.58	46.71	49.92	72.9
GA-ADI	69.58	64.25	66.08	71.53	71.15	72.55	69.7	47.12	60.87	44.37	51.66	73.94
IGA	57.13	58.55	55.5	66.08	65.92	62.47	66.51	53.28	58.28	49.08	59.53	70.57
LWPSO	56.29	57.22	57.27	61.3	56.89	60.07	52.59	64.71	55.52	54.64	61.42	54.62
MPLCS	68.28	53.74	65.73	69.05	71	67.2	70.51	50.24	56.61	46.74	56.33	70.96
PBIL	60.84	63.13	62.13	66.94	62	66.94	68.22	58.45	58.47	49.08	59.84	70.57
SGA	58.22	55.6	59.78	68.21	64.12	62.63	69	59.85	54.31	48.99	54.45	71.49
SSMA	60.74	60.27	56.47	69.26	66.33	64.02	65.26	58.27	59.3	48.99	54.45	71.49
UCS	41.43	39.45	43.57	41.43	41.48	34.79	41.48	45.38	43.61	37.19	37.19	37.19
XCS	67.29	52.8	58.89	69.9	66.7	66.99	69.81	43.61	46.48	58.87	44.99	67.09
AdaBoost	58.97	52.98	51.07	67.5	61.91	66.47	66.89	59.26	55.9	52.67	60.55	71.49
Bagging	62.3	53.18	55.11	69.41	63.71	68.64	67.68	54.76	54.27	48.31	57.21	67.09
C4.5	65.02	51.3	42.18	69.46	65.94	70.84	67.15	57.5	59.38	44.65	57.83	67.82
PART	45.44	35.62	56.69	56.58	54.41	30.09	46.09	18.71	22.68	0	0	53.44
RIPPER	57.56	78.61	55.71	72.83	67.77	71.88	67.14	64.79	63.53	52.62	70.12	63.45
SLIPPER	62.13	55.65	53.64	64.01	63.97	65.97	68.63	57.5	52.88	54.7	59.34	67.06
CHI-RW	68.14	85.56	75.52	70.49	71.78	70.06	69.15	30.51	30.51	30.42	37.82	69.98
KNN	55.79	61.89	50.05	49.16	62.59	62	62.77	50.02	53.96	44.65	54.41	67.06
KSTAR	50.84	50.74	62.17	65.19	60.59	62.25	65.35	0	18.66	32.13	0	63.45
LR	71.32	68.92	72.66	71.43	73	72.78	73.1	47.03	57.24	41.05	47	67.06
PUBLIC	72.31	54.1	51.33	66.73	68.89	72.21	67.21	58.75	60.96	0	52.74	65.66
BNGE	57.56	69.06	61.15	64.17	72.46	71.49	69.16	50.4	56.72	46	57.06	67.09
EACH	62.25	32.39	32.39	32.39	32.39	32.39	32.39	32.39	32.39	32.39	32.39	63.45
RISE	38.74	63.11	54.31	64.08	65.61	66.94	66.99	48.6	51.63	53.11	60.07	63.45
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.12: Balance Results of SMP Models after Data Resampling on Bcel Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	73.56	72.14	71.69	75.67	69.78	64.61	67.44	64.46	64.42	69.29	72.17	75.56
CHC	70.8	72.06	69.97	78.41	71.41	64.61	73.2	64.53	68.39	68.22	72.24	73.81
CPSO	63.37	65.09	63.76	64.65	63.56	68.33	59.72	64	64.3	47.18	63.53	61.74
GGA	70.54	72.24	72.44	78.6	74.87	64.61	69.59	64.49	72.21	68.08	72.21	73.61
GA-Int	75.43	72.06	74.92	78.51	75.43	64.61	75.23	64.56	72.17	64.46	72.17	74.6
GA-ADI	76.05	72.02	71.77	76.05	76.05	64.62	72.17	64.61	62.19	42.49	60.68	78.41
IGA	73.2	71.89	74.91	71.64	70.79	60.69	72.51	56.6	63.79	63.74	44.93	73.81
LWPSO	63.4	65.09	65.58	64.2	64	65.76	60.39	65.96	74.31	54.1	61.51	55.25
MPLCS	76.05	72.17	74.13	76.05	76.05	64.62	76.05	60.66	64.44	60.61	60.68	78.68
PBIL	71.7	72.02	71.37	78.41	71.64	64.61	73.2	64.53	72.17	63.74	72.21	73.81
SGA	70.67	72.1	69.95	78.41	70.72	64.61	73.51	68.34	68.37	68.39	72.3	76.21
SSMA	65.35	71.8	65.55	80.84	75.37	56.76	47.72	68.51	72.06	68.39	72.3	76.21
UCS	41.07	41.07	41.07	41.07	41.07	41.07	41.07	41.07	41.07	37.15	37.15	37.15
XCS	67.09	69.65	61.75	60.5	53.5	64.61	53.46	52.85	56.77	48.9	64.56	70.56
AdaBoost	73.57	72.1	57.95	75.3	71.69	68.51	80.76	60.64	64.49	54.82	72.24	77.17
Bagging	73.27	72.02	68.46	75.02	74.37	68.51	73.51	60.62	64.33	67.82	72.27	75.82
C4.5	72.81	72.02	68.24	75.23	71.75	60.66	70.79	60.64	68.12	52.44	68.39	78.23
PART	72.02	72.02	71.8	56.74	60.51	29.29	74	29.29	29.29	29.29	29.29	29.29
RIPPER	0	37.15	0	33.21	0	68.51	48.92	52.18	53.51	50.39	53.9	79.31
SLIPPER	70.14	68.22	72.02	72.1	72.02	68.51	75.09	68.37	68.46	64.01	76.16	76.21
CHI-RW	72.35	52.85	56.77	56.76	56.76	45	72.27	37.14	37.14	29.29	37.14	56.75
KNN	56.77	52.85	41.06	48.92	52.85	56.78	76.25	91.09	60.59	63.5	72.24	41.15
KSTAR	70.94	68.22	54.11	74.46	69.5	64.61	72.42	64.49	64.44	47.2	60.64	75.02
LR	77.28	68.44	75.61	78.85	78.94	64.62	76.46	64.59	68.48	37.14	64.59	75.82
PUBLIC	75.16	72.02	61.33	78.23	69.3	64.62	69.3	68.34	68.29	55.58	64.56	71.58
BNGE	68.85	64.44	61.94	67.98	65.92	68.52	65.06	59.85	62.85	38.91	64.3	75.29
EACH	57.86	57.86	57.86	57.86	57.86	57.86	57.86	57.86	57.86	57.86	57.86	57.86
RISE	66.82	68.22	54.26	78.23	69.2	64.61	83.67	68.39	68.34	52.6	68.34	69.09
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.13: Balance Results of SMP Models after Data Resampling on Betwixt Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	49.71	49.73	48.85	62.51	52.04	56.3	58.94	46.47	46.73	47.39	53.22	64.91
CHC	72.27	53.89	69.85	69.02	69.28	70.22	65.53	67.1	67.85	29.28	57.73	71.17
CPSO	65.42	66.94	59.46	65.89	63.91	70.06	58.32	60.29	60.77	61.62	54.07	54.28
GGA	63.72	41.72	56.88	67.89	65.97	62.21	64.85	64.76	58.73	37.11	53.94	71.73
GA-Int	64.49	41.82	60.35	67.43	64.48	64.69	64.7	49.42	59.53	49.58	47.76	72.89
GA-ADI	60.29	43.97	65.08	64.22	65.05	63.12	59.9	49.36	52.05	49.86	49.02	73.92
IGA	57.98	56.23	62.16	63.59	59.24	57.19	61.26	50.75	47.95	31.89	56.2	70.55
LWPSO	65.42	65.35	61.31	63.73	67.05	58.62	51.69	65.62	59.78	68.13	59.64	51.41
MPLCS	62.7	41.67	46.16	65.66	61.71	57.71	58.71	52.17	51.73	41.98	52.74	70.95
PBIL	61.97	52.57	66.61	69.39	64.49	67.4	66.35	59.45	56.59	31.89	56.28	70.55
SGA	64.68	47	55.88	67.65	67.89	62.44	65.53	54.36	59.37	42.29	51.43	71.47
SSMA	64.49	49.73	56.26	67.4	68.38	68.76	66.4	56.98	56.5	42.29	51.43	71.47
UCS	37.13	37.13	34.52	37.13	37.13	37.13	37.13	37.13	34.52	37.06	39.21	39.21
XCS	57.45	41.82	45.52	65.72	66.09	68.63	53.12	47.09	46.67	55.26	44.11	66.97
AdaBoost	57.12	51.99	43.64	59.22	47.46	56.72	63.37	49.1	46.34	34.31	57.33	71.47
Bagging	66.62	39.58	56.32	68.63	61.1	58.53	59.86	44.03	45.63	44.09	53.85	66.97
C4.5	60.12	44.5	47.96	45.48	61.23	54.79	64.69	41.34	45.17	37.08	53.98	67.81
PART	56.86	48.66	29.29	59.64	34.51	53.38	34.27	29.29	29.29	34.46	29.29	52.48
RIPPER	45.96	47.37	43.71	63.52	53.15	54.89	55.73	60.29	58.18	58.06	68.23	62.02
SLIPPER	56.99	52.27	36.32	67.75	61.41	57.87	63.86	49.1	46.26	51.87	56.14	67
CHI-RW	70.89	65.77	68.35	59.7	69.29	71.08	69.27	32.07	48.63	29.26	39.56	69.53
KNN	57.62	64.94	44.31	44.03	67.18	60.46	62.29	49.36	53.8	53.93	52.79	67
KSTAR	56.01	43.84	45.03	47.97	46.08	48.23	48.95	36.67	39.87	33.9	29.29	62.02
LR	71.32	36	70.28	71.82	72.06	72.06	68.23	54.28	62.83	42.21	45.37	67
PUBLIC	68.4	49.68	58.53	59.66	68.14	60.77	62.91	56.86	61.11	29.29	50.15	64.92
BNGE	62.29	61.32	56.75	51.51	65.05	59.08	61.44	49.42	52.22	58.63	53.82	66.97
EACH	37.11	37.11	37.11	37.11	37.11	37.11	37.11	37.11	37.11	37.11	36.73	62.02
RISE	53.8	49.23	39.26	63.81	60.78	60.12	60.85	46.78	51.73	59.95	57.16	62.02
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.14: Balance Results of SMP Models after Data Resampling on Io Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	56.88	50.43	50.01	66.88	50.18	57.31	70.76	77.42	56.25	70.52	57.27	80.33
CHC	69.2	64.4	66.79	69.55	69.65	69.43	75.54	50.49	50.35	43.42	57.42	73.4
CPSO	73.86	75.81	77.145	79.7	73.04	80.13	57.22	77.89	83.55	69.02	88.39	79.2
GGA	69.43	57.47	60.025	62.98	70.16	63.87	63.15	50.47	57.31	50.37	50.48	71.11
GA-Int	62.61	57.45	53.135	69.55	70.16	63.31	69.86	57.55	50.48	64.33	57.47	78.16
GA-ADI	62.1	64.47	66.825	75.97	69.09	62.79	69.65	50.49	50.46	63.81	57.24	74.1
IGA	75.54	57.31	67.6	69.43	62.61	68.46	68.59	57.4	50.3	57.52	36.33	74.93
LWPSO	76.62	79.2	76.62	76.91	69.02	76.54	58.69	42.07	78.39	64.39	79.27	72.92
MPLCS	61.99	57.47	59.605	75.83	63.23	70.16	75.68	50.49	50.45	77.32	57.37	79.06
PBIL	69.32	57.27	64.25	62.98	63.15	69.86	70.06	50.47	50.3	57.52	50.43	74.93
SGA	63.06	64.52	79.105	69.09	69.86	70.25	69.76	50.49	50.35	50.47	57.42	72.29
SSMA	62.41	64.29	64.43	68.05	62.2	67.33	61.17	50.49	57.51	50.47	57.42	72.29
UCS	43.42	43.42	46.96	43.42	43.42	43.42	50.49	50.5	43.43	57.55	57.55	57.55
XCS	80.94	57.34	56.875	70.25	70.34	70.25	70.06	45	50.49	70.16	50.47	74.93
AdaBoost	56.65	57.4	57.27	69.43	57.2	64.43	64.24	50.43	57.34	64.49	57.31	86.01
Bagging	63.15	57.49	56.71	67.19	63.31	63.61	70.34	57.54	50.37	78.19	57.27	76.5
C4.5	69.86	57.52	56.93	66.41	63.87	56.98	63.93	50.46	57.27	64.43	57.31	79.92
PART	80.94	50.41	80.13	59.86	73.58	67.48	66.79	36.36	56.77	29.64	57.34	29.29
RIPPER	71.59	57.54	57.56	43.43	57.56	74.93	57.49	69.55	69.43	73.04	68.96	83.12
SLIPPER	70.16	57.47	63.93	68.84	70.16	56.93	63.54	57.54	57.45	77.42	57.42	78.84
CHI-RW	72.67	52.75	75.33	75.24	70.42	70.51	70.7	45	45	44.97	45	74.68
KNN	49.86	57.45	50.5	78.77	56.93	56.71	69.97	50.5	50.3	77.8	57.31	71.06
KSTAR	63.46	57.52	49.36	71.11	64.54	64.4	64.36	50.46	50.27	56.88	50.49	68.72
LR	84.61	64.47	85.4	76.63	79.7	81.14	79.49	64.56	71.37	29.2	64.56	85.4
PUBLIC	75.08	57.31	50.35	65.58	63.06	69.86	70.16	43.43	56.59	29.29	57.54	80.39
BNGE	50.21	57.42	50.46	61.53	57.31	64.4	71.1	50.46	50.37	71.04	57.31	79.49
EACH	74.44	74.44	74.44	74.44	74.44	74.44	78.01	74.44	74.44	74.44	74.44	74.44
RISE	57.12	93.49	57.42	70.34	57.12	57.31	64.64	50.45	50.41	77.1	64.29	80.33
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.15: Balance Results of SMP Models after Data Resampling on Ivy Dataset

Technique	Adasyn	BSMOTE	ROS	SSMOTE	SMOTE	SMENN	SMTL	SP	SPII	CNN	NCL	RUS
BIOHEL	64.84	51.51	49.08	68.88	55.9	58.63	62.46	53.34	48.79	47.39	53.22	67.82
CHC	73.77	51.13	69.49	68.75	62.94	61.9	65.27	47.19	51.84	29.28	57.73	69.63
CPSO	68.69	69.59	70.11	62.66	68.75	69.93	46.89	44.39	60.81	61.62	54.07	70.09
GGA	62.98	52.61	63.68	67.27	60.45	55.28	50.45	49.36	51.48	37.11	53.94	73.24
GA-Int	70.59	49.26	63.72	68.47	61.34	59.74	66.56	77.16	54.34	49.58	47.76	66.87
GA-ADI	72.23	52.31	63.51	69.45	62.09	62.39	61.54	49.41	49.31	49.86	49.02	70.92
IGA	65.79	58.93	57.36	64.91	57.36	52.81	58.77	56.16	50.4	31.89	56.2	68.85
LWPSO	71.73	67.58	68	58.45	62.57	56.41	45.66	59.03	65.5	68.13	59.64	67.55
MPLCS	65.28	62.19	49.77	66.93	64.57	62.98	51.83	49.41	49.27	41.98	52.74	71.17
PBIL	68.01	52.84	61.27	66.36	64.54	58.04	62.52	51.89	48.95	31.89	56.28	68.85
SGA	65.79	54.58	55.775	61.33	63.26	63	59.07	51.84	49.19	42.29	51.43	69.47
SSMA	68.58	52.37	66.235	58.68	62.41	60.9	61.9	54.42	56.76	42.29	51.43	69.47
UCS	34.34	34.34	34.335	34.34	34.34	36.86	64.64	36.86	36.86	37.06	39.21	39.39
XCS	71.57	52.72	60.63	68.88	64.84	65.64	71.05	41.9	44.41	55.26	44.11	72.53
AdaBoost	65.65	51.72	49.29	72.16	54.1	56.36	55.94	46.82	51.59	34.31	57.33	66.87
Bagging	65.27	51.57	59.09	74.81	60.8	60.8	70.07	51.76	54.18	44.09	53.85	39.57
C4.5	65.5	53.94	49.04	69.72	62.68	59.33	33.49	49.27	49.04	37.08	53.98	67.22
PART	62.02	55.88	62.42	68.92	59.98	63.1	52.52	31.8	34.29	34.46	29.29	60.87
RIPPER	49.19	39.25	44.8	56.68	59.34	64.27	53.94	63.74	65.06	58.06	68.23	65.5
SLIPPER	72.72	49.06	59.18	69.25	67.57	65.27	62.81	54.41	51.51	51.87	56.14	73.58
CHI-RW	67.11	62.61	58.5	58.2	62.41	58.8	69.26	34.34	34.34	29.26	39.56	58.35
KNN	63.06	54.84	56.83	56.83	60.36	55.1	62.88	59.27	54.15	53.93	52.79	66.87
KSTAR	61.16	46.36	61.16	67.37	60.69	57.98	64.54	39.35	49.14	33.9	29.29	39.57
LR	66.82	66.35	66.61	67.59	68.11	69.95	70.27	51.96	61.82	42.21	45.37	66.87
PUBLIC	71.52	50.45	41.69	73.75	68.54	59.68	61.45	49.19	37.18	29.29	50.15	39.57
BNGE	61.4	46.61	41.82	61	54.01	51.31	54.84	49.37	51.65	58.63	53.82	39.57
EACH	36.87	36.87	36.87	36.87	36.87	36.87	36.87	33.52	36.87	37.11	36.73	39.57
RISE	54.01	49.09	44.36	66.61	51.13	51.05	54.84	46.84	54.07	59.95	57.16	39.57
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.16: Balance Results of SMP Models after Data Resampling on Jcs Dataset

Technique	**Adasyn**	**BSMOTE**	**ROS**	**SSMOTE**	**SMOTE**	**SMENN**	**SMTL**	**SP**	**SPII**	**CNN**	**NCL**	**RUS**
BIOHEL	76.34	70.69	68.205	77.1	72.98	77.14	78.5	72.87	80.04	65.19	81.6	74.72
CHC	86.29	82.02	85.53	81.91	83.71	83.71	85.6	84.02	85.81	68.04	86.36	82.2
CPSO	78.83	64.64	71.44	72.93	74.1	78.01	68.83	70.62	72.72	56.72	80.69	72.93
GGA	84.54	75	87.07	81.62	84.54	81.03	82.73	79.32	84.64	83.63	44.1	84.34
GA-Int	84.02	70.13	82.91	79.47	77.82	83.06	81.33	74.56	81.16	67.53	81.38	80.69
GA-ADI	83.53	75.27	82.21	81.62	78.28	83.28	83.06	73.71	83.28	58.83	84.94	77.71
IGA	76.71	78.14	68.98	74.68	74.49	81.38	81.04	74.24	78.86	84.29	78.72	80.19
LWPSO	79.32	59.36	75.72	73.76	74.68	74.72	72.01	72.93	78.5	59.91	81.98	78.83
MPLCS	83.06	72.52	80.42	78.92	74.07	80.19	81.91	79.87	79.13	64.15	81.16	78.58
PBIL	82.4	78.02	85.6	82.06	83.06	83.71	84.18	81.6	85.81	84.29	85.24	80.19
SGA	83.71	77.61	85.92	79.93	82.48	81.62	83.52	81.6	83.53	78.64	81.16	65.24
SSMA	82.72	74.86	85	80.19	85.25	82.73	81.38	78.72	83.39	78.64	81.16	65.24
UCS	63.14	63.14	65.62	63.14	63.14	63.14	63.14	63.14	67.79	50.16	50.16	50.16
XCS	85.53	76.85	62.49	85.25	80.65	82.48	82.73	79.13	80.65	37.03	84.49	79.2
AdaBoost	79.32	72.87	65.05	78.28	77.61	81.16	76.79	74.24	76.61	68.77	79.32	73.76
Bagging	78.72	77.46	75.14	78.86	78.5	79.93	76.03	67.03	79.13	62.4	74.56	73.76
C4.5	80.69	67.69	65.14	78.86	79.32	78.72	78.92	76.23	75.82	72.51	69.23	75.24
PART	64.4	37.14	37.03	73.08	39.71	34.64	48	29.29	31.9	72.72	29.29	69.37
RIPPER	67.96	70.69	68.11	71.67	69.9	79.41	77.14	78.28	78.72	59.34	84.34	69.37
SLIPPER	71.82	72.76	70.13	77.82	77.61	76.97	76.79	79.13	78.92	68.84	84.94	74.68
CHI-RW	80.91	82.89	81.7	82.1	80.91	80.51	80.51	81.49	85.09	76.1	82.67	77.7
KNN	64.52	67.59	62.58	62.58	69.23	80.69	79.41	72.26	69.37	55.17	84.34	69.37
KSTAR	68.09	64.64	73.76	78.58	71.34	68.77	68.44	76.79	82.2	63.98	77.3	73.76
LR	82.06	85.24	83.39	83.06	83.39	84.18	83.46	80.5	84.02	43.78	80.2	75.24
PUBLIC	78.01	74.4	70.23	74.68	76.33	74.25	72.93	80.42	76.85	40.67	84.02	80.08
BNGE	74.86	75.14	72.87	75.82	76.79	80.69	79.93	77.3	79.32	69.19	83.78	73.76
EACH	80.34	80.34	80.34	80.34	80.34	80.34	80.34	80.34	80.34	80.34	80.34	69.37
RISE	64.4	64.64	67.79	69.9	64.64	77.1	76.33	73.52	78.28	61.37	82.06	73.76
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.17: Balance Results of SMP Models after Data Resampling on Lang Dataset

Technique	**Adasyn**	**BSMOTE**	**ROS**	**SSMOTE**	**SMOTE**	**SMENN**	**SMTL**	**SP**	**SPII**	**CNN**	**NCL**	**RUS**
BIOHEL	76.34	76.09	72.62	79.01	73.31	73.44	77.8	72.81	79.82	68.26	80.12	76.53
CHC	78.63	81.23	78.51	82.66	80.96	78.43	80.39	78.66	81.15	76.15	80.96	79.17
CPSO	75.66	81.81	82.155	78.99	82.67	83.14	72.48	79.51	78.76	65.75	81.69	81.15
GGA	77.2	78.73	77.025	79.34	80.75	78.43	78.04	78.66	81.23	76.2	78.59	81.99
GA-Int	79.51	78.59	77.135	80.96	81.06	83.47	80.75	75.95	80.96	70.81	77.28	82.17
GA-ADI	81.43	76.09	80.8	81.43	78.51	80.64	83.35	78.24	80.75	72.63	80.64	82.34
IGA	72.25	75.32	71.895	75.35	77.13	77.84	66.94	71	69.62	73.6	67.81	78.43
LWPSO	69.84	76.58	78.61	77.49	80.35	78.71	74.23	74.65	71.67	62.63	70.59	72.64
MPLCS	78.04	78.59	74.51	82.81	80.75	81.06	78.24	81.06	81.06	68.04	77.8	84.62
PBIL	79.25	81.23	70.315	78.14	75.62	78.59	80.39	81.06	76.02	73.6	80.86	78.43
SGA	77.63	78.59	77.025	80.12	76.09	80.86	75.21	78.73	80.96	73.56	80.96	77.94
SSMA	79.01	78.66	80.74	82.81	81.06	80.75	80.26	83.47	82.5	73.56	80.96	77.94
UCS	60.61	60.61	64.56	60.61	60.61	60.61	58.02	60.61	60.63	63.22	63.22	63.22
XCS	79.17	78.66	77.09	80.75	78.24	78.43	80.64	75.88	80.86	65.2	78.04	80.62
AdaBoost	71.45	70.91	60.17	81.99	75.88	73.16	75.21	72.99	72.81	75.05	75.52	63.22
Bagging	73.99	76.02	72.81	82.5	78.51	80.96	79.82	63.16	77.8	68	78.04	73.99
C4.5	75.99	81.15	69.88	82.5	83.23	78.04	73.82	77.8	77.28	70.51	77.92	73.99
PART	34.85	81.15	50.17	60.65	78.43	42.35	47.65	37.11	34.52	57.03	29.29	44.56
RIPPER	65.67	55.36	63.19	57.9	60.58	79.25	73.24	76.67	78.63	89.11	78.24	75.71
SLIPPER	74.29	73.44	70.45	78.04	78.14	73.24	72.13	75.52	75.21	65.86	74.71	79.48
CHI-RW	77.77	77.57	79.73	72.65	81.63	79.85	74.81	48.33	48.33	68.57	48.33	84.55
KNN	71.31	73.38	68.4	68.4	73.16	75.21	74.71	70.86	67.25	64.07	77.68	63.22
KSTAR	64.33	65.62	64.11	79.67	68.1	73.5	78.34	73.38	71.19	68.75	76.02	73.99
LR	85.84	85.8	84.99	86.32	84.81	84.02	81.93	78.66	80.54	80.26	80.96	63.22
PUBLIC	77.84	81.23	74.71	83.78	80.86	80.96	80.64	70.67	72.43	36.04	76.02	74.58
BNGE	62.38	68.27	60.38	76.34	68.03	73.24	77.13	37.33	70.02	64.6	74.97	63.22
EACH	34.53	34.53	34.53	34.53	34.53	34.53	34.53	34.53	32.24	34.53	34.53	73.99
RISE	59.66	60.47	54.99	76.07	70.74	75.88	79.97	67.64	69.26	61.01	74.84	63.22

SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN

Table 6.18: Balance Results of SMP Models after Data Resampling on Log4j Dataset

Technique	**Adasyn**	**BSMOTE**	**ROS**	**SSMOTE**	**SMOTE**	**SMENN**	**SMTL**	**SP**	**SPII**	**CNN**	**NCL**	**RUS**
BIOHEL	61.33	54.19	59.55	66.36	63.71	63.43	63.52	65.21	68.94	62.9	79.33	71.19
CHC	74.23	76.26	79.73	75.8	77.84	75.07	75.63	59.15	65.11	56.07	70.01	73.63
CPSO	62.92	73.69	55.92	73.5	67.95	70.63	64.19	68.86	61.67	63.17	66.74	75.41
GGA	69.17	69.89	65.61	74.25	69.63	71.85	76.97	59.09	69.7	45.96	72.93	39.39
GA-Int	70.48	69.95	70.23	74.42	71.01	72.09	77.06	59.25	60.25	91.05	70.86	70.83
GA-ADI	68.15	65.94	67.52	73.15	71.39	74.91	74.6	62.41	61.55	60.92	68.56	69.64
IGA	68.69	72.67	69.43	69.51	70.82	67.22	68.81	64.84	67.66	52.59	61.28	76.38
LWPSO	68.15	71.2	65.75	74.25	67.73	73.47	67.2	59.01	68.92	67.26	67.13	67.6
MPLCS	73.13	69.89	64.82	70.24	69.37	70.07	69.26	57.45	65.21	54.93	64.95	67.08
PBIL	68.54	71.75	72.59	75.29	74.12	70.92	75.55	62.37	65.35	52.59	66.86	76.38
SGA	65.38	66.14	70.91	72.9	69.91	71.85	74.98	62.59	70.29	52.65	66.67	74.25
SSMA	69.59	69.01	67.58	74.23	70.63	76.89	74	65.86	63.33	52.65	66.67	74.25
UCS	51.13	51.13	51.07	51.02	51.13	56.23	51.13	51.14	51.14	52.78	52.78	52.78
XCS	72.27	62.37	62.96	72.77	67.86	67.23	72.17	53.33	63.88	60.25	67.92	70.93
AdaBoost	63.52	57.56	57.26	64.05	58.73	58.73	62.76	60.46	63.22	57.75	66.14	52.78
Bagging	68.88	60.81	61.8	69.14	61.22	66.01	71.3	54.06	56.93	57.63	65.72	52.78
C4.5	70.18	57.42	61.5	71.34	64.07	59.66	64.09	60.65	64.84	54.38	58.73	67.45
PART	58.6	53.97	61.2	60.13	63.53	59.1	50.8	32.65	39.38	52.74	32.66	50.04
RIPPER	62.88	54.34	57.39	69.57	57.16	70.78	61.5	64.67	64.42	52.13	71.84	60.73
SLIPPER	62.18	60.68	61.98	66.36	62.94	64.89	65.94	55.24	62.3	62.04	61.65	63.96
CHI-RW	69.19	77.69	75.74	73.69	75.49	76.6	71.01	36.15	54.55	35.83	49.67	69.64
KNN	68.92	65.23	44.02	44.02	69.91	74.12	73.12	31.19	80.6	48.87	59.66	60.73
KSTAR	47.39	57.75	71.48	73.24	51.04	51.08	60.61	52.78	57.58	66.83	51.1	69.64
LR	71.54	74.12	73.76	72.47	74.07	75.63	74.03	55.93	71.75	46	70.44	60.73
PUBLIC	66.22	69.37	70.07	66.87	66.37	68.58	72.96	52.45	52.37	29.29	57.67	84.5
BNGE	58.73	59.02	60.84	65.94	60.02	64.42	63.22	57.58	62.3	61.44	67.52	69.64
EACH	47.34	47.34	44.88	47.34	47.34	47.34	47.34	47.34	47.34	47.34	47.34	74.25
RISE	54.98	57.39	55.32	66.74	58.3	63.06	63.22	65.48	62.82	58.95	69.26	74.25
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

Table 6.19: Balance Results of SMP Models after Data Resampling on Ode Dataset

Technique	**Adasyn**	**BSMOTE**	**ROS**	**SSMOTE**	**SMOTE**	**SMENN**	**SMTL**	**SP**	**SPII**	**CNN**	**NCL**	**RUS**
BIOHEL	62.03	51.46	59.54	64.59	60.72	66.17	64.67	50.69	59.22	53.22	53.22	67.82
CHC	69.03	55.66	70.84	70.84	63.65	68.33	69.72	51.6	57.64	57.73	57.73	69.63
CPSO	50.5	62.77	60.12	57	54.01	54.8	71.53	55.6	62.44	54.07	54.07	70.09
GGA	58.87	56.15	67.42	67.42	65.23	65.07	65.23	56.5	52.65	53.94	53.94	73.24
GA-Int	70.24	55.62	63.72	70.37	69.66	69.63	66.11	42.92	55.08	47.76	47.76	66.87
GA-ADI	68.98	60.96	69.12	70.69	70.02	71.95	69.7	45.38	57.43	49.02	49.02	70.92
IGA	56.09	55.87	62.54	64.86	65.11	61.85	66.49	50.26	55	56.2	56.2	68.85
LWPSO	53.44	57.19	61.09	60.03	56.05	59.23	51.02	63.52	54.89	59.64	59.64	67.55
MPLCS	67.2	51.24	66.07	68.13	69.91	66.41	70.49	47.82	53.69	52.74	52.74	71.17
PBIL	59.73	60.52	66.49	66.48	60.87	65.13	68.22	54.06	55.05	56.28	56.28	68.85
SGA	57.18	53.34	65.38	67.5	63.42	61.97	69	55.3	51.39	51.43	51.43	69.47
SSMA	59.63	57.24	68.89	68.89	65.42	63.03	65.23	54.04	56.13	51.43	51.43	69.47
UCS	41.67	40.45	40.44	41.67	41.68	37.96	41.68	44.15	42.93	39.21	39.21	39.39
XCS	66.1	50.42	63.32	68.54	64.73	65.89	69.35	42.93	45.29	44.11	44.11	72.53
AdaBoost	56.7	50.2	48.89	65.77	59.31	65.13	65.82	55.22	52.65	57.33	57.33	66.87
Bagging	60.13	50.24	52.44	68.39	61.43	67.44	67.09	51.49	51.39	53.85	53.85	39.57
C4.5	62.7	48.95	42.61	68.43	63.77	70.17	66.66	53.92	56.15	53.98	53.98	67.22
PART	45.47	38.86	56.64	54.39	54.09	35.98	46.14	31.77	32.99	29.29	29.29	60.87
RIPPER	56.83	76.29	53.97	72.59	66.83	70.49	66	62.58	61.34	68.23	68.23	65.5
SLIPPER	60.03	52.59	51.21	62.64	62.13	64.34	67.82	53.92	50.18	56.14	56.14	73.58
CHI-RW	68.1	84.69	72.66	69.74	71.31	69.98	68.73	35.95	35.95	39.56	39.56	58.35
KNN	54.01	58.57	47.79	47.17	60.82	60.87	62.5	47.78	51.3	52.79	52.79	66.87
KSTAR	48.83	48.8	60.06	63.37	57.34	59.46	63.95	29.29	31.77	29.29	29.29	39.57
LR	71.3	67.13	72.44	70.94	72.74	72.7	72.92	45.37	53.86	45.37	45.37	66.87
PUBLIC	72.26	51.34	49.7	65.71	68.33	72.04	67.17	55.94	58.86	50.15	50.15	39.57
BNGE	56.83	65.86	57.5	60.93	71.75	70.99	69.11	47.84	52.8	53.82	53.82	39.57
EACH	62.02	36.73	36.73	36.73	36.73	36.73	36.73	36.73	36.73	36.73	36.73	39.57
RISE	40.28	58.88	51.39	60.9	63.6	65.86	66.52	46.59	49.01	57.16	57.16	39.57
SSMOTE indicates SafeSMOTE, SP indicates SPIDER, SPII indicates SPIDERII, SMTL indicates SMOTE-TL, SMENN indicates SMOTE-ENN												

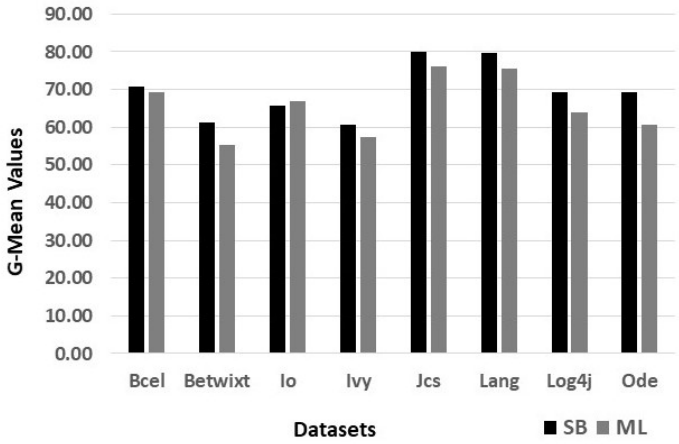


Figure 6.4: Median G-Mean values of SMP Models Developed with SB and ML Techniques after Data Resampling

6.3.4 Results Specific to RQ3

As discussed in Section 6.3.3, the performance of the SMP models has shown a vast improvement in terms of G-Mean and Balance performance metrics. To carry our investigation further, we were interested in detecting which of the data resampling technique would improve the performance of SMP models. To investigate this, we carried out statistical analysis using the Friedman test. We applied the Friedman test by evaluating the G-Mean and Balance of prediction models developed after all data resampling techniques on all datasets under examination in this work. The Friedman test is applied at a level of significance $\alpha = 0.05$ with 12 degrees of freedom. We tested the following hypothesis using the Friedman test.

- ***Null hypothesis-***Ho_1: SMP models built after pre-processing the imbalanced datasets with the data resampling techniques (Adasyn, BSMOTE, ROS, SafeSMOTE, SMOTE, SMOTE-ENN, SMOTE-TL, SPIDER, SPIDER II, CNN, NCL, RUS) do not show a significant difference in the G-Mean performance.

- ***Alternate hypothesis-*** Ha_1: SMP models built after pre-processing the imbalanced datasets with the data resampling techniques (Adasyn, BSMOTE, ROS, SafeSMOTE, SMOTE, SMOTE-ENN, SMOTE-TL, SPIDER, SPIDER II, CNN, NCL, RUS) show a significant difference in the performance evaluated in terms of performance metric G-Mean.

- ***Null hypothesis-*** Ho_2: SMP models built after pre-processing the imbalanced datasets with the data resampling techniques (Adasyn, BSMOTE, ROS, SafeSMOTE, SMOTE, SMOTE-ENN, SMOTE-TL, SPIDER, SPIDER II, CNN, NCL, RUS) do not show a significant difference in the performance evaluated in terms of performance metric Balance.

- ***Alternate hypothesis-*** Ha_2***:*** SMP models built after pre-processing the imbalanced datasets with the data resampling techniques (Adasyn, BSMOTE, ROS, SafeSMOTE, SMOTE, SMOTE-ENN, SMOTE-TL, SPIDER, SPIDER II, CNN, NCL, RUS) show a significant difference in the performance evaluated in terms of performance metric Balance.

Table 6.20 shows the results of the Friedman test carried out for G-Mean and Balance performance metrics. The p-value obtained after the Friedman test was 00 (i.e., p-value < 0.05) both for G-Mean and Balance, i.e., the results of the Friedman test were significant.

Table 6.20: Results of Friedman Test

Data Resampling Technique	**Mean Rank with respect to G-Mean**	**Mean Rank with respect to Balance**
SafeSMOTE	4.41 (Rank 1)	4.24 (Rank 1)
RUS	4.88 (Rank 2)	5.61 (Rank 5)
SMOTE-ENN	5.37 (Rank 3)	5.33 (Rank 3)
SMOTE-TL	5.39 (Rank 4)	5.17 (Rank 2)
SMOTE	5.70 (Rank 5)	5.42 (Rank 4)

Adasyn	6.14 (Rank 6)	5.72 (Rank 6)
NCL	6.62 (Rank 7)	7.33 (Rank 8)
BSMOTE	7.10 (Rank 8)	7.49 (Rank 9)
SPIDERII	7.64 (Rank 9)	7.68 (Rank 10)
ROS	7.97 (Rank 10)	7.32 (Rank 7)
SPIDER	8.56 (Rank 11)	8.73 (Rank 11)
CNN	9.70 (Rank 12)	9.41 (Rank 12)
No-Resampling	11.51 (Rank 13)	11.54 (Rank 13)

Therefore, the null hypothesis Ho_1 and Ho_2 were rejected, which stated that the performance of SMP models after applying different data resampling techniques is the same. The mean ranks assigned to each data resampling technique for performance metrics G-Mean and Balance are presented in Table 6.20. It is to be noted that the SafeSMOTE technique has obtained the best rank (lowest rank values) for G-Mean and Balance performance metrics. The data resampling techniques RUS, SMOTE-ENN, SMOTE-TL, and SMOTE are amongst the top five rankers as per the Friedman test analysis for G-Mean and Balance. The performance of the software maintainability models is very poor when datasets are imbalanced as the no-resampling scenario has obtained the worst rank in Friedman test analysis.

As per the results of the Friedman test, SafeSMOTE was the best data resampling technique that improved the performance of SMP models for G-Mean and Balance. To further examine and confirm whether SafeSMOTE is superior to other data resampling techniques used in this work, post-hoc analysis by Wilcoxon signed-rank test with Bonferroni correction was carried out. The performance of models employed after SafeSMOTE and other data-resampling techniques was compared in terms of G-Mean and Balance. In this work, for the Wilcoxon signed-rank test, the null and alternative hypothesis are given as follows:

Ho_3: $\text{G-Mean}_{\text{SafeSMOTE}} = \text{G-Mean}_{\text{X}}$

Ha_3: $\text{G-Mean}_{\text{SafeSMOTE}} \neq \text{G-Mean}_{\text{X}}$

where X denotes Adasyn, BSMOTE, ROS, SMOTE-ENN, SMOTE-TL, SMOTE, SPIDER, SPIDER II, CNN, NCL, RUS. Similarly, the null and alternate hypothesis for Balance is given as follows:

Ho_4: $\text{Balance}_{\text{SafeSMOTE}} = \text{Balance}_{\text{X}}$

Ha_4: $\text{Balance}_{\text{SafeSMOTE}} \neq \text{Balance}_{\text{X}}$

The level of significance $\alpha = 0.05$, with Bonferroni correction, was taken for Wilcoxon signed-rank test. As we compared 12 pairs of resampling techniques with the Wilcoxon signed-rank test, so with Bonferroni correction if the p-value obtained would be greater than 0.004 (i.e., 0.05/12 = 0.004), the null hypothesis Ho_3 or Ho_4 would be rejected. The results of Wilcoxon signed-rank on evaluating the G-Mean and Balance performance of SMP models after employing SafeSMOTE and other data resampling techniques are shown in Table 6.21.

Table 6.21: Results of Wilcoxon Test

Pair of Techniques	**G-Mean**	**Balance**
SafeSMOTE vs. Adasyn	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
Safe- SMOTE vs. BSMOTE	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
SafeSMOTE vs. SMOTE-ENN	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
SafeSMOTE vs. SMOTE	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
SafeSMOTE vs. SMOTE-TL	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
SafeSMOTE vs. SPIDER	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
SafeSMOTE vs. SPIDERII	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
SafeSMOTE vs. ROS	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
SafeSMOTE vs. CNN	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
SafeSMOTE vs. NCL	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)
SafeSMOTE vs. RUS	NotSig.(p-value = 0.878)	NotSig.(p-value = 0.005)
SafeSMOTE vs. No-Resampling	Sig.(p-value = 0.000)	Sig.(p-value = 0.000)

Table 6.21 shows the test statistics obtained after applying the Wilcoxon signed-rank test on the pair-wise performance of SafeSMOTE and all other data resampling techniques used in this chapter. In Table 6.21, Sig. indicates the significant difference in the performance of a pair of compared techniques and NotSig. denoted that there

is no significant difference in the corresponding pair of techniques. The results of the Wilcoxon signed-rank test indicate that the SafeSMOTE technique is significantly superior to Adasyn, BSMOTE, ROS, SMOTE-ENN, SMOTE-TL, SMOTE, SPIDER, SPIDER II, CNN, NCL, and No-Resampling scenario in terms of G-Mean and Balance performance metrics. However, the RUS technique that was amongst the top five rankers according to the Friedman test results in terms of G-Mean and Balance, is not found significantly different (p-value $>$ 0.004) in the performance in terms of G-Mean and Balance according to the results of Wilcoxon signed-rank test i.e., the performance of RUS is comparable with SafeSMOTE.

6.3.5 Results Specific to RQ4

In this RQ, we compare the performance of different classification techniques after data resampling. To make this comparison, we conducted the Friedman test for G-Mean and Balance after applying all data resampling techniques. For example, to evaluate the performance of the X technique, we extracted the G-Mean and Balance values of SMP models developed on all eight datasets after applying all data resampling techniques. The Friedman test analysis was carried out at a significance level of $\alpha = 0.05$ with 27 degrees of freedom (28 classification techniques). The Friedman test assesses the hypothesis that the performance of different ML and SB techniques on all datasets is not different from each other. The mean ranks obtained after the Friedman test in terms of G-Mean and Balance values of all ML and SB techniques (median G-Mean and Balance on ten runs of SB techniques) on all eight datasets and all data resampling techniques are depicted in Table 6.22. The p-values obtained after conducting the test were 0.00 both for G-Mean and Balance that indicate the results are significant. It yields the rejection of the null hypothesis and leads to the conclusion that the performance of different ML and SB techniques on all datasets are different

from each other. As shown in Table 6.22, the CHC technique has obtained the best rank in terms of G-Mean and second-best rank in terms of Balance whereas the LR technique has obtained the best rank in terms of G-Mean and second-best rank in terms of Balance. It is to be noted that CHC, SGA, PBIL, SSMA, GA-Int, GA-ADI, GGA, MLPCS, and CSPO are the top ten classification techniques as per the Friedman test ranking in terms of G-Mean and Balance and all these techniques are SB techniques. This implies that these techniques can be used for developing prediction models for software maintainability across different software systems. The worst performing techniques in terms of G-Mean and Balance are RIPPER, AB, BNGE, RISE, KNN, KSTAR, EACH, PART, and UCS. The PART technique has obtained the worst rank.

As the results of the Friedman test were significant, we further carry out post-hoc analysis with the help of the Wilcoxon signed-rank test with Bonferroni correction where we have adjusted the level of significance to α = 0.05/27 = 0.001 as we have evaluated 27 pairs of classification techniques. The results of the Wilcoxon test are shown in Table 6.23. As per the Friedman test ranking the most accurate method for predicting the software maintainability was LR in terms of G-Mean values. Also, according to the Wilcoxon signed-rank test, LR is significantly superior to 20 out of 28 classification techniques. Further LR is not significantly different in the performance in terms of G-Mean at α = 0.05/27 = 0.001 for 7 out of 28 classification techniques namely CHC, SGA, PBIL, SSMA, GA-ADI, GA-Int, and CPSO. CHC technique was the best SB technique for predicting the software maintainability in terms of Balance values. Also, CHC is found superior to 23 out of 28 classification techniques according to the Wilcoxon signed-rank test at α = 0.05/27=0.001. CHC is not significantly different in the performance in terms of Balance for 6 out of 28 classification techniques namely LR, SSMA, CPSO, GA-ADI, GA-Int, and CPSO. Also, CHC, LR, SGA, PBIL, SSMA, GA-Int, GA-ADI, GGA, MLPCS, and CSO are the top ten classification techniques as per the Friedman test. All of these are SB

techniques except LR.

Table 6.22: Friedman Test Results for Performance of Classification Techniques

Classification Techniques	**Category**	**Balance Rank**	**Classification Techniques**	**Category**	**G-Mean Rank**
CHC	SB	7.79	LR	ML	7.57
LR	ML	7.8	CHC	SB	8.06
SGA	SB	10.51	GA-Int	SB	10.1
PBIL	SB	10.51	GA-ADI	SB	10.5
SSMA	SB	10.51	SSMA	SB	10.57
GA-Int	SB	10.66	PBIL	SB	10.72
GA-ADI	SB	10.68	SGA	SB	10.94
GGA	SB	11.38	GGA	SB	11.73
MPLCS	SB	11.91	MPLCS	SB	11.77
CPSO	SB	12.05	CPSO	SB	12.85
LWPSO	SB	13.13	XCS	SB	13.19
SLIPPER	ML	13.41	SLIPPER	ML	13.56
XCS	SB	13.63	CHIRW	ML	13.86
Bagging	ML	14.65	Bagging	ML	13.93
IGA	SB	14.72	LWPSO	SB	14.63
PUBLIC	ML	14.84	PUBLIC	ML	14.87
CHIRW	ML	14.95	BIOHEL	SB	14.91
C4.5	ML	15.04	C4.5	ML	15.05
BIOHEL	SB	15.38	IGA	SB	15.13
RIPPER	ML	15.59	RIPPER	ML	15.86
AB	ML	15.94	AB	ML	15.98
BNGE	ML	16.87	BNGE	ML	16.68
RISE	ML	17.46	RISE	ML	17.55
KNN	ML	17.61	KNN	ML	17.83
KSTAR	ML	20.25	KSTAR	ML	19.4
EACH	ML	21.14	EACH	ML	21.18
PART	ML	22.38	PART	ML	22.57
UCS	SB	25.25	UCS	SB	25.01

Table 6.23: Wilcoxon Test Results for Performance of Classification Techniques

Pair Examined	**G-Mean**	**Pair Examined**	**Balance**
LR vs. CHC	NotSig.(p-value = 0.324)	CHC vs. LR	NotSig. (p-value = 0.423)
LR vs. SGA	NotSig. (p-value = 0.001)	CHC vs. SGA	Sig. (p-value = 0.000)

LR vs. PBIL	NotSig. (p-value = 0.002)	CHC vs. PBIL	Sig. (p-value = 0.000)
LR vs. SSMA	NotSig. (p-value = 0.005)	CHC vs. SSMA	NotSig. (p-value = 0.001)
LR vs. GA-Int	NotSig. (p-value = 0.008)	CHC vs. GA-Int	NotSig. (p-value = 0.001)
LR vs. GA-ADI	NotSig. (p-value = 0.003)	CHC vs. GA-ADI	NotSig. (p-value = 0.001)
LR vs. GGA	Sig. (p-value = 0.000)	CHC vs. GGA	Sig. (p-value = 0.000)
LR vs. MPLCS	Sig. (p-value = 0.000)	CHC vs. MPLCS	Sig. (p-value = 0.000)
LR vs. CPSO	NotSig. (p-value = 0.010)	CHC vs. CPSO	NotSig. (p- value = 0.011)
LR vs. LWPSO	Sig. (p-value = 0.000)	CHC vs. LWPSO	Sig. (p-value = 0.000)
LR vs. SLIPPER	Sig. (p-value = 0.000)	CHC vs. SLIPPER	Sig. (p-value = 0.000)
LR vs. XCS	Sig. (p-value = 0.000)	CHC vs. XCS	Sig. (p-value = 0.000)
LR vs. Bagging	Sig. (p-value = 0.000)	CHC vs. Bagging	Sig. (p-value = 0.000)
LR vs. IGA	Sig. (p-value = 0.000)	CHC vs. IGA	Sig. (p-value = 0.000)
LR vs. PUBLIC	Sig. (p-value = 0.000)	CHC vs. PUBLIC	Sig. (p-value = 0.000)
LR vs. CHIRW	Sig. (p-value = 0.000)	CHC vs. CHIRW	Sig. (p-value = 0.000)
LR vs. C4.5	Sig. (p-value = 0.000)	CHC vs. C4.5	Sig. (p-value = 0.000)
LR vs. BIOHEL	Sig. (p-value = 0.000)	CHC vs. BIOHEL	Sig. (p-value = 0.000)
LR vs. RIPPER	Sig. (p-value = 0.000)	CHC vs. RIPPER	Sig. (p-value = 0.000)
LR vs. AB	Sig. (p-value = 0.000)	CHC vs. AB	Sig. (p-value = 0.000)
LR vs. BNGE	Sig. (p-value = 0.000)	CHC vs. BNGE	Sig. (p-value = 0.000)
LR vs. RISE	Sig. (p-value = 0.000)	CHC vs. RISE	Sig.(p-value = 0.000)
LR vs. KNN	Sig.(p-value = 0.000)	CHC vs. KNN	Sig.(p-value = 0.000)
LR vs. KSTAR	Sig.(p-value = 0.000)	CHC vs. KSTAR	Sig.(p-value = 0.000)
LR vs. EACH	Sig.(p-value = 0.000)	CHC vs. EACH	Sig.(p-value = 0.000)
LR vs. PART	Sig.(p-value = 0.000)	CHC vs. PART	Sig.(p-value = 0.000)
LR vs. UCS	Sig.(p-value = 0.000)	CHC vs. UCS	Sig.(p-value = 0.000)

6.4 Discussion

The results stated in previous sections indicate good performance for the development of classification models that identify maintainability of classes in a software dataset after data resampling techniques compared to a situation when the datasets are imbalanced. We also analyzed percentage improvement in the performance of the models developed using SB & ML techniques after data resampling. For SMP models developed using SB techniques after data resampling showed an improvement of 23.99%

on the Bcel dataset after data resampling, 222.61% for the Betwixt dataset,48.48% for the Io dataset, 90.87% for the Ivy dataset, 23.63% for Jcs dataset, 3.34% for Lang dataset, 39.94% for Log4j dataset, and 85.36% for Ode dataset respect to performance metric G-Mean. For SMP models developed using ML techniques after data resampling showed an improvement of respect to performance metric G-Mean, 39.70% on the Bcel dataset after data resampling, 69.58% for the Betwixt dataset,22.75% for the Io dataset, 44.70% for the Ivy dataset, 17.62% for Jcs dataset, 14.04% for Lang dataset, 53.09% for Log4j dataset, and 45.17% for Ode dataset. Also, there is a relatively significant improvement in the performance of the prediction models after data resampling for the Balance performance metric. The prediction models developed using SB techniques after data resampling gave a gain of 29.45% for Bcel 86.01% for the Betwixt dataset, 47.08% for the Io dataset, 60.49% for the Ivy dataset, 27.77% for the Jcs dataset, 6.12% for the Lang dataset, 34.35% for Log4j dataset, and 53.02% for Ode dataset with respect to Balance. The prediction models developed using ML techniques after data resampling gave a gain of 39.25% for Bcel 47.19% for the Betwixt dataset, 26.61% for Io dataset, 37.16% for the Ivy dataset, 25.56% for the Jcs dataset, 16.45% for the Lang dataset, 45.57% for Log4j dataset, and 34.32% for Ode dataset with respect to Balance. The results of this chapter indicate that after data resampling, the performance of models developed with SB techniques is better than that of models developed using ML techniques. Also, the results of the Friedman test indicate that SafeSMOTE is an effective technique to deal with the imbalanced data, and the performance of models developed after SafeSMOTE is significantly superior to that of models developed after data resampling with other investigated techniques.

It was also observed that SB techniques CHC, SGA, PBIL, SSMA, GA-ADI, GA-Int, and CPSO are the competent techniques for predicting low maintainability software classes accurately. The worst performing techniques in terms of G-Mean and Balance are BNGE, RISE, KNN, KSTAR, EACH, PART, and UCS. All of these are

ML techniques except UCS, which is an SB technique. Furthermore, the result of the Wilcoxon test indicates that the performance of the CHC technique was better than the majority of the other techniques explored in this chapter. The superior performance of the CHC technique could be attributed to its effective fitness function which tries to balance the complexity as well as the accuracy of the rule set obtained for classifying low maintainability. The results indicate better performance of SB techniques as compared to ML techniques. Hence, this empirical experiment favors of development of prediction models using SB techniques for the identification of low maintainability classes in OO systems. Such models can be efficiently used for OO software projects so that low maintainability classes can effectively be identified.

Chapter 7

Hybridized Techniques for Software Maintainability Prediction with Imbalanced Data

7.1 Introduction

To develop effective software quality models, numerous classification techniques including statistical techniques, ML techniques, and SB techniques are available. In recent literature, various researchers encouraged the use of SB techniques to develop software quality models [211–213]. These techniques explore a large solution space to find an optimal solution to a problem where each solution is gauged by making use of a fitness function to discover the correctness of the obtained solution [220]. The parallel and robust nature of SB techniques permits exploring a considerable solution space with a challenge to ascertain an optimal solution. Various operators like crossover, mutation, etc. are used for discovering the optimal or near-optimal

solution [220]. Also, the ability to handle noisy data makes SB techniques effective for predictive modeling [212]. The performance metrics are represented as the fitness functions, which are then optimized in order to find the best solution for a predictive modeling task [211]. The results of Chapter 3 indicate that the HB techniques exhibit good performance to develop SMP models. However, very few studies in the literature evaluated their effectiveness.

Therefore, it is important to assess the competence of HB techniques in the domain of SMP. HB techniques are another category of SB technique, as these techniques associate SB techniques with ML techniques as one single method. HB techniques may yield improved results by incorporating the competence of both ML and SB techniques into one single technique. A study by [221] documented the capability of HB techniques for software defect prediction and specified that the use of these techniques would help in producing optimal solutions. Malhotra and Khanna [17] advocated the use of HB techniques to develop change-proneness prediction models. In predictive modeling, combining ML techniques with SB techniques increase the convergence speed of prediction models during development. These characteristics of HB techniques motivate us to analyze their effectiveness in SMP.

To investigate the application of HB techniques for the determination of the maintainability of classes, we used the open-source project datasets used in Chapters 4, 5, and 6. Like Chapters 4 and 6, in this chapter, we efficiently preprocess the imbalanced datasets in order to attain a uniform distribution of datapoints of low maintainability and high maintainability class datapoints before developing the SMP models using HB techniques. To do so, we apply data resampling methods used in Chapters 4 and 6. Also, in order to investigate the effective application HB techniques for SMP, we follow a generalized and repeatable approach in this chapter, similar to the one followed in Chapter 6. The approach includes the complete specification of parameter settings and using multiple executions of these techniques due to their

stochastic nature. SMP models are developed using eleven HB techniques in this chapter. These techniques DT-GA, TARGET, GFS-LB, GFS-MaxLB, GFS-AB, GP-COACH, PSOLDA, GFS-GP, GFS-GPG, GFS-SP and GFS-GCCL. The results are analyzed with the data resampling methods namely SMOTE, BSMOTE, SafeSMOTE, Adasyn, ROS, RUS, SPIDER, SPIDER II, SMOTE-ENN, SMOTE-TL, CNN, CNN-TL, and NCL. The following research questions are addressed in this chapter.

- RQ1) What is the effect of data resampling techniques on the SMP models? Do the data resampling techniques improve the performance of SMP models?
- RQ2) Is the effect of data resampling methods on the SMP models statistically significant regarding the performance metrics used?
- RQ3) Which is the best performing HB technique for predicting software maintainability? Is the best HB technique statistically better than the other investigated techniques?

The organization of this chapter is as follows: Section 7.1 describes the framework of the experiment, Section 7.2 describes results and analysis. Section 7.3 presents a discussion. The results of this chapter are published in [222, 223].

7.2 Framework of Experiment

This section describes the experimental framework of the chapter.

7.2.1 Datasets and Variables used for Empirical Validation

This chapter makes use of the eight open-source datasets for empirical validation. These datasets have been used in the previous chapters i.e., in Chapters 4 to 6. The data points in these datasets are eighteen OO metrics describing various characteristics

of OO software. A detailed description of these variables is given in Chapter 2. The dependent variable used in this chapter is "maintainability". This is a binary variable. The data collection procedure from the datasets has been discussed in detail in the previous chapters.

7.2.2 Model Development and Validation

In this chapter, the imbalanced datasets are preprocessed with the same data resampling techniques investigated in previous chapters (i.e., Chapters 4 and 6). SMP models are then trained with the aid of HB techniques and results are analyzed with the help of G-Mean and Balance performance metrics. The models are developed using ten-fold cross-validation with ten runs of each HB technique on each dataset.

7.2.3 Statistical Analysis

To test whether there is a statistical difference in the performance of investigated techniques, we gave statistical support to the results. Two non-parametric statistical tests are used for statistical analysis at a level of significance of α = 0.05. We perform two types of statistical comparison in this chapter namely multiple and pairwise. Multiple comparisons are performed among a group of techniques for which we use the Friedman test. In multiple-comparison with the Friedman test, we determined if the performance of the different techniques is the same or different. With the Friedman test, we want to find out which technique is better than the rest of the investigated techniques. The pairwise comparison is made between a pair of techniques (best techniques obtained after the Friedman test and all other techniques) for which we used the Wilcoxon signed-rank test.

7.3 Results and Analysis

This section presents the answers to the research questions of this chapter and their analysis.

7.3.1 Results and Analysis of RQ1

To assess the effect of data resampling on the performance of the SMP models developed after the application of HB techniques, we developed the models with the datasets in an imbalanced scenario, and after applying resampling techniques stated in Section 7.1. The performance of the developed models for eight datasets used in the chapter using ten-fold cross-validation by applying different HB techniques is reported in Tables 7.1 to 7.8. For each dataset, we run each HB technique ten-times to deal with the stochastic nature of these techniques. For each run, we recorded the performance of the developed SMP models with respect to G-Mean and Balance performance metrics, and the median of all ten runs are logged in Tables 7.1 to 7.8 for the eight datasets used in the chapter. As shown in Table 7.1, for the Bcel dataset, in the no resampling situation i.e., when the SMP models are developed from imbalanced data, the G-Mean, and Balance results are in the range of 0-69.94 and 29.29-69.01 respectively. In the resampling situation, for the Bcel dataset, the average G-Mean values of the SMP models developed using various HB techniques is 38.52, and the average of Balance values just 43.61. Except for the PSOLDA, for all other techniques, the performance of the SMP models on imbalanced Bcel dataset was very poor in terms of G-Mean and Balance. In 63.64% of the cases, the G-Mean values were less than 50, and in 81.81% of the cases, the Balance values were less than 50 when the models are developed with an imbalanced Bcel dataset.

When we applied different data resampling techniques before developing the SMP

models on the Bcel dataset, in 71.32% of the cases, the G-Mean values were reported to be greater than 60, and the average G-Mean was 61.80. Similarly, the average Balance values of the SMP after data resampling was 62.22, and in 69% of the cases, the Balance results were greater than 60. An analysis of Table 7.2 indicated that on applying different data resampling techniques on the Betwixt dataset, the average G-Mean and Balance results were reported to be 60.07 and 60.24, respectively. After applying different data resampling techniques on the Betwixt dataset, in 67.83% of the cases, the G-Mean values were found to be higher than 60. In 58.73% of the cases, the Balance results of the SMP models were reported to be more than 60. The performance of the SMP models after applying different data resampling techniques and in the imbalanced situation for the Io dataset is shown in Table 7.3. On analyzing this Table 7.3, we found that using data resampling techniques, improved the performance of the SMP models a lot. For the Io dataset, the average G-Mean and Balance values were reported to be 69.52 and 67.69, respectively. After applying data resampling techniques, 82.52% of the cases the G-Mean values were obtained to be greater than 60, and Balance results were found to be higher than 60 in 72.72% of the cases. When the Io dataset was imbalanced, the developed SMP models have shown an inferior predictive performance in terms of G-Mean and Balance, i.e., the average of G-Mean and Balance was 38.78 and 44.07, respectively. For the Io dataset, in the imbalanced scenario, the G-Mean values of all HB techniques were observed to be very close to 54, and Balance values were observed to be very close to 50.51 except for DT-GA, GFS-GCCL, and TARGET. For DT-GA, GFS-GCCL, and TARGET, the G-Mean values were 0, and Balance values were 29.29.

Table 7.1: G-Mean and Balance Results for Bcel Dataset

Res.Tech.	HB Tech. G-Mean Values										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	74.11	75.13	79.56	52.7	78.76	65.6	77.95	78.08	75.08	77.95	70.01
BSMOTE	75.27	73.23	61.97	47.14	61.57	69.01	74.5	75.53	61.97	61.97	74.08
ROS	68.83	78.41	69.7	47.14	75.19	47.14	75.15	78.08	61.97	69.7	71.71
SafeSMOTE	77.27	76.49	73.11	46.99	76.64	64.3	75.66	79.85	65.92	73.11	70.22
SMOTE	74.17	76.72	73.23	47.14	76.86	52.5	71.46	74.82	69.98	71.46	70.22
SMENN	73.47	62.06	46.99	46.99	52.54	65.92	73.47	73.59	11.77	52.54	69.11
SMOTE-TL	72.39	74.68	79.56	46.99	76.19	46.99	75.4	77.13	76.04	76.04	69.15
SPIDER	65.5	66.03	0	46.99	69.81	46.99	69.92	69.81	0	46.99	73.13
SPIDERII	71.53	69.7	23.5	46.99	73.23	46.99	75.79	72.26	11.77	46.99	73.9
CNN	54.05	57.37	0	47.14	47.14	57.55	47.07	69.81	0	47.14	69.91
CNN-TL	69.39	76.3	0	46.99	69.36	66.14	72.62	71.65	0	66.14	69.67
NCL	72.75	61.97	0	46.99	69.7	46.99	69.36	69.24	0	46.99	73.13
RUS	78.55	74.98	69.58	65.09	76.94	61.75	77.68	79.13	61.77	69.58	70.03
No Res.	40.57	57.55	0	47.14	52.54	47.14	40.76	52.37	0	47.14	69.94
Res.Tech.	**HB Tech. Balance Values**										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	72.81	75.02	76.16	48.93	78.74	60.65	75.61	75.67	70.445	71.975	65.76
BSMOTE	72.02	68.48	56.78	45.00	56.75	64.49	71.75	72.1	56.78	72.02	73.92
ROS	68.24	77.72	64.59	45.00	74.1	45	71.98	75.67	56.78	77.72	70.47
SafeSMOTE	75.03	76.21	68.46	45.00	76.33	60.41	72.14	78.68	60.69	76.05	68.82
SMOTE	71.64	76.40	68.47	45.00	76.52	48.93	68.06	71.87	64.62	72.08	68.82
SMENN	68.51	56.78	45	45.00	48.93	60.685	68.51	68.52	31.255	68.51	68.46
SMOTE-TL	70.72	74.6	76.16	45.00	75.95	45.00	72.06	75.23	72.24	68.34	66.4
SP	60.64	60.69	29.29	45.00	64.6	45.00	64.61	64.6	29.28	68.15	73.12
SPII	68.08	64.59	33.22	45.00	68.48	45.00	72.17	68.29	31.25	71.85	73.47
CNN	52.06	52.85	29.29	45.00	45	52.86	45.00	64.6	29.29	56.51	67.44
CNN-TL	68.68	72.30	29.29	45.00	64.54	60.7	68.37	68.12	29.29	71.8	68.89
NCL	68.39	56.78	29.29	45.00	64.59	45.00	64.54	64.53	29.29	76.05	73.12
RUS	77.82	74.88	64.57	64.37	76.58	60.86	75.49	78.23	56.77	78.03	68.61
No Res.	41.07	52.86	29.29	45.00	48.93	45.00	41.07	48.92	29.29	45.00	69.01

Res. Tech. indicates Resampling Technique, HB Tech. indicates HB technique, GFS-MLB indicates GFS-MaxLB, No Res. indicates No resampling,SMENN indicates SMOTE-ENN

Table 7.2: G-Mean and Balance Results for Betwixt Dataset

Res.Tech.	HB Tech. G-Mean Values										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	60.33	71.93	71.18	66.5	64.05	69.6	69.66	68.41	60.34	62.07	67.77
BSMOTE	40.8	52.9	66.4	56.53	56.82	60.73	39.32	57.51	60.41	59.72	71.06
ROS	41.63	63.43	72.1	70.77	62.07	72.57	55.7	69.82	67.34	60.61	64.88
SafeSMOTE	60.56	67.98	70.67	65.95	68.65	66.5	64.04	68.63	26.26	59.47	69.36
SMOTE	58.7	69.95	71.68	61.33	65.54	64.28	64.05	68.72	68.61	65.46	64.84
SMENN	63.66	68.48	72.42	67.58	66.98	67.95	57.5	66.01	73.04	89.12	60.91
SMOTE-TL	73.1	64.49	69.61	64.66	64.49	72.36	63.36	66.51	67.05	58.52	73.64
SPIDER	53.04	53.62	0	51.88	56.82	56.82	56.52	56.21	53.04	45.49	68.82
SPIDERII	60.37	59.04	45.6	59.73	61.81	61.28	60.75	64.44	63.05	61.51	69.21
CNN	32.17	45.15	0	19.11	42.54	54.74	55.44	45.82	32.48	19.24	50.56
CNN-TL	59.73	64.96	65.77	61.14	64.78	58.87	69.12	56.4	62.56	57.35	76.55
NCL	73.1	60.07	0	60.83	68.58	54.18	60.58	61.91	66.33	18.84	56.39
RUS	68.63	68.18	72.81	69.22	62.24	60.58	69.03	64.28	68.85	60.08	74.51
No Res.	0	26.84	0	19.25	32.64	19.2	18.84	18.93	0	0	50.68
Res.Tech.	**HB Tech. Balance Values**										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	59.95	71.58	70.77	66.4	63.54	69.16	69.2	68.38	59.14	59.97	67.01
BSMOTE	41.94	51.51	65.35	55.41	54.56	59.78	41.21	56.04	58.73	59.08	69.38
ROS	42.84	63.32	72.06	69.77	61.41	72.55	54.83	69.82	67.18	57.81	63.45
SafeSMOTE	60.54	67.7	70.56	65.53	68.63	66.4	64.03	67.41	34.35	56.22	68.27
SMOTE	58.18	69.8	71.31	61.2	65.23	64.07	63.54	68.49	68.31	60.99	64.18
SMENN	63.2	68.23	72.12	66.85	66.9	67.94	57.39	65.89	73.03	88.95	59.91
SMOTE-TL	71.71	64.46	67.76	62.5	64.48	70.81	63.36	65.7	67.05	56.15	73.44
SP	51.59	51.87	29.29	49.82	54.56	54.56	54.43	54.28	51.59	45.8	65.34
SPII	59.51	57.86	44.82	58.31	60.54	60.17	58.92	62.07	61.32	61.74	67.1
CNN	36.96	44.69	29.28	31.9	42.36	52.32	52.54	44.87	37.04	31.91	47.61
CNN-TL	58.63	64.94	65.63	60.46	64.59	58.15	69.11	56.18	62.48	55.54	75.58
NCL	71.71	58.53	29.29	57.6	67.71	52.11	58.83	59.53	64.31	31.83	52.75
RUS	67.98	68.16	72.79	69.02	61.97	60.57	68.67	64.11	68.8	60.3	74.28
No Res.	29.29	34.5	29.29	31.91	37.08	31.91	31.85	31.87	29.29	29.29	47.62

Res. Tech. indicates Resampling Technique, HB Tech. indicates HB technique, GFS-MLB indicates GFS-MaxLB, No Res. indicates No resampling,SMENN indicates SMOTE-ENN

Table 7.3: G-Mean and Balance Results for Io Dataset

Res.Tech.	HB Tech. G-Mean Values										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	77.93	80.13	74.81	79.28	81.68	72.56	81.57	78.39	77.46	87.57	77.78
BSMOTE	62.31	62.44	73.79	68.6	61.49	53.84	68.91	68.75	61.49	61.77	82.27
ROS	60.53	83.87	75.31	79.28	85.87	80.65	84.09	80.34	78.4	80.13	80.44
SafeSMOTE	54.7	76.12	78	79.28	76.31	77.27	78.03	76.5	78.78	76.7	76.96
SMOTE	72.24	77.78	77.35	78	76.12	79.07	72.41	66.1	81.48	71.18	78.32
SMENN	66.74	72.24	79.07	78	76.12	76.7	73.11	72.24	75.04	70.83	79.12
SMOTE-TL	72.41	80.34	78.99	78.86	76.12	75.54	73.11	66.26	75.04	69.39	77.51
SPIDER	44.15	76.97	44.72	54.54	54.31	54.54	54.19	54.08	80.07	0	44.72
SPIDERII	53.49	69.66	44.67	54.54	61.77	54.54	61.49	53.49	87.62	52.3	44.72
CNN	68.75	62.31	44.63	44.44	69.06	76.97	74.3	66.42	69.51	67.36	62.71
CNN-TL	77.27	72.93	74.25	69.21	65.63	75.54	78.43	69.03	83.01	82.4	83.51
NCL	60.94	62.85	15.81	54.54	61.63	54.54	61.77	68.29	70.11	15.74	31.62
RUS	79.92	79.92	74.3	77.57	85.65	76.04	80.88	84.99	80.44	80.13	79.65
No. Res.	0	54.66	0	54.66	54.54	54.54	54.66	54.31	54.66	0	44.72
Res.Tech.	**HB Tech. Balance Values**										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	76.8	80.13	72.34	79.27	81.25	72.48	81.14	76.48	76.5	83.51	72.07
BSMOTE	57.52	57.54	70.97	64.4	57.4	50.45	64.47	64.43	57.4	57.45	81.11
ROS	57.16	83.01	72.92	79.27	85.4	79.16	83.28	80.33	77.1	80.13	75.04
SafeSMOTE	53.98	75.54	77.94	79.27	75.68	76.38	76.87	75.83	77.32	75.97	71.18
SMOTE	70.25	77.71	77.25	77.71	75.54	78.84	70.34	63.54	75.57	69.65	72.67
SMENN	63.81	70.25	79.06	77.94	75.54	75.97	70.68	70.25	69.1	69.43	73.56
SMOTE-TL	70.34	80.33	77.19	78.84	75.54	75.08	70.68	63.61	69.1	68.46	71.78
SPIDER	43.4	71.7	43.43	50.5	50.49	50.5	50.48	50.47	77.97	29.29	43.43
SPIDERII	50.39	64.58	43.43	50.5	57.45	50.5	57.4	50.39	87.42	50.11	43.43
CNN	64.43	57.52	43.43	43.42	64.49	71.7	71.16	63.68	64.56	64.05	57.56
CNN-TL	76.38	70.6	73.86	68.32	63.31	75.08	78.39	68.19	77.305	82.27	78.61
NCL	57.27	57.56	32.83	50.5	57.42	50.5	57.45	64.33	64.62	32.825	36.36
RUS	79.92	79.92	71.77	77.48	85.14	75.82	79.43	84.35	74.4	80.13	74.15
No. Res.	29.29	50.5	29.29	50.5	50.5	50.5	50.5	50.49	50.5	29.29	43.43

Res. Tech. indicates Resampling Technique, HB Tech. indicates HB technique, GFS-MLB indicates GFS-MaxLB, No Res. indicates No resampling,SMENN indicates SMOTE-ENN

Table 7.4: G-Mean and Balance Results for Ivy Dataset

Res. Tech.	HB Tech. G-Mean Values										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	68.49	71.17	69.58	85.59	73.87	68.49	65.99	64.56	72.96	66.62	71.58
BSMOTE	51.39	65.69	69.19	65.85	70.07	63.1	62.74	66.27	60.81	67.38	71
ROS	47.61	69.81	40.77	70.19	71.66	60	67.58	62.57	64.28	70.18	67.31
SafeSMOTE	72.54	65.99	61.33	61.36	69.13	68.91	58.94	66.08	60.68	67.76	70.2
SMOTE	66.83	62.98	65.85	67.49	67.18	68.86	54.86	65.31	65.82	65.52	75.83
SMENN	61.02	62.11	60.12	73.3	69.4	65.94	55.42	62.57	63.54	65.52	68.19
SMOTE-TL	67.18	62.84	67.46	67.62	69.19	70.04	63.11	66.79	70.47	67.18	64.74
SPIDER	54.76	49.56	26.73	41.32	52.35	56.03	41.03	52.59	57.81	55.02	26.68
SPIDERII	44.01	51.71	22.5	48.89	55.27	49.27	55.69	59.2	66.37	40.94	41.94
CNN	68.75	62.31	44.63	44.44	69.06	76.97	74.3	66.42	69.51	67.36	26.72
CNN-TL	41.39	49.66	43.44	72.15	61.89	56.4	58.52	67.45	46.09	45.88	54.41
NCL	51.55	55.44	26.73	41.32	61.11	41.57	58.71	40.56	58.17	44.29	37.68
RUS	67.18	69.05	72.89	74.65	70.33	73.21	71.3	76.36	66.6	72.9	66.48
No Res.	37.69	26.69	26.73	26.73	42.13	26.71	32.59	26.65	32.64	18.84	32.73
Res.Tech.	**HB Tech. Balance Values**										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	67.9	70.88	69.55	82.25	73.79	67.9	65.77	64.47	71.36	66.16	71.43
BSMOTE	49.21	65.1	69.18	65.65	69.725	62.88	61.58	66.25	60.75	67.26	70.5
ROS	46.56	69.5	42.22	70.13	71.02	58.93	67.16	62.4	63.92	69.57	66.35
SafeSMOTE	72.28	65.77	60.54	61.27	68.91	68.9	57.43	66.06	59.39	67.38	69.84
SMOTE	65.13	62.39	65.65	67.4	66.82	68.66	53.2	65.16	65.79	65.48	75.82
SMNN	58.54	61.67	59.01	73.26	69.14	65.3	53.48	62.4	62.12	65.48	67.8
SMOTE-TL	65.33	62.76	67.45	67.53	69.18	70	62.49	66.33	70.4	66.3	63.7
SPIDER	51.78	46.95	34.34	41.83	49.41	51.99	42.16	49.44	54.32	51.84	34.34
SPIDERII	44.03	49.29	31.18	46.88	51.89	46.93	51.95	56.4	64.84	41.76	87.98
CNN	39.38	36.85	34.34	36.67	46.86	46.85	54.5	56.47	39.39	39.39	79.69
CNN-TL	41.84	48.57	43.8	72.15	60.15	56.06	56.12	67.05	46.33	44.43	64.67
NCL	49.25	51.92	34.34	41.83	56.93	41.87	54.48	41.65	54.39	44.12	63.14
RUS	66.3	69.04	72.84	73.28	70.27	72.9	71.17	74.32	64.99	72.56	85.62
No Res.	39.39	34.34	34.34	34.34	41.91	34.34	36.86	34.34	36.86	31.81	36.87

Res. Tech. indicates Resampling Technique, HB Tech. indicates HB technique, GFS-MLB indicates GFS-MaxLB, No Res. indicates No resampling,SMENN indicates SMOTE-ENN

Table 7.5: G-Mean and Balance Results for Jcs Dataset

Res.Tech.	HB Tech. G-Mean Values										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	78.66	82.5	81.85	88.25	82.85	86.17	87.41	86.47	85.75	79.21	86.14
BSMOTE	74.41	82.5	85.75	65.85	79.49	85.24	79.49	78.38	85.75	78.3	88.11
ROS	67.64	81.03	84.15	84.65	81.62	85.25	83.65	86.17	85.25	82.18	86.79
SafeSMOTE	78.59	82.76	83.17	81.92	83.16	84.64	79.46	81.62	84.47	78.19	87.44
SMOTE	77.47	83.08	83.5	82.21	81.62	86.47	79.46	79.17	84.79	78.19	87.78
SMENN	79.21	82.45	83.17	81.92	81.33	80.03	79.46	81.03	85.43	76.78	87.78
SMOTE-TL	76.18	81.81	83.17	79.17	83.16	81.92	80.74	81.03	85.11	76.53	85.48
SPIDER	70.36	79.76	33.14	77.47	76.94	85.24	76.94	78.01	86.7	80.51	85.79
SPIDERII	76.4	81.92	69.42	82.79	77.82	87.07	80.03	76.4	84.79	81.52	88.56
CNN	73.03	54.23	75.15	68.6	70.36	69.39	64.93	64.64	76.41	73.92	80.57
CNN-TL	66.4	71.95	54.27	69.59	66.78	72.31	60.71	67.17	73.8	74.1	68.44
NCL	71.83	81.45	38.38	80.3	84.06	86.17	83.37	75.67	86.38	83.37	67.52
RUS	75.84	78.94	84.15	77.26	78.01	82.21	78.38	80.74	81.17	80.52	87.77
No Res.	59.96	59.96	0	53.95	69.64	68.76	78.96	69.64	84.35	42.4	50.46
Res.Tech.	**HB Tech. Balance Values**										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	78.28	82.48	79.76	87.61	82.73	85.95	84.24	86.29	84.53	77.77	83.57
BSMOTE	72.52	82.48	84.53	65.65	78.925	85.24	78.92	78.06	84.53	78.29	84.19
ROS	64.85	81.03	82.56	84.64	81.62	84.9	83.53	85.95	84.9	80.16	84.39
SafeSMOTE	78.86	81.98	81.36	81.91	83.06	84.18	79.41	81.62	82.95	77.45	83.36
SMOTE	76.79	82.35	81.76	82.2	81.62	86.29	79.41	79.14	83.35	77.45	83.78
SMENN	78.92	81.6	81.36	81.91	81.33	79.93	79.41	81.03	84.14	76.47	83.78
SMOTE-TL	74.56	80.84	81.36	79.14	83.06	81.91	80.73	81.03	83.74	76.5	82.75
SPIDER	69.08	79.13	37.14	76.61	76.23	85.24	76.23	76.97	85.695	78.15	81.66
SPIDERII	76.03	81.91	65.59	82.75	77.59	86.97	79.93	75.82	83.35	79.36	87.98
CNN	72.51	53.16	74.07	66.76	68.93	68.97	64.24	63.29	76.33	73.28	79.69
CNN-TL	65.4	71.08	50.85	69.59	64.72	72.28	60.57	67.1	70.42	73.93	64.67
NCL	69.64	81.16	39.76	79.51	84.02	85.95	83.28	74.4	85.31	83.28	63.14
RUS	75.77	78.83	82.56	77.1	78.01	82.2	78.06	80.73	80.08	79.32	85.62
No Res.	55.45	55.43	29.29	50.22	65.65	65.39	75.99	65.65	84.34	42.35	47.61

Res. Tech. indicates Resampling Technique, HB Tech. indicates HB technique, GFS-MLB indicates GFS-MaxLB, No Res. indicates No resampling,SMENN indicates SMOTE-ENN

Table 7.6: G-Mean and Balance Results for Lang Dataset

Res.Tech.	HB Tech. G-Mean Values										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	76.86	81.54	77.76	80.37	83.21	80.11	82.04	75.42	76.33	73.41	77.76
BSMOTE	83.35	78.96	78.61	78.78	83.58	78.78	81.68	81.24	76.18	81.46	84.03
ROS	71.71	81.42	78.39	80.96	84.34	78.19	83	78.52	75.31	80.45	83.5
SafeSMOTE	83.74	81.3	80.63	82.44	84.97	79.05	82.65	82.65	78.73	84.7	81.9
SMOTE	84.46	82.53	80.05	80.8	83.98	75.49	85.64	81.01	73.98	81.24	82.44
SMENN	84.22	83.21	81.3	84.47	86.2	75.91	82.89	83.12	77.75	84.93	83.11
SMOTE-TL	82.19	83.51	77.15	78.61	84.47	79.58	82.19	83.12	74.96	84.93	81.66
SPIDER	78.85	84.26	42.92	64.38	85.41	53.15	81.01	81.46	79.09	79.51	57.73
SPIDERII	82.53	86.44	42.92	79.65	85.71	60.95	84.22	78.63	77.38	74.55	63.66
CNN	73.37	84.47	77.97	78.24	69.25	80.3	73.81	78.85	67.75	72.67	78.88
CNN-TL	81.54	71.18	3.53	62.72	60.32	64.98	65.36	72.46	47.49	80.06	45.41
NCL	79.65	81.46	9.6	79.29	84.46	64.74	78.85	78.63	77.3	80.11	54.43
RUS	77.09	85.08	82.18	78.78	84.97	78.78	83.74	80.06	75.88	77.75	82.83
No Res.	76.63	63	0	63.5	65.8	50.52	62.49	72.18	63.5	57.28	54.43
Res.Tech.	**HB Tech. Balance Values**										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	76.17	81.23	76.96	80.35	83.14	80.07	81.62	75.05	75.32	72.81	76.96
BSMOTE	81.15	77.49	78.24	78.76	81.23	78.76	78.73	78.59	74.23	78.66	84
ROS	69.715	81.03	78.39	80.62	83.165	77.45	81.01	75.84	74.15	78.29	83.44
SafeSMOTE	82.81	80.82	80.62	82.42	84.62	78.95	80.86	80.86	77.42	83.35	81.69
SMOTE	83.23	81.81	79.95	80.61	82.96	75.04	83.78	78.51	72.64	78.59	82.28
SMENN	83.1	82.91	81.03	84.23	85.5	75.87	80.96	81.06	74.9	83.47	81.3
SMOTE-TL	80.64	83.72	76.88	78.24	84.23	79.51	80.64	81.06	72.83	83.47	79.53
SPIDER	75.95	81.37	42.38	60.43	83.68	50.13	78.51	78.66	78.63	76.15	52.86
SPIDERII	81.99	85.5	42.38	77.92	85.17	57.64	83.1	75.88	77.2	70.86	58.1
CNN	71.87	84.23	77.78	78.07	68.86	80.17	72.13	77.485	65.75	70.18	78.26
CNN-TL	81.23	68.24	29.45	59	57.5	61.18	64.46	70.01	46.25	79.34	44.79
NCL	77.92	78.59	30.6	76.09	83.23	60.51	75.95	75.88	75.32	78.14	50.24
RUS	76.34	84.82	81.99	78.76	84.62	78.66	82.81	79.34	72.75	75.52	82.07
No Res.	73.38	58.04	29.29	58.09	60.67	47.61	57.99	68.27	58.09	52.85	50.24

Res. Tech. indicates Resampling Technique, HB Tech. indicates HB technique, GFS-MLB indicates GFS-MaxLB, No Res. indicates No resampling,SMENN indicates SMOTE-ENN

Table 7.7: G-Mean and Balance Results for Log4j Dataset

Res.Tech.	HB Tech. G-Mean Values										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	64.79	73.93	76.52	67.58	72.21	70.25	57.39	69.63	71.93	63.28	77.76
BSMOTE	55.13	70.86	73.36	62.52	72.59	72.03	71.87	71.41	76.19	75.09	72.27
ROS	59.45	70.42	74.89	73.45	78.02	74.07	72.73	73.82	76.45	70.83	74.65
SafeSMOTE	69.79	72.58	76.39	71.13	70.29	75.74	73.83	75.15	73.16	70.28	78.55
SMOTE	64.91	68.61	77.21	67.21	65.58	56.2	71.54	68.28	73.36	66.14	75.93
SMENN	60.79	67.72	77.34	65.35	63.05	63.36	74.97	71.05	73.85	67.8	73.35
SMOTE-TL	64.28	74.58	77.74	73.27	71.84	67.8	75.86	70.29	72.07	68.24	77.18
SPIDER	63.05	57.75	7.72	56.85	60.75	56.77	57.41	52.03	53.61	30.43	46.16
SPIDERII	66.49	61.66	7.67	59.73	62.36	46.74	57.68	62.56	65.66	7.72	54.78
CNN	74.67	42.61	37.48	56.69	55.63	60.84	53.84	58.4	56.77	53.29	46.99
CNN-TL	61.29	70.17	72.72	61.79	69.28	64.66	69.22	69.31	68.82	66.54	71.64
NCL	61.85	63.54	7.69	54.16	70.08	51.88	57.77	64.03	74.23	33.44	57.41
RUS	67.95	66.84	72.85	61.68	64.64	70.23	66.17	68.96	71.11	65.09	74.39
No Res.	26.58	30.65	0	50.47	57.34	30.52	40.54	40.14	40.66	26.62	40.71
Res.Tech.	**HB Tech. Balance Values**										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	64.59	73.68	76.53	66.74	72.2	68.85	56.69	69.27	71.15	60.52	76.65
BSMOTE	52.45	68.51	72.77	60.86	71.08	71.3	69.89	70.32	75.84	74.03	71.48
ROS	56.97	70.01	74.72	73.36	77.96	73.12	72.27	73.82	76.38	70.56	74.21
SafeSMOTE	69.48	72.44	76.39	71.09	70.15	75.63	73.75	74.95	72.51	70.27	78.46
SMOTE	63.6	67.23	77.19	64.95	64.03	54.59	70.92	66.33	72.77	66.08	75.63
SMENN	58.44	66.01	77.32	63.17	61.16	61.33	73.95	70.07	73.15	66.69	72.77
SMOTE-TL	62.58	74.39	77.58	73.26	71.53	67.28	75.8	69.91	72.07	68	77.14
SPIDER	60.34	54.3	30.13	52.81	57.45	52.8	54.22	49.36	50.91	35.99	44.44
SPIDERII	63.71	58.81	30.13	57.08	60.02	45.81	55.32	60.12	63.33	30.13	51.13
CNN	71.03	42.66	39.38	52.79	52.59	57.48	50.97	55.61	55.95	50.83	47.14
CNN-TL	61.06	69.8	72.56	60.39	69.03	64.48	68.68	68.3	68.82	66.13	70.48
NCL	58.89	60.54	30.13	51.04	68.11	49.33	55.36	60.72	73.43	37.56	52.85
RUS	67.45	66.64	72.84	61.66	64.59	69.91	65.81	68.96	71.11	63.71	74.01
No Res.	34.34	36.02	29.29	47.77	52.85	36	41.07	41.03	41.07	34.34	41.07

Res. Tech. indicates Resampling Technique, HB Tech. indicates HB technique, GFS-MLB indicates GFS-MaxLB, No Res. indicates No resampling,SMENN indicates SMOTE-ENN

Table 7.8: G-Mean and Balance Results for Ode Dataset

Res.Tech.	HB Tech. G-Mean Values										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	60.7	67.27	72.56	71.02	65.79	70.11	58.73	68.53	70.87	68.72	59.41
BSMOTE	46.54	69.66	70.86	70.08	63.31	71.16	64.72	61.22	72.26	62.21	59.41
ROS	50.94	66.7	72.68	70.54	61.11	68.8	64.94	71.78	72.38	64.99	64.99
SafeSMOTE	69.95	69.46	72.08	71.78	67.77	68.42	68.4	67.69	65.18	72.69	57.35
SMOTE	65.06	71.78	71.79	71.36	69.15	69.34	69.42	72.73	71.11	69.77	63.73
SMENN	67.94	71	72.22	71.32	55.56	69.06	68.87	70.66	70.08	72.27	61.99
SMOTE-TL	66.23	69.06	68.81	68.83	70.3	67.89	67.5	69.39	68.25	70.23	72.17
SPIDER	58.01	53.42	26.49	32.15	47.06	37.33	43.53	43.24	34.85	56.5	43.71
SPIDERII	64.93	60.83	26.49	57.24	49.54	48.51	54.55	59.18	39.09	52.67	55.68
CNN	40.99	52.89	29.6	36.18	31.97	52.13	48.54	45.79	45.61	48.29	49.49
CNN-TL	61.47	55.18	62.69	67.88	56.01	60.1	57.79	65.84	60.68	63.73	68.69
NCL	53.42	53.28	18.7	39.26	55.1	39.52	45.41	48.38	39.4	22.75	49.32
RUS	69.84	67.2	70.38	71.44	63.26	71.3	68.05	72.02	68.96	67.84	60.07
No Res.	37.26	37.35	6.62	29.6	37.33	26.47	37.33	37.4	22.94	0	45.82
Res.Tech.	**HB Tech. Balance Values**										
	DT-GA	**GFS-AB**	**GFS-GCCL**	**GFS-GP**	**GFS-LB**	**GFS-GPG**	**GFS-MLB**	**GFS-SP**	**GP-CAOCH**	**TARGET**	**PSOLDA**
Adasyn	58.12	66.98	72.56	69.88	65.29	68.87	58.47	68.53	70.68	68.16	56.91
BSMOTE	45.295	69.44	70.47	68.85	60.6	70.72	63.09	58.99	72.09	60.08	56.91
ROS	48.86	66.68	72.67	70.49	59.89	68.44	64.57	71.48	72.19	64.39	64.39
SafeSMOTE	69.16	69.45	72.04	71.78	66.83	67.67	67.94	65.88	64.23	71.5	54.7
SMOTE	62.72	71.78	71.76	71.36	68.55	69.33	68.75	71.95	70.89	68.64	60.77
SMENN	66.53	70.81	72.2	71.15	55.135	68.525	68.825	70.305	69.965	71.22	59.35
SMOTE-TL	65.34	68.53	68.02	68.64	70.27	66.43	67.35	69.34	67.65	69.99	71.46
SPIDER	54.92	50.28	34.25	36.72	45.38	39.21	42.92	42.89	37.97	53.66	42.93
SPIDERIII	62.65	57.42	34.25	53.86	47.68	46.575	51.445	56.09	40.41	50.135	51.6
CNN	41.62	50.95	35.49	39.03	36.7	49.98	46.58	45.12	44.17	46.54	46.66
CNN-TL	61.32	54.75	62.42	67.76	56.01	60.01	57.71	65.84	59.43	62	65.72
NCL	50.28	50.26	31.77	40.43	51.54	40.45	44.16	46.55	40.44	33	46.65
RUS	67.45	66.64	72.84	61.66	64.59	69.91	65.81	68.96	71.11	63.71	56.34
No Res.	39.21	39.21	29.29	35.49	39.21	34.25	39.21	39.21	33.01	29.29	44.18

Res. Tech. indicates Resampling Technique, HB Tech. indicates HB technique, GFS-MLB indicates GFS-MaxLB, No Res. indicates No resampling,SMENN indicates SMOTE-ENN

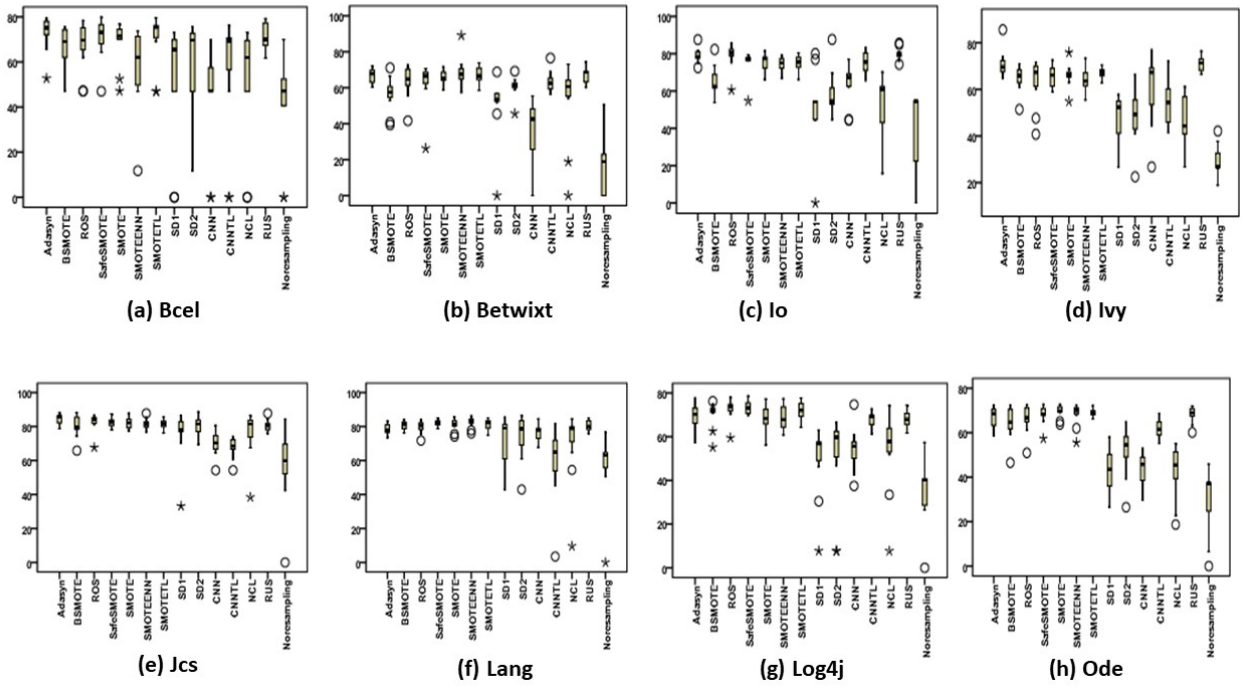


Figure 7.1: Boxplot analysis of G-Mean Results after Applying Data Resampling for Different Datasets (a) Bcel (b) Betwixt (c) IO (d) Ivy (e) JCS (f) Lang (g) Log4j (h) Ode.

In the no resampling situation, for the Ivy dataset (Table 7.4), the average G-Mean values of the SMP models developed using various HB techniques was 30.01, and the average of Balance values just 44.07. For all techniques, the performance of the SMP models on the imbalanced Ivy dataset was inferior in terms of G-Mean and Balance. The G-Mean and Balance values were less than 40 when the models are developed with imbalanced Ivy dataset, and for the GFS-LB technique, the G-Mean and Balance values were just close to 40. When we applied different data resampling techniques before developing the SMP models on the Ivy dataset, in 66.43% of the cases, the G-Mean values were reported to be higher than 60, and the average G-Mean was 60.79. Similarly, the average Balance values of the SMP after data resampling was 59.84, and in 60.13% of the cases, the Balance results were greater than 60. Table 7.5

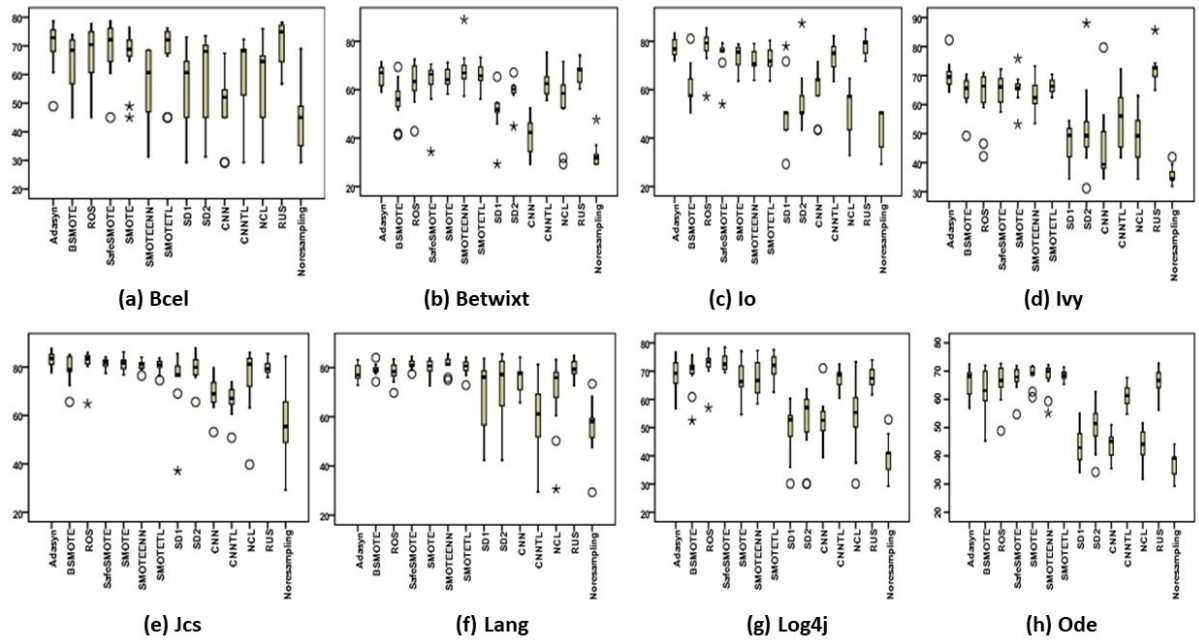


Figure 7.2: Boxplot analysis of Balance Results after Applying Data Resampling for Different Datasets (a) Bcel (b) Betwixt (c) IO (d) Ivy (e) JCS (f) Lang (g) Log4j (h) Ode.

indicates the performance of the SMP models after applying different data resampling techniques and the imbalanced Jcs dataset.

Also, for the Jcs dataset (Table 7.5), applying data resampling techniques improved the performance of the SMP models. For this dataset, the average G-Mean and Balance result after data resampling were 78.86 and 77.97, respectively. After applying data resampling techniques, 97.20% of the cases the G-Mean and Balance values were found to be higher than 60. But on the imbalanced Jcs dataset, the constructed SMP models have shown a poorer performance in terms of both G-Mean and Balance. The average of G-Mean and Balance was 58.00 and 57.94, respectively. In Table 7.6, the predictive performance of the SMP models developed on imbalanced and resampled Lang dataset is recorded. On analyzing the predictive performance of the SMP models recorded in Table 7.6, the average of G-Mean and Balance values was reported to

be 76.73 and 75.68, respectively. For data resampling on the Lang dataset, 76.24% of the SMP models gave G-Mean greater than 60, and 76.24% of the SMP models gave G-Mean greater than 60 and 90.09% of the SMP models gave Balance greater than 60. For this dataset, no resampling case gave the poor performance of the SMP models with average G-Mean and Balance results of 34.93 and 29.45, respectively. Like Betwixt, Io, Ivy, Jcs, and Lang dataset, data resampling techniques have also improved the performance of the SMP models for Log4j, and Ode datasets for the effective prediction of low maintainability classes. The average G-Mean results of the models after applying data resampling techniques were 64.58 and 60.29 for Log4j (Table 7.7) and Ode datasets (Table 7.8), respectively.

In the case of the Log4j dataset, for the no resampling situation, the average G-Mean and Balance performance of the models was 34.93 and 39.43, respectively. In the case of imbalanced Ode, the average G-Mean and Balance results were 28.92 and 36.51, respectively. We also analyzed the results of the SMP models in terms of G-Mean and Balance before and after applying data resampling with the help of boxplots shown in Figure 7.1 and Figure 7.2 respectively. As shown in Figure 7.1(a), the G-Mean values of the SMP models developed after the majority of data resampling techniques reach up to 80 for the Bcel dataset, and the median of G-Mean values reported to be close to 75. The no resampling situation lead to a median of G-Mean slightly above 40. For the Betwixt dataset median of G-Mean values reported to be higher than 60 for the majority of the data resampling techniques, but the median of G-Mean in the imbalanced scenario is very poor as indicated in Figure 7.1(b). The G-Mean results shown in Figure 7.1(c) for the IO dataset have a median G-Mean greater than 75 for the majority of data resampling techniques, and in Figure 7.1(d) Ivy datasets median G-Mean results found to be higher than 65 for most of the examined data resampling techniques. Median G-Mean results were greater than 80 for most of the data resampling techniques for the JCS and Lang datasets. Similarity G-Mean

results of remaining datasets, namely Log4j and ODE, were very good, as shown in Figure 7.1(g) and Figure 7.1(h) when data resampling techniques were used.

According to Figure 7.2(a), the Balance values of the SMP models developed after the majority of data resampling techniques reach up to 75 for the Bcel dataset, and the median of Balance values found to be greater than 70 for most of the data resampling techniques. The no resampling situation in the case on the Bcel dataset has to the median of Balance values slightly above 40. For the Betwixt dataset median of Balance values reported to be greater than 65 for the majority of the data resampling techniques, whereas the median of Balance is less than 20 in no resampling situation shown in Figure 7.2(b). The Balance results are shown in Figure 7.2(c) for the IO dataset have median Balance results greater than 75 for the majority of data resampling techniques. As in Figure 7.2(d), for the Ivy dataset, the median Balance results were observed to be higher than 65 for most of the investigated data resampling techniques. The median of Balance results was higher than 80 for most of the data resampling techniques for Jcs and Lang datasets. Also, the Balance results of Log4j and Ode were very good, as shown in Figure 7.2(g) and Figure 7.2(h) when data resampling techniques were used. Therefore, after analyzing the results of Tables 7.1 to 7.8, and Figures 7.1 and 7.2, we come to the conclusion that the application of data resampling techniques on the imbalanced datasets used in the chapter has improved the performance of the SMP models because the values of both the performance measures, G-Mean and Balance, have increased after applying data resampling.

7.3.2 Results and Analysis of RQ2

As the analysis of results of RQ1 indicates that the performance of the models has improved after data resampling, it is necessary to analyze the results strengthen the results statistically. In this section, the statistical difference in the performance of

the SMP models developed after data resampling methods used in this chapter has been discussed. The results of Section 7.3.2 are evaluated the help of two statistical tests, namely the Freidman test and the Wilcoxon signed-rank test in this section. The following hypothesis is evaluated with the help of the Friedman test.

- *Null hypothesis* $H0_1$: The performance of the SMP models evaluated in terms of G-Mean does not show a significant difference after applying data resampling techniques, namely SMOTE, BSMOTE, SafeSMOTE, Adasyn, ROS, SMOTE-ENN, SMOTE-TL, SPIDER, SPIDER II, RUS, CNN, CNN-TL, NCL.

- *ALternate hypothesis* Ha_1*:* The performance of the SMP models evaluated in terms of G-Mean shows a significant difference after applying data resampling techniques, namely SMOTE, BSMOTE, SafeSMOTE, Adasyn, ROS, SMOTE-ENN, SMOTE-TL, SPIDER, SPIDER II, RUS, CNN, CNN-TL, NCL

- *Null hypothesis* $H0_2$*:* The performance of the SMP models evaluated in terms of Balance does not show a significant difference after applying data resampling techniques, namely SMOTE, BSMOTE, SafeSMOTE, Adasyn, ROS, SMOTE-ENN, SMOTE-TL, SPIDER, SPIDER II, RUS, CNN, CNN-TL, NCL

- *Alternate hypothesis* Ha_2*:* The performance of the SMP models evaluated in terms of Balance shows a significant difference after applying data resampling techniques, namely SMOTE, BSMOTE, SafeSMOTE, Adasyn, ROS, SMOTE-ENN, SMOTE-TL, SPIDER, SPIDER II, RUS, CNN, CNN-TL, NCL.

In order to evaluate the above hypothesis, we apply the Freidman test at a level of significance, α = 0.05, with 13 degrees of freedom (as the performance of 13 data resampling along with one case of no resampling is compared). To do so, we extracted the G-Mean and Balance results of ten runs of the SMP models before and after applying data resampling techniques. The results of the Friedman test analysis are

presented in Table 7.9. The outcome of the Friedman test is the rank assigned to each resampling technique according to its performance. If a technique X obtained a lower rank than that of technique Y, it means X is better than Y.

Table 7.9: Friedman Ranking based on G-Mean and Balance

Based on G-Mean		**Based on Balance**	
Technique	**Mean Rank**	**Technique**	**Mean Rank**
Adasyn	4.72	Adasyn	4.66
SafeSMOTE	4.72	SafeSMOTE	4.66
RUS	5.06	RUS	4.73
SMOTE	5.36	ROS	5.21
ROS	5.41	SMOTE	5.38
SMOTE-TL	5.48	SMOTE-TL	5.45
SMOTE-ENN	5.82	SMOTE-ENN	5.78
BSMOTE	6.89	BSMOTE	6.99
SPIDERII	8.79	CNN-TL	8.57
CNN-TL	8.85	SPIDERII	8.71
NCL	9.73	NCL	9.78
CNN	10.39	SD1	10.66
SPIDER	10.53	CNN	11.07
No resampling	13.25	No resampling	13.33

The p-value obtained after the Friedman test was zero when the test was applied to the G-Mean and Balance results of SMP models constructed after using various data resampling techniques. This means that the outcome of the Friedman test is significant as $p < 0.05$. Therefore, in this case, we rejected $H0_1$ and $H0_2$, which states that all data resampling techniques have the same effect on the performance of the developed SMP models. We, therefore, accepted the alternate hypothesis, Ha_1, and Ha_2. As shown in Table 7.9, the lowest (i.e., best) rank is attained by Adasyn and SafeSMOTE techniques both for G-Mean and Balance. This means these two techniques are equally competent techniques to improve the performance of the SMP models developed from datasets. Therefore, to conclude, we can state that SafeSMOTE and Adasyn are the

best data resampling techniques to build efficient SMP models. It is to be noted that the no resampling case (i.e., the case when the datasets are imbalanced) has attained the worst rank both with respect to G-Mean and Balance. This means that the situation when we develop the SMP models from the imbalanced dataset, the objective of developing the models to predict low maintainability classes accurately is not met even when the models are developed using HB techniques.

As the outcome of the Friedman analysis was significant, we further examined whether Adasyn and SafeSMOTE are significantly better than the rest of the data resampling techniques or not. To investigate this, we used the Wilcoxon signed-rank test and performed a pairwise comparison of the G-Mean and Balance performance of Adasyn and SafeSMOTE techniques with all other examined data resampling techniques. The Wilcoxon signed-rank test is performed with Bonferroni correction to cut down the family-wise error when comparing multiple techniques, i.e., α = 0.05/13 = 0.0038 is taken as a confidence level to compare techniques. The outcome of the Wilcoxon signed-rank test is recorded in Table 7.10 and Table 7.11 for techniques Adasyn and SafeSMOTE, respectively. In Tables 7.10 and 7.11 Sig.+ indicates that there is a significant difference in the performance of a pair of resampling techniques, and NotSig.+ indicated no significant difference. As shown in Table 7.10, in six of the thirteen pairs, Adasyn is not significantly different in terms of its performance both w.r.t. G-Mean and Balance because p-values for these pairs is higher than 0.0038. A similar situation is observed after analyzing the results of the Wilcoxon signed-rank of SafeSMOTE (Table 7.11) with other resampling techniques. Again as per Table 7.11, the SafeSMOTE technique is not significantly different in terms of its performance both w.r.t. G-Mean and Balance from Adasyn, SMOTE, SMOTE-ENN, SMOTE-TL, ROS, and RUS. This leads to the conclusion that ROS, SMOTE, SMOTE-ENN, SMOTE-TL, and RUS are comparable in their performance with Adaysn and SafeSMOTE. These techniques can effectively be applied in case of imbalanced data

scenarios to develop competent SMP models that can predict low maintainability classes accurately.

Table 7.10: Wilcoxon Test Results of Adasyn with all other Resampling Techniques w.r.t G-Mean and Balance

Based on G-Mean			**Based on Balance**		
Adasyn vs.	**p-value**	**Significance**	**Adasyn vs.**	**p-value**	**Significance**
BSMOTE	0.000	Sig.+	BSMOTE	0.000	Sig.+
ROS	0.105	Not Sig.+	ROS	0.271	Not Sig.+
SafeSMOTE	0.359	Not Sig.+	SafeSMOTE	0.692	Not Sig.+
SMOTE	0.032	Not Sig.+	SMOTE	0.028	Not Sig.+
SMOTE-ENN	0.004	Not Sig.+	SMOTE-ENN	0.005	Not Sig.+
SMOTE-TL	0.094	Not Sig.+	SMOTE-TL	0.086	Not Sig.+
SPIDER	0.000	Sig.+	SPIDER	0.000	Sig.+
SPIDERII	0.000	Sig.+	SPIDERII	0.000	Sig.+
CNN	0.000	Sig.+	CNN	0.000	Sig.+
CNN-TL	0.000	Sig.+	CNN-TL	0.000	Sig.+
NCL	0.000	Sig.+	NCL	0.000	Sig.+
RUS	0.745	Not Sig.+	RUS	0.691	Not Sig.+
No resampling	0.000	Sig.+	No resampling	0.000	Sig.+

Table 7.11: Wilcoxon Test Results of SafeSMOTE with all other Resampling Techniques w.r.t G-Mean and Balance

Based on G-Mean			**Based on Balance**		
SafeSMOTE vs.	**p-value**	**Significance**	**SafeSMOTE vs.**	**p-value**	**Significance**
Adasyn	0.359	Not Sig.+	Adasyn	0.58	Not Sig.+
BSMOTE	0.000	Sig.+	BSMOTE	0.000	Sig.+
ROS	0.460	Not. Sig.+	ROS	0.468	Not. Sig.+
SMOTE	0.055	Not. Sig.+	SMOTE	0.035	Not. Sig.+
SMOTE-ENN	0.002	Sig.+	SMOTE-ENN	0.001	Sig.+
SMOTE-TL	0.314	Not. Sig.+	SMOTE-TL	0.215	Not. Sig.+
SPIDER	0.000	Sig.+	SPIDER	0.000	Sig.+
SPIDERII2	0.000	Sig.+	SPIDERII	0.000	Sig.+
CNN	0.000	Sig.+	CNN	0.000	Sig.+
CNN-TL	0.000	Sig.+	CNN-TL	0.000	Sig.+
NCL	0.000	Sig.+	NCL	0.000	Sig.+
RUS	0.777	Not. Sig.+	RUS	0.792	Not. Sig.+
No resampling	0.000	Sig.+	No resampling	0.000	Sig.+

7.3.3 Results and Analysis of RQ3

The second aspect of this study was to examine the best HB technique to develop SMP models. For this examination, we evaluated the performance of the SMP models developed with different HB techniques for all eight datasets used in the study. For each dataset, after applying a data resampling technique, we run each HB technique ten times. The median of G-Mean and Balance performance of the models after ten runs were recorded, which we analyzed further to determine the best HB technique. We analyzed the G-Mean and Balance results with the help of the Friedman and the Wilcoxon signed-rank test. The following hypothesis was evaluated using the Friedman test.

- *Null hypothesis* $H0_3$: The G-Mean performance of SMP models developed using HB techniques, DT-GA, GFS-AB, GFS-GCCL, GFS-GP, GFS-LB, GFS-GPG, GFS-maxLB, GFS-SP, GP-COACH, TARGET, PSOLDA does not show a significant difference.

- *Alternate hypothesis* Ha_3*:* The G-Mean performance of SMP models developed using HB techniques, DT-GA, GFS-AB, GFS-GCCL, GFS-GP, GFS-LB, GFS-GPG, GFS-maxLB, GFS-SP, GP-COACH, TARGET, PSOLDA have a significant difference.

- *Null hypothesis* $H0_4$*:* The Balance performance of SMP models developed using HB techniques, DT-GA, GFS-AB, GFS-GCCL, GFS-GP, GFS-LB, GFS-GPG, GFS-maxLB, GFS-SP, GP-COACH, TARGET, PSOLDA does not show a significant difference.

- *Alternate hypothesis* Ha_4*:* The Balance performance of SMP models developed using HB techniques, DT-GA, GFS-AB, GFS-GCCL, GFS-GP, GFS-LB,

GFS-GPG, GFS-maxLB, GFS-SP, GP-COACH, TARGET, PSOLDA have a significant difference.

We applied the Freidman test at a level of significance, α = 0.05, with 10 degrees of freedom (as the performance of 11 data HB techniques is compared). The results of the Friedman test analysis are depicted in Table 7.12.

Table 7.12: Friedman ranking of HB techniques

Based on G-Mean		**Based on Balance**	
Technique	**Mean Rank**	**Technique**	**Mean Rank**
GFS-LB	4.97	GFS-LB	4.92
PSOLDA	4.99	PSOLDA	4.94
GFS-AB	5.35	GFSAB	5.26
GFS-SP	5.42	GFS-SP	5.51
GP-COACH	5.85	GFS-MxaLB	6.02
GFS-MaxLB	5.86	GP-COACH	6.02
GFS-GPG	6.16	GFS-GPG	6.07
GFS-GP	6.4	GFS-GP	6.39
GFS-GCCL	6.86	TARGET	6.8
TARGET	7.03	GFS-GCCL	6.89
DT-GA	7.12	DT-GA	7.17

The p-value was 0.000 both for G-Mean and Balance for the Friedman test. As p-value obtained was less than 0.05, we rejected the null hypothesis $H0_3$ and $H0_4$ that stated the existence of no significant difference in the performance of the SMP models developed after applying different HB techniques. We, therefore, accepted the alternate hypothesis and concluded that the performance of SMP models is different in term of G-Mean and Balance with different HB technique. The GFS-LB technique has attained the best rank w.r.t. G-Mean and Balance. The second best rank is assigned to PSOLDA techniques. The worst rank is assigned to DT-GA with respect to both of the performance measures. We further investigated that the performance of the best two

HB techniques (GFS-LB and PSOLDA) has a significant difference from the rest of the examined techniques or not. To do so, we conducted a post-hoc comparison with the Wilcoxon signed-rank test first of GFS-LB with all other investigated techniques and then of PSOLDA with all other investigated techniques. The post-hoc investigation with the Wilcoxon signed-rank test was carried out at significance level of α = 0.05 with Bonferroni correction (i.e., α = 0.05/10 = 0.005) and 10 degree of freedom. Table 7.13 presents the results of the Wilcoxon signed-rank test of pairwise comparison of GFS-LB and PSOLDA with all other HB techniques with respect to G-Mean and Balance.

Table 7.13: Wilcoxon Test Results of SafeSMOTE with all other Resampling Techniques w.r.t. G-Mean and Balance

Based on G-Mean			**Based on Balance**		
PSOLDA vs.	***p-value***	***Significance***	***PSOLDA vs.***	***p-value***	***Significance***
DT-GA	0.000	Sig.+	GFS-GCCL	0.000	Sig.+
GFS-AB	0 0.00	Sig.+	DT-GA	0.002	Sig.+
GFS-GCCL	0.006	Not Sig.+	TARGET	0.002	Sig.+
GFS-GP	0.007	Not Sig.+	GFS-GPG	0.007	Not Sig.+
GFS-LB	0.011	Not Sig.+	GFS-GP	0.009	Not Sig.+
GFS-GPG	0.146	Not Sig.+	GP-COACH	0.066	Not Sig.+
GFS-MaxLB	0.592	Not Sig.+	GFS-MaxLB	0.157	Not Sig.+
GFS-SP	0.673	Not Sig.+	GFS-SP	0.72	Not Sig.+
GP-COACH	0.833	Not Sig.+	GFS-AB	0.893	Not Sig.+
TARGET	0.005	Sig.+	GFS-LB	0.99	Not Sig.+
Based on G-Mean			**Based on Balance**		
GFS-LB vs.	***p-value***	***Significance***	***GFS-LB vs.***	***p-value***	***Significance***
DT-GA	0.000	Sig.+	GFS-GCCL	0.000	Sig.+
GFS-GCCL	0.000	Sig.+	GFS-GP	0.000	Sig.+
GFS-GP	0.000	Sig.+	TARGET	0.000	Sig.+
TARGET	0.000	Sig.+	DT-GA	0.000	Sig.+
GFS-GPG	0.006	Not Sig.+	GFS-GPG	0.008	Not Sig.+
GFS-MaxLB	0.033	Not Sig.+	GFS-MaxLB	0.01	Not Sig.+
GP-COACH	0.048	Not Sig.+	GP-COACH	0.012	Not Sig.+

GFS-AB	0.543	Not Sig.+	GFS-SP	0.472	Not Sig.+
GFS-SP	0.545	Not Sig.+	GFS-AB	0.515	Not Sig.+
PSOLDA	0.673	Not Sig.+	PSOLDA	0.99	Not Sig.+

As shown in Table 7.13, the PSOLDA technique is significantly better (p-value $<$ 0.005) than DT-GA, GFS-AB, and TARGET in terms of its performance with respect to G-Mean. Also, with respect to Balance, PSOLDA is significantly better (p-value $<$ 0.005) performing technique than DT-GA, GFS-GFS-GCCL, and TARGET. However, for the performance of PSOLDA is better but not significantly than GFS-AB, GFS-GP, GFS-LB, GFS-GPG, GFS-maxLB, GFS-SP, and GP-COACH with respect to Balance. On comparing the relative performance of GFS-LB with all other techniques according to the results of the Wilcoxon signed-rank test with respect to G-Mean and Balance, it is to be noted that GFS-LB is significantly better than DT-GA, GFS-GCCL, GFS-GP, and TARGET in terms of its performance with respect to G-Mean and Balance as the obtained p-values are less than 0.005. However, the performance of GFS-LB is better but not significantly than GFS-GPG, GFS-MaxLB, GP-COACH, GFS-SP, GFS-AB, and PSOLDA with respect to G-Mean and Balance.

To conclude, we say that PSOLDA and GFS-LB are the best techniques for developing SMP models in order to correctly predict low maintainability classes. The performance of the other techniques except DT-GA, TARGET, GFS-AB, and GFS-GCCL is comparable with PSOLDA. The performance of GFS-GPG, GFS-MaxLB, GP-COACH, GFS-SP, GFS-AB is comparable with GFS-LB. In the GFS-LB technique, the logit boost boosting mechanism boosts the performance of fuzzy classifiers, and GA helps to form the optimal rule set with the aim of enhancing the accuracy of the fuzzy classifier. The PSOLDA technique that the hybridization of PSO and LDA uses PSO for the optimal selecting set of features that are then presented to the LDA classifier. Since the LDA is a competent classification technique, using PSO

with LDA further enhances the classification capability of LDA.

7.4 Discussion

The chapter assessed the predictive capability of the HB techniques for developing efficient SMP models by treating the imbalanced data. The results of the chapter advocate that the use of data resampling improved the performance of the SMP models.The SafeSMOTE and Adasyn are reported to be the the best techniques to handle imbalanced data and enhance the performance of the SMP models. Also, the performance of ROS, SMOTE, SMOTE-ENN, SMOTE-TL, and RUS is found comparable to Adasyn and SafeSMOTE both with respect to G-Mean and Balance as per post-hoc analysis carried out with the help of the Wilcoxon signed-rank test.

The experimental set up of this chapter took into account the non-deterministic nature of HB techniques by performing ten runs of each HB technique to develop SMP models. GFS-LB and PSOLDA were observed as the best performing HB techniques after data resampling. The performance of SMP models developed using GFS-LB and PSOLDA gave median G-Mean of 70.77 and 70.22 respectively. The median Balance values of SMP models developed using GFS-LB and PSOLDA reported to be 70.00 and 70.47 respectively.

Chapter 8

Modified Safe Level Synthetic Minority Oversampling Technique for Handling Imbalanced Data in Software Maintainability Prediction

8.1 Introduction

Numerous studies in the literature [9, 13, 56, 168] and previous chapters have ascertained the application of several categories of techniques for developing prediction models to determine the maintainability software classes on different datasets. The results to the previous chapters advocate that in order to develop a useful maintainability prediction model, it is vital to have the efficient training datasets, which contains a sufficient number of data points corresponding to low maintainability and high maintainability classes so that the prediction model can be trained effectively and

identify the maintainability of the future classes.

Numerous strategies have been proposed to mitigate the problem of imbalanced in which data resampling techniques have been much abundantly advocated. The results of the previous chapter also confirm that data resampling improved the performance of the SMP models learned from imbalanced datasets. Oversampling and undersampling based data resampling techniques are investigated in the previous chapters. Oversampling techniques work by boosting the minority class's strength by adding more data points into that class, whereas undersampling techniques work by reducing the strength of the majority class by randomly discarding that results in data loss. Consider a training dataset with 10000 data points in which 500 data points belong to the minority class, and 9500 data points are of the majority class. To achieve perfect balance with undersampling, resulting data would contain only 1000 data points (500 minority and 500 majority data points). In this way, 90% of the training data (i.e., 9000 data points) would be lost. However, with oversampling, there will be no such loss of data and it would result in 20000 data points (10000 minority and 10000 majority). The training dataset after data resampling with oversampling either have synthetic or duplicated minority class data points.

In oversampling techniques, SMOTE [28] and its variants like BSMOTE [186], SafeSMOTE [187] have been widely used in the literature and also validated in previous chapters. SMOTE blindly creates the synthetic data points for each minority class data points. Han et al. [186] revised the SMOTE and proposed BSMOTE, which preprocesses the training data to generate synthetic samples corresponding to borderline minority class data points. Bunkhumpornpat et al. [187] proposed SafeSMOTE, another modification of SMOTE that generates synthetic data points only corresponding to safe minority data points as discussed in Chapter 4. In an imbalanced data, every minority data point is important, however, the data resampling techniques like BSMOTE and SafeSMOTE oversample only a few of the minority

data points BSMOTE oversamples only the borderline data points, and SafeSMOTE oversamples minority data points in accordance with their Safe-Levels. This would result in decreasing the minority class recognition rate of the model.

To contribute to handle the imbalanced data problem in SMP and to use it to improve the predictions of maintainability prediction models, this chapter proposes a novel oversampling technique MSLSMOTE. The proposed technique is a modification of SafeSMOTE and generates synthetic data points carefully. The MSLSMOTE finds the $safe$ and $noisy$ minority class data points using a weighted KNN algorithm and then carefully creates the synthetic data points both corresponding to $safe$ and $noisy$ minority class data points very precisely according to their $\mathrm{Safe\text{-}Levels}$.

MSLSMOTE is proposed as a modification of one of the variants of SMOTE (i.e., SafeSMOTE). This chapter presents an extensive empirical investigation comparing the performance of MSLSMOTE with SMOTE and its two variants SafeSMOTE and BSMOTE at different rates of oversampling. Using eleven different ML techniques and three different performance metrics, the performance of MSLSMOTE, SMOTE, BSMOTE, and SafeSMOTE on eight imbalanced datasets in the domain of software maintainability is assessed. Precisely the following research questions are addressed in this chapter.

- RQ1) At different oversampling rates, what is the performance of MSLSMOTE, SMOTE, BSMOTE, and SafeSMOTE for predicting software maintainability?
- RQ2) Does MSLSMOTE significantly improves the performance of SMP models as compare to SMOTE, BSMOTE, and SafeSMOTE?

The above RQs are empirically evaluated using eight open-source datasets. ML techniques used in this chapter are AdaBoost, Bagging, C4.5, RIPPER, SLIPPER, KNN, KSTAR, LR, BNGE, EACH, and RISE. The chapter is organized in the following manner: Section 8.2 explains the proposed MSLSMOTE technique along with

its pseudocode. Section 8.3 states the experimental framework which includes the datasets used, performance measures, and other design considerations of the chapter. Section 8.4 states the results of the investigated RQs and analyzes them. In section 8.5 a comparison of the previous chapter results and the results of literature studies is presented. Finally, Section 8.6 summarizes the findings of the chapter. The results of this chapter are reported in [224].

8.2 The Proposed MSLSMOTE Technique

Bunkhumpornpat et al. [187], modified the SMOTE and came up with SafeSMOTE. As BSMOTE synthesizes only the borderline minority data points as discussed in Chapter 4, the resultant dataset after oversampling still may contain too few data points of the minority class compared to the majority class. Instead of only oversampling the borderline minority data points, SafeSMOTE oversamples the minority class data point D_i in accordance with its $\mathrm{Safe\text{-}Level}$ ratio. For a minority data point D_i $\mathrm{Safe\text{-}Level}$ ratio is given as a ratio of $\mathrm{Safe\text{-}Level}$ of D_i to that nearest neighbor's Safe-Level. The $\mathrm{Safe\text{-}Level}$ of a data point is given as the number of minority class data points in its KNN. The focus of the SafeSMOTE technique is to generate synthetic data points in safe regions.

Generally, the isolated data points whose surrounding data points belong to a different class and located far away from a group of the same class data points are treated as noise. Many of the classification techniques overlook the noisy data points in the classification process. The mentioned noisy data points can be either from the positive (minority) class or negative (majority) class, but only noisy data points from the minority class are treated as minority outcast data points. In an imbalanced dataset, each minority class data point is rare and vital, eliminating these minority outcast data points decreases the model's positive recognition rate.

SafeSMOTE discards the minority outcast data point during the synthesizing. After synthesizing, the resulting dataset decreases the positive recognition rate of the model that leads the algorithm to attain the lower recall. This chapter proposes an oversampling technique, MSLSMOTE that modifies the SafeSMOTE in such a way that there are no minority outcasting. Considering each minority class data point is rare and vital and its elimination can decrease the model's positive recognition rate, MSLSMOTE does not discard these data points during synthesizing. It rather generates synthetic data points very carefully surrounding such data points as per their $\mathrm{Safe\text{-}Level}$ ratio. In MSLSMOTE, we distinguish between the two types of data points from the minority class $safe$ and $noisy$.

The first function takes a minority class data point, $i \in D_{min}$ using weighted KNN and returns the classification status. The KNN uses a majority voting approach for classification in which every neighbor has the same impact on the classification. However, the closest neighbors have more impact on classifying a data point than the distant neighbors. The proposed technique MSLSMOTE uses weighted KNN that weights each nearest neighbor according to its distance. The pseudo-code of the MSLSMOTE is given in Figure 8.1.

In Figure 8.1, D_{min} denotes the minority class and D_{maj} denotes the majority class. The correctly classified minority class data points by the weighted-KNN are considered $safe$, whereas incorrectly classified minority class data points are considered $noisy$. The minority class data points are then oversampled considering whether they are $safe$ or $noisy$. All the noisy data from the minority class are oversampled more carefully. We use two functions: $\mathrm{Classify\text{-}wKNN(i, k)}$ and $\mathrm{Compute\text{-}KNN(i, k)}$.

The second function determines the KNN of i from $D_{min} \cup D_{maj}$. In the pseudo-code shown in Figure 8.1, various variables are described as follows: T denotes the original training dataset, T' indicates the dataset after oversampling. i is a data point in the set of minority class data points D_{min}. A synthetic data point generated by the

```
T← Original dataset
T´← Dataset after oversampling
Dmin ← set of all datapoints in T that belong to the minority class
Dmaj ← set of all samples in T that belong to the majority class

1. T´ = T
2. for each i ϵ Dmin
   if Classify-wkNN (i, k)==correct
   flag(i) ← safe
   else
   flag(i) ← noisy

3. for each i ϵ Dmin && flag(i)==safe
   s ← generate-datapoint (SMOTE, i, k)
   T´ ← T´ ∪ s
   return T´

4. for each i ϵ Dmin && flag(i)==noisy
   n ← Compute-kNN (i , k)
   Safe-Level(i) ← number of datapoints that belong to Dmin from kNN of i in T
   Safe-Level(n) ← number of datapoints that belong to Dmin from kNN of n in T
   Safe-Ratio = Safe-Level(i)/Safe-Level(n)

5. if Safe-Level(i)==0 && Safe-Level(n)==0
   do not generate synthetic datapoints corresponding to i
   elseif Safe-Level(i)! =0 && Safe-Level(n)=0
      i ← replicate(i)
      T´= T´ ∪ i
    elseif Safe-Level(i)==Safe-Level(n)
      r←generate_random_number (0,1)
      d←n-i
      s←i + r * d
      T´= T´ ∪ s
    elseif Safe-Level(i) > Safe-Level(n)
      r←generate_random_number(0,1/Safe-Ratio)
      d← n - i
      s←i + r * d
      T´= T´ ∪ s
    else
      r←generate_random_number (1-Safe-Ratio, 1)
      d←n - i
      s←i + r * d
      T´= T´ ∪ s
```

Figure 8.1: MSLSMOTE Pseudocode

algorithm is denoted by s. The nearest neighbor of i is denoted by n. r is a random numeral between 0 and 1. All the minority data points that are flagged as $safe$ in step 2 are oversampled using SMOTE and the generated synthetic data points are added in T'. The steps 4 onwards describe how data points flagged as $noisy$ are oversampled. In step 4, the function $\mathrm{Compute\text{-}KNN(i, k)}$ determines the nearest neighbors of i that are flagged as noisy. Also, the $\mathrm{Safe\text{-}Level}$ of each data point i is computed and denoted as $\mathrm{Safe\text{-}Level(i)}$.

Step 1 describes that at the beginning of the algorithm, T' contains original training data. In step 2, we classify each minority class data points into $safe$ and $noisy$ using weighted KNN. All the minority data points that are flagged as $safe$ in step 2 are oversampled with the help of SMOTE and all the generated synthetic data points are added in the set T' as shown in step 3. The steps 4 onwards describe how data points flagged as noisy are oversampled. In step 4, the function $\mathrm{Compute\text{-}KNN(i, k)}$ determined the nearest neighbors of i that is flagged as noisy. Also, the $\mathrm{Safe\text{-}Level}$ of each data point i and it is denoted as $\mathrm{Safe\text{-}Level(i)}$. Similarly, $\mathrm{Safe\text{-}Level}$ of each of the nearest neighbor n of i $\mathrm{Safe\text{-}Level(n)}$ is computed. The $\mathrm{Safe\text{-}Level}$ of a data point $i \in D_{min}$ is defined as the number of data points belong to minority class in KNN of in the whole training data. In step 4, $\mathrm{Safe\text{-}Ratio}$ is computed as the ratio of $\mathrm{Safe\text{-}Level(i)}$ to that of $\mathrm{Safe\text{-}Level(n)}$. The $\mathrm{Safe\text{-}Ratio}$ would be helpful in the generation of synthetic samples in step 5. The generation of synthetic samples using $\mathrm{Safe\text{-}Level}$ and $\mathrm{Safe\text{-}Ratio}$ is similar to SafeSMOTE. In step 5, if the $\mathrm{Safe\text{-}Level}$ of both i and n is zero, no synthetic sample is generated corresponding to i.

If the $\mathrm{Safe\text{-}Level}$ of i is not zero but the $\mathrm{Safe\text{-}Level}$ of n is zero, n is considered as $noise$ and i is oversampled by merely creating a copy of it and the duplicated data point is added in T'. If the $\mathrm{Safe\text{-}Level}$ of i is the same as that of n, the synthetic data points corresponding to i is created by interpolating i and n i.e., using the procedure adopted by SMOTE. In the next case, if the $\mathrm{Safe\text{-}Level}$ of i is greater than that of

$\mathrm{Safe\text{-}Level}$ of n, the synthetic data point is created close to i rather than n. For this, the random number is generated between $[0, 1/\mathrm{Safe\text{-}Ratio}]$ and synthetic data point is generated by interpolating i and n. For the last case, if the $\mathrm{Safe\text{-}Level}$ of i is less than that of the $\mathrm{Safe\text{-}Level}$ of n, in this case, the synthetic data point is created nearest to n rather than i. For generating the synthetic data point close to n, the random number is generated between $[1\text{-}\mathrm{Safe\text{-}Ratio}, 1]$ and i and n are interpolated taking the random number. The above procedure is repeated for each $i \in D_{min}$. When the algorithm terminates, the training dataset would have the original data in addition to the synthetic data generated by the above procedure.

8.3 Research Methodology

This section states the research methodology followed in this chapter.

8.3.1 Datasets and Variables

In this chapter eight Apache open-source software datasets have been used for empirical validation. These datasets have been validated in Chapters 4 to 6. Each dataset contains eighteen OO metrics described in Chapter 2. In this chapter, the best predictors are selected using the CFS method. The selected predictors corresponding to each dataset are given in Chapter 6 (Section 6.3.1). The dependent variable in this chapter is "maintainability".

8.3.2 Model Development and Evaluation

In this chapter, each dataset is oversampled at different oversampling rates with proposed MSLSMOTE, SafeSmote, BSMOTE, and SMOTE. The different oversampling rates were 100%, 200%, 300%, 400%, and 500%. The resampled dataset is then

divided into ten partitions and SMP models are developed using ML techniques given Section 8.1. For training the models, nine partitions are utilized after that for validating the developed model, the tenth partition is used. To allow each partition to participate in training and validation, the complete process is carried out ten times. The performance metrics AUC, G-Mean, and Balance are used to evaluate the models. The Friedman test and Wilcoxon signed-rank test is used for statistical evaluation of the developed models.

8.4 Results and Analysis

This section presents the results and analysis of answers to the research questions.

8.4.1 Results and Analysis of RQ1

The aim of the chapter is to develop SMP models to predict the low maintainability software classes accurately. After data resampling, the maintainability prediction models are developed with eleven ML techniques. The performance evaluators, Balance, AUC, and G-Mean are used to assess the performance of models. In the first experiment, the predictive performance of the maintainability prediction models on original imbalanced data is observed. The original imbalanced datasets are taken, and after learning the data through the classification methods, the SMP models are developed. In Table 8.1, for ML technique on all eight datasets, the average performance in AUC, G-Mean, and Balance is reported.

Table 8.1: Results on Imbalanced Data

ML Technique	**AUC**	**G-Mean**	**Balance**
Adaboost	52.68	52.68	52.68
Bagging	47.14	47.14	47.14
C4.5	42.72	42.72	42.72

RIPPER	62.85	62.85	62.85
SLIPPER	53.20	53.20	53.20
KNN	52.08	52.08	52.08
KSTAR	43.24	43.24	43.24
LR	44.44	44.44	44.44
BNGE	49.71	49.71	49.71
EACH	50.64	50.64	50.64
RISE	50.92	50.92	50.92

For experimental simulation with learning techniques, the KEEL tool is used with the default parameter setting of different techniques. As shown in Chapter 5 and in Table 8.1, on the imbalanced data, the model's performance in terms of AUC, G-Mean, and Balance is not appropriate. In terms of AUC, except the RIPPER, the average AUC for all other ML techniques are close to less or than 0.50. Similar is the performance of the maintainability prediction models in terms of G-Mean and Balance in the imbalanced scenario, i.e., except RIPPER for no other learning technique, the models could achieve G-Mean or Balance of even 60%. As in the imbalanced datasets, there are skewed distributions of the low maintainability and high maintainability data points; the prediction models resulted in a very low positive rate. That resulted in the poor performance of the models in terms of AUC, G-Mean, and Balance. In the second phase of the experiment, the performance of SMP models developed after data resampling with MSLSMOTE, SafeSMOTE, BSMOTE, and SMOTE is compared. All tested resampling methods are combined with eleven ML techniques and SMP models are developed. To determine the best degree of oversampling of the data resampling methods, the different oversampling rates are tried for each method from 100% to 500%. The different oversampling rates for the data resampling techniques have been tested because the imbalanced datasets may have different skewness levels. In this experiment, all eight datasets used in the chapter are oversampled at different rates using MSLSMOTE, SLSMOTE, BSMOTE, and SMOTE. After oversampling

a dataset with the different oversampling techniques at an oversampling rate, ML techniques have been applied to develop the prediction models. The oversampling rate depends on the selection of the nearest neighbors n out of k of a minority class data point. For instance, in the case of generation of synthetic data points using 100% oversampling, one nearest neighbor n, out of k is picked that participates in generating synthetic data points. Similarly, for an oversampling rate of 200%, two neighbors out of k are picked, and so on. For instance, in the case of SMOTE at an oversampling rate of 100%, corresponding to each minority class instance, a synthetic data point is generated, and 100% oversampling doubles the strength of the minority class. In this chapter, the nearest neighbors were determined with k = 5. All the experiments were developed after the ten-fold cross-validation. After different oversampling rates, during the experiments, sensitivity, and specificity values are determined and values of Balance, G-Mean and AUC are computed. The average AUC, Balance, and G-Mean values for all eight datasets at a different oversampling rate (OR) are shown in Table 8.2, 8.3 and 8.4 respectively. For a straightforward interpretation of the results, the best outcome for each oversampling technique is highlighted in bold. As shown in Table 8.2, at OR=100%, the average AUC of SMP models for oversampling with MSLSMOTE range from 0.62 to 0.75 whereas, for SafeSMOTE, BSMOTE, and SMOTE, the average AUC ranged from 0.60 to 0.69, 0.58 to 0.69, and 0.59 to 0.68 respectively. At OR=200%, the average AUC of SMP models for oversampling with MSLSMOTE range from 0.62 to 0.78, whereas for SafeSMOTE, BSMOTE, and SMOTE, the average AUC ranged from 0.60 to 0.68, 0.59 to 0.69, and 0.59 to 0.68 respectively. The average AUC ranged for MSLSMOTE is 0.62 to 0.78, whereas, for SafeSMOTE, BSMOTE, and SMOTE, the average AUC ranged from 0.60 to 0.68, 0.59 to 0.70 and 0.59 to 0.69 respectively at OR= 300%. On OR=400% and 500%, for MSLSMOTE, the average AUC range was observed to be 0.58 to 0.78. The performance of the prediction models using MSLSMOTE does

not improve beyond OR=400%. For SLSMOTE, at OR=400% and 500%, the average AUC range was observed to be 0.60 to 0.69 and 0.61 to 0.69, respectively, and it was observed that SafeSMOTE was the second-best performer after MSLSMOTE in terms of average AUC at OR=400% and 500%. The average Balance for all datasets at different oversampling rates is recorded in Table 8.3. As evident from Table 8.3, at the low rate of oversampling, the performance of SMP models in terms of Balance is not satisfactory. Except for the RIPPER for all other ML techniques and MSLSMOTE combinations, the average Balance over all datasets reported less than 65%. At OR=100%, for SLMSOTE, BSMOTE, and SMOTE, the average Balance values are reported being very poor i.e., less than 60% for all ML techniques. At OR=200%, for MSLSMOTE and different ML techniques combinations, the performance of models has been improved in terms of Balance. Unlike MSLSMOTE, however, this trend is not evident for SafeSMOTE, BSMOTE, and SMOTE. At a further increased rate of oversampling, i.e., at OR=300%, 400%, and 500% of MSLSMOTE, prediction models developed with different ML techniques showed improved Balance results. As shown in Table 8.3, at OR=300%, for six out of eleven MSLSMOTE and ML techniquesâ combinations, the average Balance is higher than 70%. For OR=400%, further, improvement has been observed in the case of MSLMOTE for all classifiers, i.e., at this rate of oversampling, for eight of the eleven classifiers, the average Balance is higher than 70%, and we can see from Table 8.3, that at OR=500%, there is saturation, the average Balance has not increased. However, unlike MSLSMOTE, similar trends are not evident for the other data resampling techniques used in the chapter. For SafeSMOTE, the best results have been reported at OR=500%, but the average Balance range reported being quite low, i.e., the lowest average Balance was 46.79% and the highest 66.46%. Like SafeSMOTE, the performance of other resampling methods, BSMOTE and SMOTE, were highest at OR=500%, but the range was quite inferior. The average Balance values for BSMOTE and SMOTE at OR=500%

ranged from 45.35 to 62.51% and 47.28 to 64.595, respectively. The performance of SMP models at different rates of oversampling analyzed in terms of G-Mean is given in Table 8.4. At the lowest rate of oversampling, SMP models' performance in terms of G-Mean is reported to be very poor. Except for the RIPPER, SLIPPER, and KNN, for all other classifiers and MSLSMOTE combinations, the average G-Mean was less than 65%. Similar to analysis in terms of Balance, at OR=100%, for SafeSMOTE, BSMOTE, and SMOTE and different classifiers combinations, the average G-Mean values were very poor and less than 60% for the majority of the cases. At OR=200%, for MSLSMOTE and different classifiers combinations, SMP models' performance has shown improvement in terms of G-Mean also. But like MSLSMOTE, there are no significant improvements in average G-Mean values for SafeSMOTE, BSMOTE, and SMOTE at OR=200%. At a further increased rate of oversampling, i.e., at OR=300%, 400%, and 500% of MSLSMOTE, prediction models developed with different classifiers showed improved results in terms of G-Mean. As shown in Table 8.4, at OR=300%, for seven out of eleven MSLSMOTE and classifiers combinations, average G-Mean values are higher than 70%. For OR=400%, further, improvement has been observed in the case of MSLMOTE for all classifiers and eight of the eleven classifiers, the average G-Mean values reported greater than 70%. Like the Balance results, at OR=500%, there is saturation, the average G-Mean has not increased further. Unlike MSLSMOTE, the other data resampling techniques used in the chapter have not shown such performance in terms of G-Mean. For SafeSMOTE, the best results have been reported at OR=500%, but the range of average G-Mean was very low, i.e., the lowest average G-Mean was 46.61% and the highest being 67.94%. Similar to SLSMOTE, the performance of other resampling methods, BSMOTE and SMOTE, was highest at OR=500%, but the range was quite lower. The average Balance values for BSMOTE and SMOTE at OR=500% ranged from 41.45 to 64.35% and 43.23 to 66.29%. The experimental results depicted in Tables 8.2 to 8.4 are summarised in

Figure 8.2.

The x-axis in Figure 8.2 signifies the different oversampling rate of the minority class, and the y-axis symbolizes the performance evaluators; AUC, G-Mean, and Balance in sequence from Figure 8.2(a)-8.2(c). In Figure 8.2, SLSMOTE stands for SafeSMOTE. Analysis of Figure 8.2 indicates that MSLSMOTE achieves the highest performance on AUC, G-Mean, and Balance. Also, it is apparent for MSLSMOTE, AUC, G-Mean, and Balance are better with an increase in the rate of oversampling on the minority class. But, improved performance with an increase in the rate of oversampling on the minority class is not much prevalent for the other resampling techniques used in the chapter.

Table 8.2: AUC Result of SMP Models

ML Technique	**MSLSMOTE**	**SafeSMOTE**	**BSMOTE**	**SMOTE**
OR=100%				
AdaBoost	**0.69**	0.64	0.65	0.65
Bagging	**0.7**	0.63	0.62	0.62
C4.5	**0.68**	0.6	0.61	0.6
RIPPER	**0.75**	0.69	0.69	0.69
SLIPPER	**0.70**	0.66	0.66	0.64
KNN	**0.71**	0.64	0.64	0.64
KSTAR	**0.64**	0.60	0.58	0.59
LR	**0.66**	0.62	0.62	0.62
BNGE	**0.70**	0.62	0.62	0.62
EACH	**0.62**	0.62	0.62	0.61
RISE	**0.68**	0.66	0.63	0.64
OR=200%				
AdaBoost	**0.73**	0.64	0.65	0.65
Bagging	**0.74**	0.63	0.64	0.63
C4.5	**0.73**	0.61	0.6	0.62
RIPPER	**0.78**	0.68	0.68	0.68
SLIPPER	**0.75**	0.67	0.66	0.64
KNN	**0.74**	0.64	0.65	0.64
KSTAR	**0.66**	0.60	0.59	0.59
LR	**0.69**	0.63	0.63	0.63

BNGE	**0.74**	0.62	0.62	0.62
EACH	**0.62**	0.62	0.61	0.61
RISE	**0.73**	0.66	0.64	0.64
OR=300%				
AdaBoost	**0.76**	0.64	0.63	0.63
Bagging	**0.76**	0.63	0.65	0.64
C4.5	**0.77**	0.63	0.61	0.62
RIPPER	**0.78**	0.68	0.7	0.69
SLIPPER	**0.77**	0.67	0.65	0.66
KNN	**0.76**	0.64	0.65	0.64
KSTAR	**0.69**	0.6	0.59	0.59
LR	**0.71**	0.64	0.64	0.64
BNGE	**0.74**	0.63	0.63	0.61
EACH	**0.62**	0.62	0.62	0.61
RISE	**0.73**	0.66	0.64	0.64
OR=400%				
AB	**0.77**	0.63	0.65	0.64
Bagging	**0.77**	0.62	0.65	0.64
C4.5	**0.77**	0.63	0.6	0.61
RIPPER	**0.78**	0.69	0.69	0.69
SLIPPER	**0.76**	0.66	0.66	0.67
KNN	**0.76**	0.64	0.65	0.65
KSTAR	**0.72**	0.61	0.59	0.59
LR	**0.72**	0.64	0.65	0.64
BNGE	**0.74**	0.63	0.64	0.62
EACH	0.58	**0.62**	0.61	**0.62**
RISE	**0.74**	0.66	0.64	0.64
OR=500%				
AdaBoost	**0.75**	0.65	0.67	0.65
Bagging	**0.76**	0.65	0.66	0.63
C4.5	**0.76**	0.63	0.63	0.63
RIPPER	**0.78**	0.71	0.68	0.69
SLIPPER	**0.75**	0.67	0.68	0.65
KNN	**0.74**	0.64	0.66	0.65
KSTAR	**0.72**	0.61	0.59	0.61
LR	**0.74**	0.65	0.66	0.64
BNGE	**0.74**	0.64	0.65	0.63

EACH	0.58	**0.62**	0.61	0.61
RISE	**0.72**	0.66	0.64	0.64

Table 8.3: Balance Result of SMP Models

ML Technique	MSLSMOTE	SafeSMOTE	BSMOTE	SMOTE
OR=100%				
AdaBoost	**60.06**	53.13	54.53	55.03
Bagging	**60.21**	49.78	47.54	47.67
C4.5	**57.26**	45.77	46.7	45.41
RIPPER	**72.43**	63.94	64.42	63.35
SLIPPER	**64.02**	55.67	54.67	53.54
KNN	**64.5**	52.35	53.88	54.11
KSTAR	**50.29**	43.48	43.78	43.82
LR	**54.11**	47.32	47.88	47.09
BNGE	**61.88**	50.95	51.29	50.63
EACH	48.89	**50.65**	**50.65**	**50.65**
RISE	**61.09**	56.7	53.68	53.93
OR=200%				
AdaBoost	**67.18**	51.4	53.17	52.73
Bagging	**65.46**	48.83	51.49	50.69
C4.5	**66.22**	46.63	47.4	47.89
RIPPER	**75.13**	63.48	62.77	63.03
SLIPPER	**69.47**	56.76	56.57	52.52
KNN	**69.92**	52.92	55.44	54.51
KSTAR	**54.21**	45.01	43.77	43.93
LR	**59.19**	48.11	48.68	48.05
BNGE	**69.35**	50.81	50.82	50.57
EACH	49.01	**50.65**	**50.65**	**50.65**
RISE	**68.07**	57.13	54.37	54.26
OR=300%				
AdaBoost	**71.02**	52.74	51.25	52.07
Bagging	**70.92**	49.79	51.77	51.65
C4.5	46.78	**50.17**	47.68	48.05
RIPPER	**74.4**	62.59	64.5	64.47
SLIPPER	**72.66**	56.33	56.51	54.84
KNN	**72.42**	52.92	55.66	56.05

KSTAR	**59.92**	46.74	44.57	44.45
LR	**63.33**	49.02	46.79	49.94
BNGE	**71.67**	52.03	52.25	49.7
EACH	48.6	**51.28**	50.65	51.28
RISE	**64.7**	57.12	54.4	54.24
OR=400%				
AdaBoost	**73.02**	51.46	54.02	53.2
Bagging	**73.49**	47.84	53.73	52.16
C4.5	**74**	48.28	45.7	47.67
RIPPER	**74.64**	64.41	63.34	65
SLIPPER	**72.56**	55.29	55.67	58.22
KNN	**76.34**	52.92	56.52	56.61
KSTAR	**63.47**	47.1	46.48	45.82
LR	**65.88**	50.1	50.98	49.24
BNGE	**71.57**	52.95	53.11	51.46
EACH	41.79	**51.28**	49.17	**51.28**
RISE	**71.87**	57.15	56.77	54.64
OR=500%				
AdaBoost	**71.36**	54.49	56.98	59.1
Bagging	**72.09**	52.02	54.47	49.98
C4.5	**71.54**	50.58	50.42	50.69
RIPPER	**75.03**	66.47	62.51	64.59
SLIPPER	**71.32**	57.05	56	53.65
KNN	**72.63**	52.92	56.44	56.27
KSTAR	**61.97**	46.79	45.35	47.28
LR	**64.64**	51.52	52.92	51.5
BNGE	**70.69**	55.42	53.83	52.82
EACH	42.66	**50.66**	**50.66**	**50.66**
RISE	**67.78**	57.09	54.15	54.55

Table 8.4: G-Mean Result of SMP Models

ML Technique	**MSLSMOTE**	**SafeSMOTE**	**BSMOTE**	**SMOTE**
OR=100%				
AdaBoost	**62.84**	54.9	57.32	58.35
Bagging	**63.32**	49.93	44.36	44.41
C4.5	**59.49**	38.7	40.73	38.74

RIPPER	**73.38**	65.61	66.39	65.35
SLIPPER	**67.07**	58.33	57.96	56.69
KNN	**67.22**	54.43	56.75	56.95
KSTAR	**48.05**	36.8	37.17	37.5
LR	**55.87**	47.82	48.7	47.68
BNGE	**64.51**	52.96	53.68	52.97
EACH	47.62	**48.72**	**48.72**	**48.72**
RISE	**63.94**	59.8	56.83	57
OR=200%				
AdaBoost	**70.24**	53.26	55.44	49.69
Bagging	**68.91**	46.15	52.51	51.58
C4.5	**69.61**	39.94	43.69	42.33
RIPPER	**76.47**	66.03	65.33	65.11
SLIPPER	**72.61**	59.98	59.58	54.41
KNN	**72.25**	55.5	58.2	57.44
KSTAR	**53.69**	38.8	37.12	37.44
LR	**61.85**	48.6	49.54	48.53
BNGE	**71.67**	52.82	52.25	51.94
EACH	48.30	**48.72**	**48.72**	**48.72**
RISE	**70.66**	60.14	57.58	57.34
OR=300%				
AdaBoost	**73.51**	55.13	50.11	54.4
Bagging	**73.55**	48.61	53.25	52.15
C4.5	**73.63**	47.75	44.23	43.49
RIPPER	**76.38**	64.78	67.05	66.83
SLIPPER	**75.06**	59.83	60.18	57.37
KNN	**74.11**	55.5	58.5	58.98
KSTAR	**61.6**	43.58	38.52	40.11
LR	**66.34**	46.93	47.75	50.93
BNGE	**73.61**	53.67	54.32	50.67
EACH	47.79	**49.91**	48.72	**49.91**
RISE	**65.50**	60.13	57.62	57.3
OR=400%				
AdaBoost	**75.2**	53.42	56.79	55.53
Bagging	**75.59**	46.26	55.78	53.55
C4.5	**75.95**	42.28	38.56	43.74
RIPPER	**76.64**	65.76	65.52	67.22

SLIPPER	**74.64**	58.22	58.99	61.39
KNN	**77.36**	55.5	59.38	59.44
KSTAR	**65.49**	40.7	42.99	41.32
LR	**68.44**	51.08	52.68	50.23
BNGE	**73.23**	55.06	55.61	53.62
EACH	38.58	**49.91**	48.37	**49.91**
RISE	**73.45**	60.42	59.9	57.54
OR=500%				
AdaBoost	**72.81**	56.59	60.24	61.77
Bagging	**73.3**	52.8	56.97	50.46
C4.5	**73.06**	51.25	51.45	51.24
RIPPER	**76.66**	67.95	64.35	66.29
SLIPPER	**72.68**	60.07	58.08	56.84
KNN	**73.68**	55.5	59.34	59.02
KSTAR	**64.15**	42.61	41.46	43.24
LR	**70.68**	53.16	55.03	53.12
BNGE	**72.22**	57.95	56.56	55.39
EACH	41.17	**48.77**	**48.77**	**48.77**
RISE	**65.03**	60.17	57.21	57.47

8.4.2 Results and Analysis of RQ2

As among the four data resampling techniques, the bar plots depict that the performance of MSLSMOTE is the best in terms of all three overall performance metrics (Figure 8.2(a)-(c)); results are statistically analyzed with the Friedman test and Wilcoxon signed-rank test with Bonferroni correction (significance level of α = 0.05/3 = 0.0166. The Friedman test is applied at a confidence level equal to 95% (α = 0.05). With the help of the Friedman test, the statistically best performing technique was identified. The hypothesis for conducting the Friedman test is:

- *Null hypothesis* H_0: The maintainability prediction models after MLSMOTE, SafeSMOTE, BSMOTE., and SMOTE have the same performance in terms of AUC, Balance, and G-Mean.

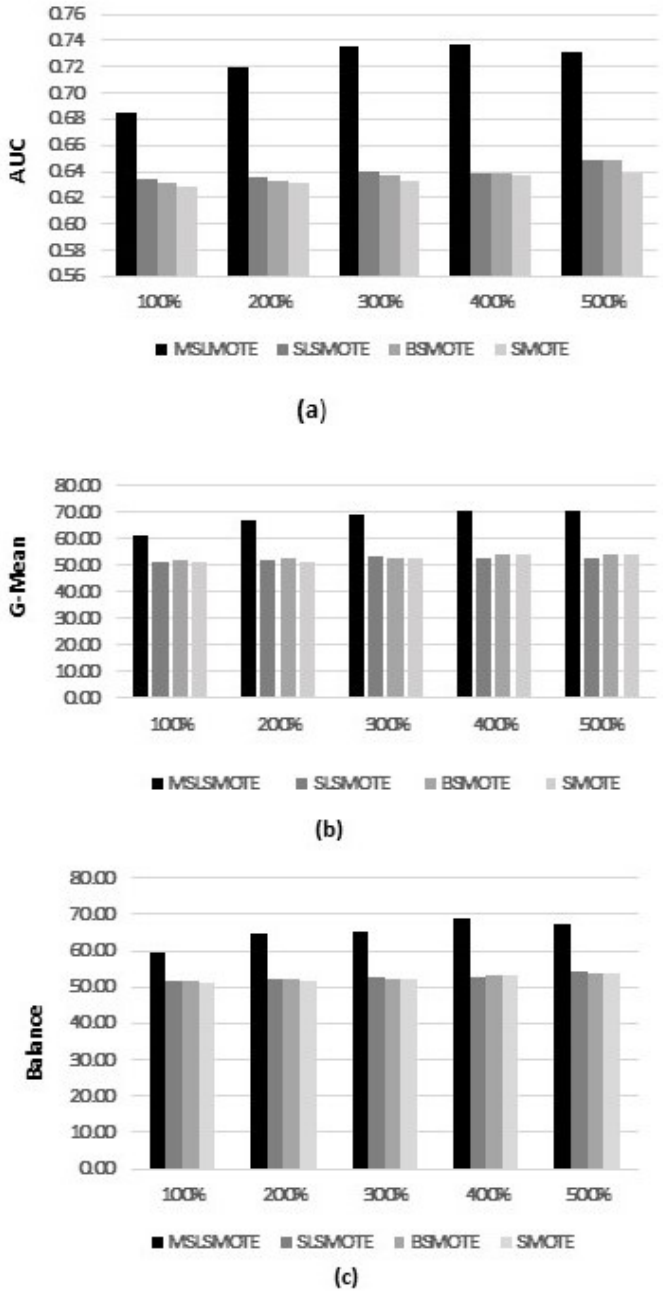


Figure 8.2: The Results (a) average AUC (b) average G-Mean (c) average Balance of SMP Models

- *Alternate hypothesis* H_a: The maintainability prediction models after MLSMOTE, SafeSMOTE, BSMOTE, and SMOTE do have the same performance in terms of AUC, Balance, and G-Mean.

The Friedman test gives a mean rank to each compared technique in terms of its performance. The mean ranks attained by each technique by the Friedman test for performance measure Balance, AUC, G-Mean are shown in Table 8.5. The p-values obtained were 0.00 for all three performance measures which mean that the performance of the compared techniques differs significantly in terms of Balance, G-Mean, and AUC. As in all three cases, MSLSMOTE has attained the lowest rank, which implies that MSLMSOTE is a statistically best-performing data resampling technique.

Table 8.5: Friedman Test Ranking

Technique	**Mean Rank (Balance)**	**Mean Rank (G-Mean)**	**Mean Rank (AUC)**
MSLSMOTE	1.33	1.27	1.1.5
SafeSMOTE	2.84	2.94	2.76
BSMOTE	2.8	2.73	2.86
SMOTE	3.04	3.06	3.22

With the Wilcoxon test 's help, the performance of models predicting software maintainability developed after data resampling with MSLSMOTE is compared against those developed after data resampling with SafeSMOTE, BSMOTE, SMOTE. The pairwise performance with regard to Balance, G-Mean, and AUC on all datasets is compared irrespective of the ML technique used in the chapter. In the test procedure, the null hypothesis is set as a difference between average AUC, G-Mean, and Balance from MSLSMOTE and other data resampling techniques; SafeSMOTE, BSMOTE, and SMOTE are not significant. The test's alternative hypothesis evaluated was the difference between average AUC, G-Mean, and Balance after data resampling with

MSLSMOTE and other data resampling techniques is significant.

Table 8.6: Friedman Test Ranking

Dataset	**MSLSMOTE vs.**	**p-value**
Bcel	SafeSMOTE	0.000
	BSMOTE	0.000
	SMOTE	0.000
Betwixt	SafeSMOTE	0.000
	BSMOTE	0.000
	SMOTE	0.000
Io	SafeSMOTE	0.000
	BSMOTE	0.000
	SMOTE	0.000
Ivy	SafeSMOTE	0.000
	BSMOTE	0.000
	SMOTE	0.000
Jcs	SafeSMOTE	0.000
	BSMOTE	0.000
	SMOTE	0.000
Lang	SafeSMOTE	0.000
	BSMOTE	0.000
	SMOTE	0.000
Log4j	SafeSMOTE	0.000
	BSMOTE	0.000
	SMOTE	0.000
Ode	SafeSMOTE	0.000
	BSMOTE	0.000
	SMOTE	0.000

For the test, the sample is chosen independently of the ML techniques. The p-values obtained after the Wilcoxon signed-rank test are recorded in Table 8.6. In all the comparisons, the p-values were lesser than 0.0166 the led to the rejection of null hypothesis. The alternate, which states that the difference between average AUC, G-Mean, and Balance after data resampling with MSLSMOTE and other data resampling techniques is significant, is accepted. From the test result, it is concluded

that in terms of G-Mean, AUC, and Balance, maintainability models developed after data resampling with MSLSMOTE are significantly superior to those built after the data resampling using SafeSMOTE, BSMOTE, and SMOTE independent of ML techniques.

8.5 Comparison of Various Studies

In this section, our results are compared with literature studies and previous chapters in terms of various performance measures (AUC, G-Mean, and Balance). The values of different performance measures from previous chapters and literature studies are presented in Table 8.7. In this table, we reported the median values of each performance measure, obtained over all the datasets in a chapter or a study. It is to be noted that the values for only that technique are stated in Table 8.7 which is the best in the chapter or a study and prediction models are developed using OO metrics. In Table 8.7, "–" denotes that corresponding performance measure value could not be extracted from the study. It is to be noted that, the literature studies for comparison with our results are those in which the prediction models are developed for identifying change prone classes. Table 8.7, the LR technique was the best in Chapter 4, underbagging technique was the best in Chapter 5. CHC, an SB technique was best in Chapter 6 and two HB techniques PSOLDA and GFS-LB exhibited the best performance Chapter 7. Catolino et al. [183] assessed a number of a different set of predictors (number of developers, structural and semantic scattering of developers, evolution-based metrics, and OO metrics)

Table 8.7: Results of Comparison of Classification Techniques

Performance Measure	Chapter 4 (LR)	Chapter 5 (UB)	Chapter 6 (CHC)	Chapter 7 (GFS-LB, PSOLDA)	[225] (MPLCS)	[17] (PSOLDA)	[183] (Vot)	[21] (NB)
G-Mean	68.35	72.28	71.17	(70.77, 70.22)	70	69	–	–
Balance	68.00	71.85	70.10	(70, 70.47)	69.06	66	–	–
AUC	–	–	–	–	–	–	0.56	0.74
UB indicates underbagging, vot indicates voting ensemble								

Table 8.8: Results of Comparison of Imbalance Learning Techniques

Performance Measure	Chapter 4 (SafeS-MOTE)	Chapter 5 (UB)	Chapter 6 (SafeS-MOTE)	Chapter 7 (SafeS-MOTE)	Chapter 8 (MSLSMOTE)	[74] (ANC)	[74] (DANC)
G-Mean	69.14	72.28	71.01	73.11	73.70	68.09	70.02
Balance	68.34	71.85	71.01	72.21	70.48	68.34	69.53
AUC	–	–	–	–	0.76	0.80	0.79
UB indicates underbagging, ANC indicates AdaBoost.NC, DANC indicates Dynamic AdaBoost.NC							

using the ML techniques NB, LR, MLP, Bagging, RF, and voting ensemble technique for developing change prediction models. The results of this study advocated a voting-based ensemble as the best technique with a median AUC of 0.56. Romano and Pinzger [21] investigated ML techniques namely NB, SVM, and NB. The NB, an ML technique reported best in terms of median AUC value of 0.74, The results of our Chapter 4 advocated that LR is the best technique with median G-Mean and Balance of 68.35 and 68 respectively. The results of Chapter 4 could not be compared with the results of Calolino et al. [183] and Romano and Pinzger [21] because of the use of different performance measures. The study by Malhotra and Khanna [17] advocated the PSOLDA as best technique for predicting change-prone software classes. Their

results were validated on six open-source datasets and we reported the median results on these six datasets. Another study by Malhotra and Khanna [225] evaluated eight open-source datasets for identifying change prone classes using eight SB techniques, four ML, and one statistical technique. We reported the median value of the best technique amongst all investigated techniques. The results depicted in Table 8.7 shows that underbagging technique showed the best G-Mean and Balance values in Chapter 5. The G-Mean result of underbagging technique was closely followed by the CHC, GFS-LB, and PSOLDA techniques. CHC is an SB technique, whereas PSOLDA and GFS-LB are HB techniques. Thus, we advocate underbagging, PSOLDA, GFS-LB, and CHC techniques for predicting software maintainability. Also, the study by Malhotra and Khanna [17] investigated HB techniques on eight open-source datasets and they advocated PSOLDA as the best HB technique with a median G-Mean of 69 and median Balance of 66. In Chapter 7, PSOLDA is advocated as one of the best HB techniques with a median G-Mean of 70.22 and median Balance of 70.47 and there is a small difference performance of the two (the performance of PSOLDA in the Chapter 7 and reported by [17]. Therefore, we advocate the effectiveness of the PSOLDA technique in the domain of SMP.

To take care of the imbalanced data problem, data resampling techniques have been investigated in the thesis. SMP models have been developed with ML, SB, and HB techniques in previous chapters after balancing data with resampling techniques. It is to be noted that we have examined the same set of data resampling techniques on the same datasets to establish their effectiveness with a different set of classification techniques. In Chapter 5, data resampling techniques are aggregated with ensemble learners to provide algorithmic level solutions to the imbalanced data problem in SMP. The best resampling technique in each chapter and literature studies of change prediction are reported in Table 8.8. It is to be noted that the imbalanced data problem is not addressed in any of the software maintainability studies in the literature and

very few studies in the for prediction of change-prone software classes addressed this issue. Therefore, we could compare the results with only those studies.

In Table 8.8, values of performance best performing data resampling technique from previous chapters and literature studies have been reported. Wang & Yao [74] evaluated the use of five class imbalance learning methods and found Adaboost.NC to be best for the ten investigated datasets. They further proposed a new method Dynamic Adaboost.NC. The G-Mean values were 68.9 and 70.2 for Adaboost.NC and Dynamic AdaBoost.NC respectively whereas the Balance values were 68.34 and 69.53 respectively. Both Adaboost.NC and Dynamic AdaBoost.NC are the algorithmic level solution to the imbalanced data problem. In Chapter 5, we have investigated algorithmic level techniques to take care of imbalance data problem and underbagging technique was the best with a median G-Mean of 72.18 and a median Balance of 72.21. As shown in Table 7.15, the best performing data resampling technique in Chapter 4 was SafeSMOTE i.e., SMP models developed using ML techniques performed best after data resampling with SafeSMOTE. The SafeSMOTE technique performed best with a median G-Mean of 69.14 and a median Balance Value of 68.34 in Chapter 4. Also, SMP models developed using SB techniques performed best after data resampling with SafeSMOTE in Chapter 5. The SafeSMOTE technique performed best with a median G-Mean of 71.01 and a median Balance value of 71.07 in Chapter 5. A similar trend prevailed in the Chapter 7. In Chapter 7, the SafeSMOTE technique was investigated as the best technique. The SMP models developed using HB techniques after data resampling with SafeSMOTE performed best with a median G-mean of 73.11 and a median Balance of 72.21 in Chapter 7. Thus, we favor the use of the SafeSMOTE technique as ML, SB, and HB techniques investigated in this thesis performed best with SafeSMOTE. In the current chapter, we proposed MSLSMOTE that is the modification of SafeSMOTE. The SMP models are developed using ML techniques after data resampling with MSLSMOTE gave median G-Mean, Balance

and AUC of 73.70, 70.48 and 0.76 respectively. It is to be noted that SMP models on the same ML techniques after data resampling with SafeSMOTE gave median G-Mean and Balance of 69.14 and 68.34 respectively in Chapter 4. Therefore, these results indicate that MSLSMOTE is an effective technique to deal with the imbalanced data.

8.6 Discussion

The objective of this chapter was to determine the low maintainability software classes accurately. We examined four data resampling techniques including the proposed technique MSLSMOTE at different rates of oversampling with ML techniques. The oversampling rates determines the number of sythetic data points to be introduced in the imbalanced training dataset. MSLMSOTE offers an effective enhancement in the performance of prediction models of software maintainability, according to AUC, Balance, and G-Mean at different oversampling rates. However, at oversampling rate of 400% and 500%, the SMP models developed using MSLSMOTE gave the highest performance in terms of G-Mean and Balance. The highest performance of MSLSMOTE at these oversampling rates was due to the reason that at these oversampling rates, MSLSMOTE able to get more synthetic instances of low maintainability class data points to make a appropriate balance with the data points of high maintainability class data points.

MSLMSOTE offers an substantial enhancement in the performance of prediction models of software maintainability, according to AUC, Balance, and G-Mean. Its efficacy is shown as being the best resampling technique in all datasets and ML techniques. The results of Wilcoxon signed-rank test also confirmed that the significant positive difference using AUC, Balance, and G-Mean from MSLSMOTE to SafeSMOTE, BSMOTE, and SMOTE is attained autonomously of datasets and ML

techniques.

In this chapter, MSLSMOTE has been applied to imbalanced maintainability prediction datasets to accurately predict low maintainability classes. This resampling technique can be useful to an imbalanced dataset to obtain a balanced dataset containing the original minority class data and synthetic data points. The balanced dataset can benefit training a valuable model to forecast unknown data points. The proposed approach introduces an effective way to handle imbalanced data that results in developing efficient SMP models.

Chapter 9

Inter-Project Validation For Software Maintainability Prediction

9.1 Introduction

Researchers have developed various prediction models in the literature to accurately predict the software maintainability [8, 13, 56, 66, 173]. The development of such models requires a dataset for training. The training dataset for the development of SMP model comprises of software metrics and maintainability of the concerned project in the form of lines of code added, deleted, or modified in the maintenance period. This data collection for training the prediction model is one of the difficult tasks because in most cases either such data is unavailable or it is difficult to collect. In order to overcome this limitation of historical data collection, the development of generalized maintainability prediction models is necessary where the historical data of a particular project can be utilized for other similar kinds of projects. This approach is called inter-project or inter-project validation. Thus, the situation in which there is the inadequacy of resources and lack of time to capture training data for

the development of maintainability prediction model, inter-project validation can be employed. This chapter investigates the applicability of the inter-project validation approach where the training dataset of one project is validated on other projects. In software engineering predictive modeling, inter-project validation is the process of validating or testing a prediction model that is being developed by making use of training data of some other project. Thus, in order to investigate the effectiveness of this strategy, several studies have been published. Kitchenham et al. [226], analyzed ten studies that compared cost estimation models developed using within company and inter-company projects datasets. The results of their analysis were declared inconclusive as in three studies under their analysis, inter-company models were equally good as within company models while in four studies within company models outperformed compared to inter-company estimation models. Turhan et al. [227] investigated the suitableness of inter-project strategy for the development of defect prediction models. They used analogy-based learning to inter-company data to fine-tune the models for prediction of defects in software classes. The performance of inter-company defect prediction models was compared with defect prediction models developed using within-company data in this study. The performance of both was found comparable in their study. Canfora et al. [228] presented an inter-project defect prediction model using multi-objective regression analysis. The model was empirically validated on ten datasets and gave quite promising results. Watanabe et al. advocated the use of inter-project validation for fault prediction. A large-scale study, examining the use of inter-project defect prediction by analyzing twelve real-world projects was conducted by Zimmermann et al. [229]. The study revealed that inter-project prediction models developed using the projects with the same domain are not always successful and this study determined the factors that strongly influence inter-project defect prediction. He et al. [143] conducted three large experiments on 34 datasets extracted from 10 open-source projects to examine the effectiveness of inter-project

validations. Their research revealed that in some cases inter-project validation strategy gives better estimation. Malhotra et al. [230] developed change proneness prediction models to predict change-prone classes using ML and statistical techniques. They advocated that inter-project validation helps in developing generalized prediction models. Thus, all the above studies attributed to establishing the capability of inter-project validation in predictive modelings like defect prediction, prediction of change proneness, and cost estimation. However, to the best of our knowledge inter-project validation for SMP has not been investigated in the literature. Also, the existence of this research gap was ascertained by the review conducted in Chapter 3. We investigate the following RQs in this chapter.

- RQ1) What is the predictive performance of SMP models developed using inter-project validation?
- RQ2: What is the predictive performance of SMP models that are trained and tested on the same dataset using ten-fold validation?
- RQ3: Does the performance of models developed for predicting software maintainability using inter-project validation and ten-fold validation differ significantly?

In order to answer the above research questions, the SMP models developed using ML techniques (RF, DS, AR, Bagging, KNN, KSTAR, MLR, PART, CART, MLP-BP, RBFNN, REPTree, SVM). The experiments were conducted on three open-source software datasets (HtmlUnit, Click, and Maven). This chapter is structured as follows: Section 9.2 presents an experimental framework for developing SMP models using inter-project validation. Section 9.3 presents the results using ten-fold cross validation. Section 9.4 presents inter-project validation results. The statistical analysis of models developed using ten-fold cross-validation and inter-project validation is described in

section 9.5. Finally, Section 9.6 presents the discussion. The results of this chapter are published in [231].

9.2 Research Methodology

This section describes the research methodology followed in this chapter. Figure 9.1 presents the experimental framework of this chapter.

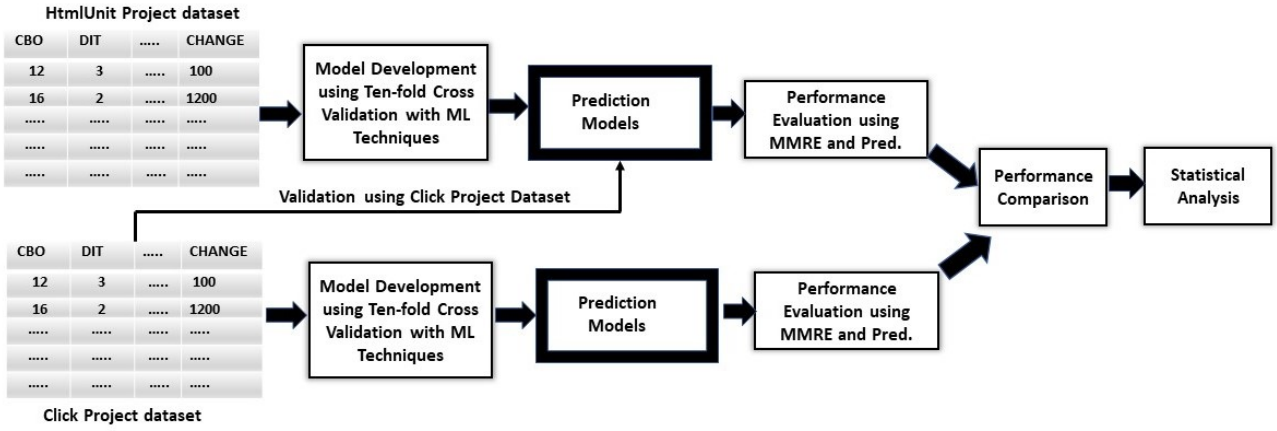


Figure 9.1: Experimental Framework for Developing SMP Models using Click Project

9.2.1 Independent and Dependent Variables

Training a model in order to accomplish a prediction task requires a dataset comprising of independent variables and dependent variables. Independent variables used in this chapter are C&K metrics [36]. Maintenance effort is termed as CHANGE (total SLOC changed) is used as the dependent variable used in this chapter. This is a continuous variable and is defined as the total number of source code lines changed in a class.

9.2.2 Empirical Data Collection

In order to validate inter-project validation, three Apache application packages (HtmlUnit, Maven and Click) are used in this chapter. The datasets from these projects are collected according to the data extraction procedure described in Chapter 2.

9.2.3 Model Development and Performance Evaluation

In this chapter, SMP models are developed using inter-project validation and ten-fold cross-validation. The predictive performance of the models developed using both strategies is compared with the help of performance measures MMRE and Pred. The formulae to compute MMRE and Pred are given in Chapter 2. The statistical comparison is carried with the help of the Wilcoxon signed-rank test.

9.3 Inter-Project Validation Results

The performance of SMP models developed using inter-project validation is reported in this section.

9.3.1 Answer Specific to RQ1

In order to answer RQ1, we used the HtmlUnit dataset for training and applied ML techniques stated in Section 9.1 to develop maintainability prediction models. These models are then validated using test datasets. In this experiment, we used two datasets for validating the prediction models developed using the HtmlUnit dataset. These datasets are the Maven dataset and the Click dataset. SMP models developed using the HtmlUnit dataset as training are validated on the above-mentioned datasets individually. This explains the process of inter-project validation. The process of

inter-project validation is schematically described in Figure 9.1. The performance of the maintainability prediction models built using inter-project validation is described with the help of performance measures given stated in Section 9.2. The reason for considering these performance measures for the purpose of evaluation of the developed prediction models is that these are the de-facto standard for performance evaluation for predictive modeling. The performance of models in terms of MMRE and Pred, developed by applying inter-project validation for Click and Maven datasets is shown in Table 9.1 and Table 9.2 respectively.

Table 9.1: Performance of Click dataset using inter-project validation

ML Technique	**MMRE**	**Pred(25)**%	**Pred(30)**%	**Pred(75)**%
AR	0.73	14	24	55
Bagging	0.64	14	17	49
DS	0.66	13	14	54
RF	0.67	15	17	50
KNN	0.79	16	18	55
KSTAR	0.68	08	14	55
MLR	1.02	10	11	44
PART	0.71	17	22	59
CART	0.72	16	21	52
MLP-BP	1.20	36	37	57
RBFNN	0.68	13	16	50
REPTree	0.66	14	17	50
SVM	1.59	14	18	46

As shown in Table 9.1, when the prediction model trained using HtmlUnit dataset with the application of the Bagging technique is validated on the Click dataset, it gave an MMRE of 0.64. Other ML techniques: DS, REPTree, RF, KSTAR, and RBFNN gave an MMRE of 0.66, 0.66, 0.67, 0.67, and 0.68 respectively. It is to be noted that the MMRE of the above-mentioned ML techniques is in the range of 0.64 to 0.68 which is quite encouraging. The PART technique gave the highest Pred(75) performance of 59% whereas the MLP-BP technique outperforms with a with respect to

Pred(25) and Pred(30) with a value of 36% and 37% respectively. As shown in Table 9.2 when the prediction model trained using HtmlUnit dataset with the application of the CART technique is validated on the Maven dataset, it gave MMRE of 0.61. Except for SVM and MLP-BP, the results of all other ML techniques for inter-project validation carried out using the Maven dataset are in the range of 0.61 to 0.68. For this dataset, KNN outperforms with respect to Pred(25), Pred(30), and Pred(75) with values 26%, 27%, and 65% respectively.

Table 9.2: Performance of Maven dataset using inter-project validation

ML Technique	**MMRE**	**Pred(25)**%	**Pred(30)**%	**Pred(75)**%
AR	0.65	09	19	56
Bagging	0.67	12	14	46
DS	0.63	14	14	44
RF	0.67	13	13	53
KNN	0.63	26	27	65
KSTAR	0.63	15	18	57
MLR	0.64	14	16	64
PART	0.64	13	14	51
CART	0.61	12	14	51
MLP-BP	0.75	17	20	62
RBFNN	0.67	12	13	46
REPTree	0.66	13	13	45
SVM	0.73	13	16	45

we used Friedman Test at 95% level of significance with 12 degrees of freedom (13 ML techniques) with respect to performance measure MMRE to statistically examine the performance of thirteen ML techniques for inter-project validation. The result of the Friedman Test is shown in Table 9.3. The p-value obtained after the Friedman test was 0.000 that indicated that result of the Friedman test is significant.

Table 9.3: Ranking produced by Friedman Test According to MMRE for inter-Project Validation

Technique	Mean Rank
DS	2.75
KSTAR	4.25
CART	5.50
REPTree	6.25
Bagging	6.50
KNN	7.50
PART	7.50
RF	7.50
RBFNN	8.75
MLR	9.75
AR	10.00
MLP-BP	14.50
SVM	14.50

As shown in Table 9.3, with respect to MMRE, the DS technique is best for maintainability prediction for inter-project validation as this technique has obtained the smallest rank amongst all 13 ML techniques used in this chapter followed by the KSTAR technique. Therefore, for RQ1 we conclude that for inter-project validation, maintainability prediction models developed with the applications of DS, REPTree, RF, KSTAR, and RBFNN techniques gave MMRE in a range of 0.63 to 0.68 over both datasets which are quite encouraging. Also, DS is the best technique among all ML techniques selected in this chapter for predicting software maintainability.

9.4 Ten Fold Cross-Validation Results

The performance of SMP models developed using ten-fold cross-validation is reported in this section.

9.4.1 Answer Specific to RQ2

In the second experiment, the predictive performance of the SMP models that are trained and tested on the same dataset is analyzed. In this experiment, we developed maintainability prediction models for Click and Maven datasets using ten-fold cross-validation. The performance of the developed models for Click and Maven datasets is reported in Table 9.4 and Table 9.5 respectively.

Table 9.4: Performance of Click dataset using Ten-Fold cross-validation

ML Technique	**MMRE**	**Pred(25)**%	**Pred(30)**%	**Pred(75)**%
AR	0.62	16	19	57
Bagging	0.64	14	19	50
DS	0.69	12	15	45
RF	0.64	15	17	54
KNN	0.76	09	28	63
KSTAR	0.59	20	25	65
MLR	0.62	15	17	52
PART	0.66	15	18	52
CART	0.66	14	15	50
MLP-BP	0.84	13	15	50
RBFNN	0.68	11	12	50
REPTree	0.68	11	13	47
SVM	0.66	15	22	61

As shown in Table 9.4, for the Click dataset, the KSTAR technique gave the lowest MMRE of 0.59 followed by AR and LR. Also, For the Click dataset, the KSTAR technique gave the best Pred(25) and Pred(75). The corresponding values of Pred(25) and Pred(75) of the KSTAR technique are 20% and 65% respectively. KNN technique reported the highest Pred(30) of 28%. For the Maven dataset, again KSTAR gave the lowest MMRE of 0.53 followed by AR as shown in Table 9.5.

Table 9.5: Performance of Maven dataset using Ten-Fold cross-validation

ML Technique	**MMRE**	**Pred(25)**%	**Pred(30)**%	**Pred(75)**%
AR	0.60	11	15	69

Bagging	0.61	12	14	55
DS	0.64	13	17	51
RF	0.66	12	16	50
KNN	0.90	31	31	56
KSTAR	0.53	23	27	73
MLR	0.61	16	19	54
PART	0.61	15	17	61
CART	0.62	15	18	59
MLP-BP	0.69	16	20	63
RBF	0.62	10	16	55
REPTree	0.61	15	15	48
SVM	0.74	16	20	69

Also, the performance of the KSTAR technique is best in terms of Pred(75) whereas, KNN is best in terms of Pred(25) and Pred(30) as shown in Table 9.4. Further, in order to statistically examine the performance of ML techniques for ten-fold validation over two datasets, we apply Freidman Test at a 95% level of significance with 12 degrees of freedom (13 ML techniques) with respect to performance measure MMRE. The results of the Freidman Test are depicted in Table 9.6.

Table 9.6: Ranking produced by Friedman Test according to MMRE for Ten-Fold cross-validation

Technique	**Mean Rank**
KSTAR	1.0
AR	2.25
MLR	3.75
BAGG	5.25
PART	6.75
REPTree	8.25
RF	8.75
CART	8.75
RBFNN	10.25
SVM	11.25
DS	12.00
MLP-BP	14.00
KNN	14.

Thus, as shown in Table 9.6, the KSTAR technique is best for maintainability prediction for when prediction models are trained and tested on the same datasets considered in this chapter. This technique has obtained the smallest rank amongst all ML techniques used in this chapter followed by AR technique which has obtained a rank of 2.25. Therefore, for RQ2 we conclude that For ten-fold cross-validation, maintainability prediction models developed with the application of AR, Bagging, RF, KSTAR, MLR, PART, CART, RBFNN, and REPTree techniques gave MMRE in a range of 0.61 to 0.68 over both datasets which is a quite acceptable range. Also, KSTAR is the best technique amongst all ML techniques selected in this chapter for predicting software maintainability on the basis of ranking given by the Friedman test on ten-fold cross-validation.

9.5 Statistical Analysis using Wilcoxon Signed-Rank Test

To establish the applicability of inter-project validation in the prediction of software maintainability, the performance of the prediction model developed by applying a particular ML technique using inter-project validation (Section 9.3) should not differ significantly from that of the models developed using ten-fold cross-validation by applying the same ML technique (Section 9.4). The existence of a significant difference in the performance weakens the applicability of inter-project validation. The following hypothesis is tested.

- ***Null Hypothesis (H0):*** There is no significant difference in the performance of the maintainability prediction model developed using inter-project validation and ten-fold cross-validation.

- ***Alternate Hypothesis (Ha):*** The performance of the maintainability prediction model developed using inter-project validation and ten-fold cross-validation differs significantly.

Therefore, in order to statistically validate the feasibility of inter-project validation, we apply the pairwise Wilcoxon test at a 95% significance level ($\alpha = 0.05$) on the performance of each of the thirteen ML techniques for inter-project validation and ten-fold validation and validated $H0$ and Ha. The performance of individual ML technique with respect to MMRE values for inter-project validation and ten-fold cross-validation approach is compared over the two datasets considered under this chapter. Table 9.7 shows the pairwise results of the Wilcoxon test for each of the thirteen ML techniques for inter-project and ten-fold validation.

Table 9.7: Results of Wilcoxon Test on cross-validation vs Ten fold validation

Technique	**p-value**
AR^C-AR^T	0.65
$Bagging^C$-$Bagging^T$	0.31
DS^C-DS^T	0.18
RF^C-RF^T	0.18
KNN^C-KNN^T	0.65
$KSTAR^C$-$KSTAR^T$	0.18
MLR^C-MLR^T	0.18
$PART^C$-$PART^T$	0.65
$CART^C$-$CART^T$	0.65
SVM^C-SVM^T	0.65
$REPTree^C$- $REPTree^T$	0.65
$RBFNN^C$-$RBFNN^T$	0.31
MLP-BP^C-MLP-BP^T	0.18

Table 9.7 depicts the p-value of the pairwise comparison of different techniques for inter-project and ten-fold validation. It is evident from Table 9.7 that for all of the 12 pairs, the p-value is greater than 0.004 (with Bonferroni correction). Therefore, we accept the null hypothesis and reject the alternate hypothesis. Therefore, for

RQ3, on the basis of the results of the pairwise Wilcoxon test, it is concluded that *the performance of maintainability prediction models developed using inter-project validation and ten-fold validation does not differ significantly. Thus, a training dataset computed from the past project can effectively be utilized to predict the maintainability of the software project under development.*

9.6 Discussion

The aim of this chapter was to investigate the applicability of inter-project validation for the prediction of software maintainability. To assess and analyze the capability of inter-project validation, three datasets namely HtmlUnit, Click, and Maven are used. Two experiments are being conducted in this chapter.

In the first experiment, the SMP models are developed using inter-project validation and the performance of the models is examined. In the second experiment SMP models are developed using ten-fold cross-validation and the performance of the developed models is compared with the models developed using inter-project validation. The performance of SMP models in terms of mean MMRE, Pred(25%), Pred(50%), and Pred(75%) for click dataset in case of inter-project validation was observed to be 0.81, 15.33, 18.73, 52.20 respectively and for ten-fold cross-validation the performance of models for Click dataset was in was 0.67, 14, 18.20, and 53.33 respectively in terms of mean MMRE, Pred(25%), Pred(50%), and Pred(75%). Similarly, for Maven dataset, the performance of SMP models was 0.66, 14.13, 16.46, and 52.60 in terms of mean mean MMRE, Pred(25%), Pred(50%), and Pred(75%) respectively in case of inter-project validation whereas same dataset for ten-fold cross-validation gave a performance of 0.64, 15.60, 18.80, and 57.73 respectively in terms of mean MMRE, Pred(25%), Pred(50%), and Pred(75%). Therefore, these results clearly indicate that the performance of SMP models developed using ten-fold

cross-validation and inter-project validation is nearly same.

Also, results of the Wilcoxon test confirmed that there is no significant difference in the performance of the SMP models developed with the help of two approaches. Therefore, the results of this chapter confirm that the training dataset of one software project can be utilized effectively to develop generalised prediction models to forecast software maintainability.

Chapter 10

Conclusion

10.1 Summary of the Work

Software systems are subjected to continuous modifications and evolution to respond to the changes in the real world. The changes in the software systems are due to multiple causes, few of which include enhancing existing features, adding new features, and correction of the existing defects. As the complexity of software is increasing, maintaining such complex systems is becoming a challenge for software practitioners. The modifications may result in drastic changes in the structure and design of software components. As software maintenance costs a significant amount in the software development life cycle, therefore it is essential to take care that the modifications should not degrade the software quality. The prime aim of the work carried out in this thesis is to develop model effective models to determine the software maintainability of OO software systems by dealing with imbalanced data. As imbalanced data limits the performance and accuracy SMP models, various techniques have been applied in the thesis to overwhelm the adverse effects of imbalanced data. These approaches have been validated using various classification techniques to get the generalized SMP

models with validation on various datasets. The classification techniques ranging from ML techniques widely applied in predictive modeling to ML techniques never explored in the domain of SMP are examined in this thesis with data resampling techniques. For effective identification of low maintainability classes, a wide array of SB and HB techniques have also been investigated in this work with various data resampling techniques. The thesis also evaluates many ensembles learning techniques to effectively handle imbalanced data. To develop models, the thesis evaluated a number of predictors from various OO metric suites well established in software engineering predictive modeling. The work is conducted in this thesis in the form of organized and rigorous empirical experiments that make it very useful for the researchers and software practitioners. The empirical experiments are conducted systematically and properly in the thesis. The sequence of steps to conduct the experiments is discussed. The independent and dependent variables are described. All the data analysis techniques to develop models in the thesis are described with their basic functioning, characteristics, and parameter setting. This work focused on predicting the maintainability of datasets extracted from the open-source software systems. The open-source software systems are continuously modified by developers all over the world but due to lack of documentation and technical support, it is difficult to estimate their maintainability. In the area of SMP, hardly a few studies have developed models by extracting datasets from open-source software systems. The open-source project datasets aid easy replicability and generalizability of the results. For each dataset used in this thesis, we mentioned its descriptive statistics. The data preprocessing steps such as data discretization, outlier analysis, and feature selection method is described. The validation methods used to validate the prediction models are described. We also defined various performance metrics used in this thesis to evaluate the performance of the models and statistical. We also describe the statistical analysis methods that are used to statistically analyze the results of the chapters of the

thesis.

To assess and analyze works done in literature in software maintainability, we performed a systematic literature review in which we systematically reviewed 34 primary studies in the period from January 1990 to October 2019. The data extracted from these studies is analyzed with respect to various RQs framed to cater to the need for the systematic review. These studies were analyzed to answer various RQs with respect to the type of variables used for developing the models, categories of various data analysis techniques used for developing models, datasets used, the predictive performance of various techniques (most popular category of data analysis techniques found), validation methods used for validation of the models, the statistical tests used for results verification and the threats encountered and addressed model development. It was found that a majority of studies in literature used OO metrics for developing models and C&K metrics are popular metric suite in the literature studies, as predictors to develop models. The ML techniques were used and found effective in developing SMP models. In ML techniques, NN was explored by the majority of the studies to develop maintainability prediction models. The GMDH in the category was found effective to develop models. The performance of a few HB techniques was found effective, but we found a lack of studies (only eight studies) that assessed the HB techniques. It was found that most of the work in SMP was carried out on public datasets, UIMS, and QUES. There is a lack of studies that validated open-source or proprietary project datasets. Based on the review results, we also identified several future directions in the domain. As UIMS and QUES are very small sized datasets investigated in the majority of the studies, the large size dataset should be evaluated. There is no study in the literature, that has examined cross-project validation. So, cross-project validation should be explored to get generalized prediction models. Furthermore, more studies are required to empirically verify, assess, and compare the performance of various categories of data analysis techniques. Moreover, the

comparison results should be statistically assessed as according to the systematic review, 62% of primary studies did not use any statistical test for solidifying their conclusions. To fill the above-stated gaps, we develop SMP models using various ML techniques on eight open-source datasets. As investigated in the literature review, ML techniques are the most popular in the domain of SMP. ML techniques are effective, adaptive, and can learn easily from historical data. But imbalanced data puts hurdles for the effective training of models using ML techniques. Therefore, we preprocessed the imbalanced datasets with different data resampling techniques before the learning process. The results of the models are evaluated using G-Mean and Balance metrics. The performance of ML techniques significantly improved after the use of different sampling methods. The SMP models developed after resampling with SafeSMOTE performed well on all the datasets. The SafeSMOTE technique improved the performance of the models in terms of G-Mean and Balance. According to statistical analysis carried out with the Friedman test, the SafeSMOTE technique achieved the highest rank in terms of G-Mean and Balance.

The literature review conducted on software maintainability also pointed a few recent studies have explored the use of ensemble learners for developing models to predict software maintainability. We also assessed the use of ensembles techniques as they result in improved prediction than individual learners. Taking care of the imbalanced nature of data, ensemble techniques used in this thesis include the ensembles aggregating data resampling techniques with them. The performance of several bagging-based, boosting-based, and hybrid ensembles is evaluated. Amongst bagging-based ensembles, underbagging performed best with an average G-Mean of 73.14 and an average Balance of 72.77. In the category of Boosting based ensembles, RUSBoost outperformed with average G-Mean and average Balance 70.36 and 69.55 respectively. Both hybrid ensembles, EasyEnsemble, and BalanceCascade also found it effective to develop competent models with average G-Mean and Balance greater

than 69.00.

As the results of models developed using ML techniques and ensembles were found suitable, we also explored the use of SB techniques, a sub-category of ML techniques for predicting software maintainability. We completed a comparative empirical study with an effective experimental setup using multiple runs, use of firm performance metrics, and rigorous statistical evaluation, to construct SMP models using fourteen SB and fourteen ML fourteen datasets. Also, in this work, the imbalanced data is handled with data resampling techniques. The SMP models developed using the CHC technique, an SB technique obtained best results than all the other investigated techniques when assessed using the Friedman test on G-Mean and Balance results. The average G-Mean and Balance result of SMP models developed using the CHC technique on all the fourteen investigated datasets were 71.17 and 70.01 respectively. The successful results of the models developed using the CHC technique was because of its fitness function that analyzed the accuracy and the complexity of the ruleset. The pairwise comparisons of the models developed using the CHC technique and the other investigated techniques were evaluated using the Wilcoxon test. It was found that the models developed using the CHC technique were better than most of the other compared techniques. Also, in this work, the SafeSMOTE data resampling technique was advocated as the best data resampling technique.

As both ML and SB techniques were found effective, we further explored HB techniques for SMP. The HB techniques combined an ML technique with the SB technique. Combining ML techniques with SB resulted in effective models as HB techniques combine the characteristics and strengths of both of its constituent techniques. We examined the efficacy of eleven HB techniques after balancing datasets with data resampling techniques. Again, the consistent result was obtained in this work and the SafeSMOTE technique resulted in effective models with HB techniques. The SMP models developed using HB techniques after data resampling with SafeSMOTE

performed best with a median G-Mean of 73.11 and a median Balance of 72.21. The prediction models developed from GFS-LB and PSO-LDA were best performers in terms of their predictive power. The average values of the models developed using the PSOLDA technique were 70.22 and 70.47 respectively and the average values of the models developed using the GFS-LB technique were 70.77 and 70.00 respectively. These results were also statistically assessed by the use of the Wilcoxon test. The performance of GFS-LB and PSOLDA was was compared with other HB techniques and significant results have been obtained.

The systematic literature review conducted on software maintainability also pointed out the models constructed for predicting the maintainability of a project are trained using the past data of the same project. Sometimes, there is a lack of availability of the training dataset or it difficult to collect. Therefore, it was essential to examine the scenario where the historical datasets from one project can be validated on another project. For empirical evaluation using cross-project validation, we used three datasets Htmlunit, Maven, and Click. The prediction models are developed using the Htmlunit project dataset and Maven and Click datasets were used for validation. The prediction models were also developed using ten-fold cross-validation with Click and Maven datasets. The performance of the models developed using the two approaches (ten-fold and inter-project validation) was compared with the aid of the Wilcoxon signed-rank test. The results of the analysis indicated that models developed using ten-fold validation (i.e., trained and tested on the same project) and inter-project validation exhibited no significant difference in their performance. Therefore, these results advocated the applicability of inter-project validation in the domain of SMP.

As SafeSMOTE was found effective with ML, SB, and HB techniques, we proposed a novel data resampling technique (MSLSMOTE). The idea was to further improve the performance of the models by handling imbalanced data. The proposed technique is an enhanced version of SafeMSOTE that distinguishes between the $safe$

and $noisy$ datapoints more carefully with weighted K nearer neighbor algorithm and then generates synthetic data points according to Safe-Level of each data point. The imbalanced datasets are oversampled at a different rate of oversampling. SMP models are developed with ML techniques after oversampling data at different oversampling rates with MSLSMOTE. The performance of MSLSMOTE is compared with SafeSMOTE, BSMOTE, and SMOTE concerning G-Mean, Balance, and AUC. The MSLSMOTE techniques attained Friedman rank according to G-Mean, Balance, and AUC. Also, the Wilcoxon signed-rank test confirmed that MSLSMOTE is statistically better than SafeSMOTE, BSMOTE, and SMOTE.

10.2 Application of the Work

The work conducted in the thesis would aid the Software practitioners and researchers in the following ways:

- A well-defined and systematic approach for developing effective prediction models can be used for further experiments in SMP domain.
- As multiple techniques have been evaluated in this work to determine a technique that can provide effective balanced data to develop efficient SMP models, it will guide researchers to select an appropriate technique to deal with the imbalanced data problem.
- Predicting low maintainability classes in advance with the effective SMP models, the design engineers will have appropriate time to improve designs, including new plans or resources to get the recognized difficulties. Also, the efficient SMP models aid in the development of quality software which is reliable too.
- The efficient SMP models would also highlights for the designers, those areas

of poor maintainability which justify product improvement, modification, or a change of design.

- Software developers can strategically utilize the resources, enhance process efficiency, and optimize/reduce the associated maintenance costs. It will also enable the project managers to compare the productivity and cost amongst projects and keep the maintenance cost and effort under control.
- Software testers can devote extra time to the testing phase for testing the low maintainability classes that would lessen the chances of discovering faults in these classes during software maintenance.
- Regression testing effort will be reduced as the SMP models developed can be used in early phases for predicting low maintainable classes.

10.3 Future Work

The work conducted in this thesis evaluated wide array of classification techniques on open-source datasets for SMP with methods to handle imbalanced data. Future studies may plan to replicate this work on datasets belonging to other applications developed in other programming languages. The replication would increase in generalizability of the findings of this work. The ensemble techniques for imbalanced data problem may also be investigated further with SB techniques as base learners. Further, researchers may investigate SB and HB techniques to develop SMP models using inter-project validation as these techniques are found as efficient modelling techniques in this work. Inter-project validation with SB and HB techniques would aid in developing generalized models to predict software maintainability. The MSLSMOTE for learning from imbalanced data proposed in this thesis may be also be evaluated further by using

SB and HB techniques. The replication of the empirical investigation is important as it aid in addition of evidence based on which the application of work in the real-word scenario is ensured. Therefore, to obtain generalized conclusions and strengthen the findings, this work may be replicated.

Appendices

Descriptive Statistics of Datasets

The descriptive statistics of datasets used in this thesis as given below.

Descriptive Statistics Bcel dataset

Metric	Minimum	Maximum	Mean	Std. Dev.	Median
WMC	0	181	8.8	19.61	4
DIT	0	6	0.46	0.79	0
NOC	0	36	0.63	3.10	0
CBO	0	20	0.45	2.03	0
RFC	0	182	9.81	19.61	5
LCOM	0	16290	226.28	1590.12	6
Ca	0	13	0.24	1.41	0
Ce	0	9	0.22	1.06	0
NPM	0	181	7.43	19.10	3
LCOM3	1.01	2	1.68	0.42	2
SLOC	10	1086	54.29	119.31	24
DAM	0	1	0.36	0.47	0
MOA	0	121	0.69	6.47	0
MFA	0	1	0.03	0.162	0
CAM	0	1	0.5	0.25	0.5
IC	0	2	0.02	0.15	0
CBM	0	2	0.02	0.15	0
AMC	0	5	1.59	2.25	0

Descriptive Statistics Betwixt dataset

Metric	Minimum	Maximum	Mean	Std. Dev.	Median
WMC	1	55	7.2	7.27	5
DIT	0	4	0.72	0.50	1
NOC	0	38	0.26	2.33	0
CBO	0	39	2.68	4.40	1
RFC	1	63	10.36	8.80	8
LCOM	0	1485	42.19	157.35	6
Ca	0	38	1.32	3.67	0
Ce	0	11	1.44	1.83	1
NPM	1	47	6.43	5.92	5
LCOM3	0	2	1.11	0.70	1
SLOC	11	2760	66.78	177.90	36
DAM	0	1	0.63	0.47	1
MOA	0	7	0.39	0.84	0
MFA	0	1	0.0097	0.09	0
CAM	0.07	1	0.55	0.21	0.53
IC	0	1	0.043	0.20	0
CBM	0	1	0.043	0.20	0
AMC	0	304.78	5.50	18.64	4.58

Descriptive Statistics Io dataset

Metric	Minimum	Maximum	Mean	Std. Dev.	Median
WMC	2	101	11.51	13.69	8
DIT	0	4	0.75	0.90	1
NOC	0	19	0.25	1.52	0
CBO	0	20	1.14	1.82	1
RFC	3	102	12.50	13.69	9
LCOM	1	50	153.86	537.06	28
Ca	0	19	0.428	1.77	0
Ce	0	3	0.75	0.70	1
NPM	0	97	9.84	13.18	6
LCOM3	1.01	2	1.44	0.40	1.25
SLOC	18	61	71.06	83.45	52
DAM	0	1	0.57	0.47	1
MOA	0	3	0.27	0.68	0
MFA	0	1	0.11	0.25	0
CAM	0.08	1	0.49	0.20	0.45
IC	0	2	0.19	0.43	0
CBM	0	9	0.41	1.23	0
AMC	0	5	2.49	2.48	1.36

Descriptive Statistics Ivy dataset

Metric	Minimum	Maximum	Mean	Std. Dev.	Median
WMC	1	206	11.26	19.93	6
DIT	0	4	0.67	0.62	1
NOC	0	19	0.23	1.43	0
CBO	0	19	2.19	2.95	1
RFC	2	207	12.26	19.93	7
LCOM	0	21	255.88	1661.74	15
Ca	0	19	1.08	2.45	0
Ce	0	12	1.18	1.56	1
NPM	0	180	9.36	17.52	5
LCOM3	1	2	1.47	0.41	1.25
SLOC	96	1282	67.39	123.65	37
DAM	0	1	0.61	0.47	1
MOA	0	6	0.27	0.74	0
MFA	0	1	0.02	0.14	0
CAM	0.07	1	0.56	0.27	0.5
IC	0	2	0.06	0.24	0
CBM	0	2	0.06	0.24	0
AMC	0	5	2.7	2.39	5

Descriptive Statistics Jcs dataset

Metric	Minimum	Maximum	Mean	Std. Dev.	Median
WMC	0	39	8.07	7.44	5
DIT	0	4	0.89	0.48	1
NOC	0	7	0.05	0.49	0
CBO	0	7	0.85	0.95	1
RFC	0	40	9.05	7.47	6
LCOM	0	741	56.14	112.43	10
Ca	0	7	0.32	0.75	0
Ce	0	3	0.54	0.74	0
NPM	0	35	7.04	6.9	5
LCOM3	1.03	2	1.5	0.41	1.33
SLOC	13	247	44.62	44.94	30
DAM	0	1	0.51	0.47	0.5
MOA	0	3	0.21	0.52	0
MFA	0	0.95	0.02	0.12	0
CAM	0	1	0.49	0.25	0.44
IC	0	1	0.03	0.18	0
CBM	0	1	0.03	0.18	0
AMC	0	5	3.41	2.18	5

Descriptive Statistics Lang dataset

Metric	Minimum	Maximum	Mean	Std. Dev.	Median
WMC	1	175	20.61	25.38	14
DIT	0	5	0.66	0.86	1
NOC	0	5	0.1	0.59	0
CBO	0	8	1.37	1.18	2
RFC	2	176	21.61	25.38	15
LCOM	0	15225	522.97	1824.29	91
Ca	0	8	0.27	0.92	0
Ce	0	3	1.12	0.93	1
NPM	0	172	18.54	23.97	12
LCOM3	1.01	2	1.46	0.45	1.16
SLOC	62	1070	126.19	154.54	87
DAM	0	1	0.4	0.47	0
MOA	0	12	0.54	1.54	0
MFA	0	1	0.04	0.16	0
CAM	0.06	1	0.37	0.21	0.33
IC	0	2	0.12	0.34	0
CBM	0	3	0.18	0.52	0
AMC	0	5	2.51	2.45	4.28

Descriptive Statistics Log4j dataset

Metric	Minimum	Maximum	Mean	Std. Dev.	Median
WMC	1	104	7.92	8.48	5
DIT	0	6	1.12	1.08	1

NOC	0	4	0.11	0.47	0
CBO	0	12	1.16	1.74	1
RFC	2	105	8.92	8.48	6
LCOM	0	5356	63.3	283.04	10
Ca	0	11	0.53	1.48	0
Ce	0	5	0.64	0.85	0
NPM	0	48	6.1	6.25	4
LCOM3	1.01	2	1.49	0.4	1.33
SLOC	66	653	50.14	54.58	31
DAM	0	1	0.31	0.42	0
MOA	0	14	0.32	1.38	0
MFA	0	1	0.12	0.31	0
CAM	0	1	0.47	0.23	0.44
IC	0	2	0.08	0.31	0
CBM	0	3	0.11	0.46	0
AMC	0	5	3.83	2.02	5

Descriptive Statistics Ode dataset

Metric	Minimum	Maximum	Mean	Std. Dev.	Median
WMC	0	76	6.79	8.63	4
DIT	0	4	0.69	0.7	1
NOC	0	29	0.35	1.92	0
CBO	0	40	2.19	3.3	1
RFC	0	77	7.77	8.66	5
LCOM	0	2850	56.9	221.95	6
Ca	0	40	1.05	2.95	0
Ce	0	10	1.22	1.43	1
NPM	0	76	5.66	7.6	3
LCOM3	0	2	1.6	0.42	1.5
SLOC	10	456	40.23	52.77	24
DAM	0	1	0.5	0.47	0.5
MOA	0	8	0.32	0.87	0
MFA	0	1	0.04	0.18	0
CAM	0	1	0.58	0.29	0.58
IC	0	1	0.02	0.14	0
CBM	0	2	0.02	0.15	0

Descriptive Statistics HtmlUnit dataset

Metric	Minimum	Maximum	Mean	Std. Dev.
LCOM	0	100	21.69	36.822
CBO	1	593	12.34	26.924
DIT	0	10	2.64	1.944
NOC	0	454	1.81	20.503
WMC	1	13818	40.61	540.928
RFC	14	13913	144.11	546.551
SLOC	16	90956	361.1	3566.075

Descriptive Statistics Maven dataset

Metric	Minimum	Maximum	Mean	Std. Dev.
LCOM	0	100	47.12	35.178
CBO	0	78	12.88	10.926
DIT	0	3	0.32	0.637
NOC	0	6	0.2	0.719
WMC	0	2	05	20.513
RFC	13	265	31.21	40.381
SLOC	15	2552	158.63	237.772

Descriptive Statistics Click dataset

Metric	Minimum	Maximum	Mean	Std. Dev.
LCOM	0	100	52.47	38.45
DIT	0	5	1.43	1.07
CBO	0	72	10.94	9.34
NOC	0	25	0.58	2.25
WMC	0	97	10.39	15.4
RFC	13	208	64.21	46.28
SLOC	12	1421	115.5	183.03

Bibliography

[1] Y. Singh and R. Malhotra, *Object-oriented software engineering*. PHI Learning Pvt. Ltd., 2012.

[2] K. Erdil, E. Finn, K. Keating, J. Meattle, S. Park, and D. Yoon, "Software maintenance as part of the software life cycle," *Comp180: Software Engineering Project*, pp. 1–49, 2003.

[3] L. C. Briand, C. Bunse, J. W. Daly, and C. Differding, "An experimental comparison of the maintainability of object-oriented and structured design documents," *Empirical Software Engineering*, vol. 2, no. 3, pp. 291–312, 1997.

[4] I. S. S. Engineering, *IEEE Std. 828-1998 IEEE Standard for Software Configuration Management Plans standard*. Standards Committee of the IEEE Computer Society, 1998.

[5] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of systems and software*, vol. 23, no. 2, pp. 111–122, 1993.

[6] M. Dagpinar and J. H. Jahnke, "Predicting maintainability with object-oriented metrics-an empirical comparison," in *Proceedings of the 10th IEEE Working Conference on Reverse Engineering, WCRE*, 2003, pp. 155–170, Victoria, B.C., Canada.

[7] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," *Journal of systems and software*, vol. 80, no. 8, pp. 1349–1361, 2007.

[8] L.-j. Wang, X.-x. Hu, Z.-y. Ning, and W.-h. Ke, "Predicting object-oriented software maintainability using projection pursuit regression," in *2009 First International Conference on Information Science and Engineering*. IEEE, 2009, pp. 3827–3830.

[9] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Application of artificial neural network for predicting maintainability using object-oriented metrics," *Transactions on Engineering, Computing and Technology*, vol. 15, pp. 285–289, 2006.

[10] B. P. Lientz and E. B. Swanson, "Problems in application software maintenance," *Communications of the ACM*, vol. 24, no. 11, pp. pp. 763-769, 1981.

[11] A. G. Koru and H. Liu, "Identifying and characterizing change-prone classes in two large-scale open-source products," *Journal of Systems and Software*, vol. 80, no. 1, pp. 63–73, 2007.

[12] A. G. Koru and J. Tian, "Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products," *IEEE Transactions on Software Engineering*, vol. 31, no. 8, pp. 625–642, 2005.

[13] V. C. Koten and A. R. Gray, "An application of bayesian network for predicting object-oriented software maintainability," *Information and Software Technology*, vol. 48, no. 1, pp. pp. 59-67, 2006.

[14] M. M. Thwin and T. S. Quah, "Application of neural networks for software qual-

ity prediction using object-oriented metrics," *Journal of systems and software*, vol. 76, no. 2, pp. pp. 147-156, 2005.

[15] M. O. Elish and K. O. Elish, "Application of treenet in predicting object-oriented software maintainability: A comparative study," in *Proceedings of the 13th European Conference on Software Maintenance and Reengineering, CSMR*, 2009, pp. 69–78, Kaiserslautern, Germany.

[16] L. Kumar, S. Lal, and L. B. Murthy, "Estimation of maintainability parameters for object-oriented software using hybrid neural network and class level metrics," *International Journal of System Assurance Engineering and Management*, vol. 10, no. 5, pp. 1234–1264, 2019.

[17] R. Malhotra and M. Khanna, "An exploratory study for software change prediction in object-oriented systems using hybridized techniques," *Automated Software Engineering*, vol. 24, no. 3, pp. 673–717, 2017.

[18] Y. Zhou, H. Leung, and B. Xu, "Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 607–623, 2009.

[19] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis," *Empirical software engineering*, vol. 17, no. 3, pp. 200–242, 2012.

[20] M. O. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," *Journal of Software: Evolution and Process*, vol. 25, no. 5, pp. 407–437, 2013.

[21] D. Romano and M. Pinzger, "Using source code metrics to predict change-prone java interfaces," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 303–312.

[22] A. C. Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, "Cost sensitive credit card fraud detection using bayes minimum risk," in *2013 12th international conference on machine learning and applications*, vol. 1. IEEE, 2013, pp. 333–338.

[23] A. Abbasi and H. Chen, "A comparison of fraud cues and classification methods for fake escrow website detection," *Information Technology and Management*, vol. 10, no. 2-3, pp. 83–101, 2009.

[24] P. C. Lane, D. Clarke, and P. Hender, "On developing robust models for favourability analysis: Model choice, feature sets and imbalanced data," *Decision Support Systems*, vol. 53, no. 4, pp. 712–718, 2012.

[25] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263-1284, 2009.

[26] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, 2011.

[27] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Special issue on learning from imbalanced data sets," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 1–6, 2004.

[28] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "Smote: Synthetic mi-

nority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[29] B. Zadrozny and C. Elkan, "Learning and making decisions when costs and probabilities are both unknown," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 204-213.

[30] E. B. Swanson, "The dimensions of maintenance," in *Proceedings of the 2nd international conference on Software engineering, ICSE*, 1976, pp. 492–497, Estoril, Portugal.

[31] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[32] J. T. Nosek and P. Palvia, "Software maintenance management: changes in the last decade," *Journal of Software Maintenance: Research and Practice*, vol. 2, no. 3, pp. pp. 157-174, 1990.

[33] H. Halstead, *Elements of Software Science*. Elsevier North-Holland, ISBN 0-444-00205-7, 1977.

[34] J. T. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. pp. 308-320, 1976.

[35] H. D. Rombach, "Design measurement: Some lessons learned," *Software, IEEE*, vol. 7, no. 2, pp. pp. 17-25, 1990.

[36] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[37] W. Li, S. Henry, D. Kafura, and R. Schulman, "Measuring object-oriented design," *Journal of Object-Oriented Programming*, vol. 8, no. 4, pp. pp. 48-55, 1995.

[38] M. Lorenz and J. Kidd, *Object-oriented software metrics: A Practical Guide*. Prentice-Hall, Inc., 1994.

[39] F. B. e Abreu and W. Melo, "Evaluating the impact of object-oriented design on software quality," in *Software Metrics Symposium, 1996., Proceedings of the 3rd International*. IEEE, 1996, pp. 90–99.

[40] L. C. Briand, , C. Bunse, and J. Daly, "A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs," *IEEE Transactions on Software Engineering*, vol. 27, no. 6, pp. pp. 513-530, 2001.

[41] J. Bansiya and C. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. pp. 4-17, 2002.

[42] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Software Metrics Symposium, 1999. Proceedings. Sixth International*. IEEE, 1999, pp. 242–249.

[43] Y.-S. Lee, "Measuring the coupling and cohesion of an object-oriented program based on information flow," in *Proc. Int. Conf. on Software Quality, 1995*, 1995, pp. 81–90.

[44] J. M. Bieman and B.-K. Kang, "Cohesion and reuse in an object-oriented system," *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. SI, pp. 259–262, 1995.

[45] R. Malhotra and A. Chug, "Software maintainability prediction using machine learning algorithms," *Software Engineering: An International Journal (SEIJ)*, vol. 2, no. 2, pp. pp. 19-36, 2012.

[46] A. Kaur, K. Kaur, and R. Malhotra, "Soft computing approaches for prediction of software maintenance effort," *International Journal of Computer Applications*, vol. 1, no. 16, pp. pp. 80-86, 2010.

[47] V. Basili, L. Briand, and W. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transaction on Software Engineering*, vol. 22, no. 10, pp. pp. 751-761, 1996.

[48] F. Fioravanti and P. Nesi, "Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 27, no. 12, pp. pp. 1062–1084, 2001.

[49] J. Al Dallal, "Object-oriented class maintainability prediction using internal quality attributes," *Information and Software Technology*, vol. 55, no. 11, pp. 2028–2048, 2013.

[50] C. Jin and J. A. Liu, "Applications of support vector mathine and unsupervised learning for predicting maintainability using object-oriented metrics," in *Proceedings of the Second International Conference on Multimedia and Information Technology (MMIT)*, 2010, pp. 24–27, Kaifeng, China.

[51] S. C. Misra, "Modeling design/coding factors that drive maintainability of software systems," *Software Quality Journal*, vol. 13, no. 3, pp. pp. 297-320, 2005.

[52] M. Jorgensen, "Experience with the accuracy of software maintenance task

effort prediction models," *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. pp. 674-681, 1995.

[53] A. D. Lucia, E. Pompella, and S. Stefanucci, "Assessing effort estimation models for corrective maintenance through empirical studies," *Information and Software Technology*, vol. 47, no. 1, pp. pp. 3-15, 2005.

[54] W. Scacchi, "Understanding the requirements for developing open source software systems," *Software IEE Proceedings*, vol. 149, no. 1, pp. pp. 24-39, 2002.

[55] Y. Zhou and B. Xu, "Predicting the maintainability of open source software using design metrics," *Wuhan University Journal of Natural Sciences*, vol. 13, no. 1, pp. pp. 14-20, 2008.

[56] W. Zhang, L. Huang, V. Ng, and J. Ge, "Smplearner: learning to predict software maintainability," *Automated Software Engineering*, vol. 22, no. 1, pp. 111–141, 2015.

[57] A. Chug and R. Malhotra, "Benchmarking framework for maintainability prediction of open source software using object oriented metrics," *International Journal of Innovative Computing, Information and Control*, vol. 12, no. 2, pp. 615–634, 2016.

[58] F. Ramil, Juan, A. Lozano, M. Wermelinger, and A. Capiluppi, "Empirical studies of open source evolution," in *Book Chapter : Software evolution, Springer*, 2008, pp. 263–288.

[59] X. Wang, A. Gegov, F. Arabikhan, Y. Chen, and Q. Hu, "Fuzzy network based framework for software maintainability prediction," *International Journal of*

Uncertainty, Fuzziness and Knowledge Based Systems, vol. 27, no. 05, pp. 841–862, 2019.

[60] M. Schnappinger, M. H. Osman, A. Pretschner, and A. Fietzke, "Learning a classifier for prediction of maintainability based on static analysis tools," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 243–248.

[61] T. Back, D. B. Fogel, and Z. Michalewicz, *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

[62] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms," in *Proceedings of the third international conference on Genetic algorithms*, 1989, pp. 379–384, Virginia, USA.

[63] F. Li, R. Morgan, and D. Williams, "Hybrid genetic approaches to ramping rate constrained dynamic economic dispatch," *Electric Power Systems Research*, vol. 43, no. 2, pp. pp. 97-103, 1997.

[64] E. Alba, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. pp. 443-462, 2002.

[65] G. Balogh, A. Zoltan, and A. Baszedes, "Prediction of software development effort enhanced by a genetic algorithm," in *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 28–30, Bremen, Germany.

[66] L. Kumar and S. K. Rath, "Hybrid functional link artificial neural network approach for predicting maintainability of object-oriented software," *Journal of Systems and Software*, vol. 121, pp. 170–190, 2016.

[67] P. Sun and X. Wang, "Application of ant colony optimization in preventive software maintenance policy," in *Proceedings of the International Conference on Information Science and Technology (ICIST), Guangdong, China*, 2012, pp. 141–144, Guangdong, China.

[68] T. Choeikiwong and P. Vateekul, "Software defect prediction in imbalanced data sets using unbiased support vector machine," in *Information Science and Applications*. Springer, 2015, pp. 923–931.

[69] K. Gao, T. M. Khoshgoftaar, and A. Napolitano, "Combining feature subset selection and data sampling for coping with highly imbalanced software data." in *SEKE*, 2015, pp. 439–444.

[70] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, pp. 388–402, 2015.

[71] M. J. Siers and M. Z. Islam, "Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem," *Information Systems*, vol. 51, pp. 62–71, 2015.

[72] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *NAFIPS 2007-2007 Annual meeting of the North American fuzzy information processing society*. IEEE, 2007, pp. 69–72.

[73] Z. Sun, Q. Song, and X. Zhu, "Using coding-based ensemble learning to improve software defect prediction," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1806–1817, 2012.

[74] S. Wang and X. Yao, "Using class imbalance learning for software prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[75] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4537–4543, 2010.

[76] R. Malhotra and M. Khanna, "An empirical study for software change prediction using imbalanced data," *Empirical Software Engineering*, vol. 22, no. 6, pp. 2806–2851, 2017.

[77] K. K. Aggarwal and Y. Singh, "Software engineering programs documentation, operating procedures," *New Age international publishers*, 2008.

[78] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM, Orlando*, 2009, pp. 367–377, Florida, USA.

[79] F. Calzolari, P. Tonella, and G. Antonioli, "Dynamic model for maintenance and testing effort," in *Proceedings of the International Conference on Software Maintenance, ICSM*, 1998, pp. 104–112, Maryland, USA.

[80] S. Ghosh, A. Rana, and A. Kumar, "Comparative study of the factors that affect maintainability," *International Journal on Computer Science and Engineering*, vol. 3, no. 12, pp. pp. 3763-3769, 2011.

[81] J. Saraiva, "A roadmap for software maintainability measurement," in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 1453–1455 , San Francisco, CA, USA.

[82] S. Gupta and A. Chug, "Software maintainability prediction of open source datasets using least squares support vector machines," *Journal of Statistics and Management Systems*, pp. 1–11, 2020.

[83] S. Gupta and A. Chug, "Software maintainability prediction using an enhanced random forest algorithm," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 23, no. 2, pp. 441–449, 2020.

[84] K. El Emam, W. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," *Journal of systems and software*, vol. 56, no. 1, pp. 63–75, 2001.

[85] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[86] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995.

[87] Y. Singh, A. Kaur, and R. Malhotra, "Empirical validation of object-oriented metrics for predicting fault proneness models," *Software quality journal*, vol. 18, no. 1, p. 3, 2010.

[88] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *IEEE Transactions on software Engineering*, vol. 33, no. 6, pp. 402–419, 2007.

[89] S. Kpodjedo, F. Ricca, P. Galinier, Y.-G. Guéhéneuc, and G. Antoniol, "Design

evolution metrics for defect prediction in object oriented systems," *Empirical Software Engineering*, vol. 16, no. 1, pp. 141–175, 2011.

[90] M. O. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," *Journal of Software: Evolution and Process*, vol. 25, no. 5, pp. 407–437, 2013.

[91] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and software technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

[92] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis," *Empirical software engineering*, vol. 17, no. 3, pp. 200–242, 2012.

[93] S. Eski and F. Buzluca, "An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2011, pp. 566–571.

[94] S. Haykin, "A comprehensive foundation: Neural networks," 1999.

[95] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[96] D. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 11, pp. 321–355, 1988.

[97] J. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kauffman, 1993.

[98] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Chapman and Hall (Wadsworth and Inc.), 1984.

[99] R. Rastogi and K. Shim, "Public: A decision tree classifier that integrates building and pruning," *Data Mining and Knowledge Discovery*, vol. 4, no. 4, pp. 315–344, 2000.

[100] Y. Zhao and Y. Zhang, "Comparison of decision tree methods for finding active objects," *Advances in Space Research*, vol. 41, no. 12, pp. 1955–1959, 2008.

[101] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Education India, 2016.

[102] W. Cohen, "Fast effective rule induction," in *Machine Learning: Proceedings of the Twelfth International Conference*, 1995, pp. 1–10.

[103] E. Frank and I. Witten, "Generating accurate rule sets without global optimization," in *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998, pp. 144–151.

[104] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[105] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[106] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*. John Wiley & Sons, 2013, vol. 398.

[107] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[108] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[109] J. Cleary and L. Trigg, "K*: An instance-based learner using an entropic distance measure," in *Proceedings of the 12th International Conference on Machine Learning*, 1995, pp. 108–114.

[110] D. Wettschereck and T. Dietterich, "An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms," *Machine Learning*, vol. 19, pp. 5–27, 1995.

[111] S. Salzberg, "A nearest hyperrectangle learning method," *Machine Learning*, vol. 6, pp. 251–276, 1991.

[112] P. Domingos, "Unifying instance-based and rule-based induction," *Machine Learning*, vol. 24, no. 2, pp. 141–168, 1996.

[113] J. Cano, F. Herrera, and M. Lozano, "Using evolutionary algorithms as instance selection for data reduction in kdd: An experimental study," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 6, pp. 561–575, 2003.

[114] S. Ho, C. Liu, and S. Liu, "Design of an optimal nearest neighbor classifier using an intelligent genetic algorithm," *Pattern Recognition Letters*, vol. 23, pp. 1495–1503, 2002.

[115] S. García, J. Cano, and F. Herrera, "A memetic algorithm for evolutionary prototype selection: A scaling up approach," *Pattern Recognition*, vol. 41, no. 8, pp. 2693–2709, 2008.

[116] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, "Evolutionary learning of hierarchical decision rules," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 33, no. 2, pp. 324–331, 2003.

[117] J. Bacardit and J. Garrell, "Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system," in *Genetic and Evolutionary Computation Conference(GECCO'03)*, ser. Lecture Notes on Computer Science, vol. 2724. Lecture Notes on Computer Science, 2003, pp. 1818–1831.

[118] J. Bacardit and J. Garrell, "Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system," in *Advances at the frontier of Learning Classifier Systems*, ser. Lecture Notes on Computer Science, vol. 4399. Springer Berlin-Heidelberg, 2007, pp. 61–80.

[119] S. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[120] E. Bernadó-Mansilla and J. Garrell, "Accuracy-based learning classifier systems: Models and analysis and applications to classification tasks," *Evolutionary Computation*, vol. 11, no. 3, pp. 209–238, 2003.

[121] J. Bacardit, E. Burke, and N. Krasnogor, "Improving the scalability of rule-based evolutionary learning," *Memetic computing*, vol. 1, no. 1, pp. 55–67, 2009.

[122] J. Bacardit and N. Krasnogor, "Performance and efficiency of memetic pittsburgh learning classifier systems," *Evolutionary Computation*, vol. 17, no. 3, pp. 307–342, 2009.

[123] D. Carvalho and A. Freitas, "A hybrid decision tree/genetic algorithm method for data mining," *Information Sciences*, vol. 163, no. 1, pp. 13–35, 2004.

[124] J. Gray and G. Fan, "Classification tree analysis using target," *Computational Statistics and Data Analysis*, vol. 52, no. 3, pp. 1362–1372, 2008.

[125] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. Ieee, 1995, pp. 39–43.

[126] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[127] T. Sousa, A. Silva, and A. Neves, "Particle swarm based data mining algorithms for classification tasks," *Parallel Computing*, vol. 30, pp. 767–783, 2004.

[128] S. Lin and S. Chen, "Psolda: A particle swarm optimization approach for enhancing classification accuracy rate of linear discriminant analysis," *Applied Soft Computing*, vol. 9, pp. 1008–1015, 2009.

[129] J. Otero and L. Sánchez, "Induction of descriptive fuzzy classifiers with the logitboost algorithm," *Soft Computing*, vol. 10, no. 9, pp. 825–835, 2006.

[130] M. del Jesus, F. Hoffmann, L. Junco, and L. Sánchez, "Induction of fuzzy-rule-based classifiers with evolutionary boosting algorithms," *IEEE Transactions on Fuzzy Systems*, vol. 12, no. 3, pp. 296–308, 2004.

[131] L. Sánchez and J. Otero, "Boosting fuzzy rules in classification problems under single-winner inference," *International Journal of Intelligent Systems*, vol. 22, no. 9, pp. 1021–1034, 2007.

[132] L. Sánchez, I. Couso, and J. Corrales, "Combining gp operators with sa search to evolve fuzzy rule based classifiers," *Information Sciences*, vol. 136, no. 1-4, pp. 175–192, 2001.

[133] H. Ishibuchi, T. Nakashima, and T. Murata, "Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems," *IEEE Transactions on Systems and Man and and Cybernetics and Part B: Cybernetics*, vol. 29, no. 5, pp. 601–618, 1999.

[134] F. Berlanga, A. Rivera, M. del Jesus, and F. Herrera, "Gp-coach: Genetic programming based learning of compact and accurate fuzzy rule based classification systems for high dimensional problems," *Information Sciences*, vol. 180, no. 8, pp. 1183–1200, 2010.

[135] R. Malhotra, N. Pritam, K. Nagpal, and P. Upmanyu, "Defect collection and reporting system for git based open source software," in *2014 International Conference on Data Mining and Intelligent Computing (ICDMIC)*. IEEE, 2014, pp. 1–7.

[136] M. A. Hall, "Correlation-based feature selection for machine learning," 1999.

[137] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data engineering*, vol. 15, no. 6, pp. 1437–1447, 2003.

[138] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[139] 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.

[140] A. B. De Carvalho, A. Pozo, and S. R. Vergilio, "A symbolic fault-prediction model based on multiobjective particle swarm optimization," *Journal of Systems and Software*, vol. 83, no. 5, pp. 868–882, 2010.

[141] R. Malhotra and M. Khanna, "Investigation of relationship between object-oriented metrics and change proneness," *International Journal of Machine Learning and Cybernetics*, vol. 4, no. 4, pp. 273–286, 2013.

[142] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[143] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, no. 2, pp. 167–199, 2012.

[144] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 409–418.

[145] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[146] K. Gao, T. M. Khoshgoftaar, and A. Napolitano, "Combining feature subset selection and data sampling for coping with highly imbalanced software data." in *SEKE*, 2015, pp. 439–444.

[147] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to" comments on'data mining static code attributes to learn defect predictors'"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 637–640, 2007.

[148] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and

novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[149] R. Shatnawi, "Improving software fault-prediction for imbalanced data," in *2012 international conference on innovations in information technology (IIT)*. IEEE, 2012, pp. 54–59.

[150] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86–92, 1940.

[151] D. W. Zimmerman and B. D. Zumbo, "Relative power of the wilcoxon test, the friedman test, and repeated-measures anova on ranks," *The Journal of Experimental Education*, vol. 62, no. 1, pp. 75–86, 1993.

[152] R. Malhotra and K. Lata, "A systematic literature review on empirical studies towards prediction of software maintainability."

[153] B. A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-based software engineering and systematic reviews*. CRC press, 2015, vol. 4.

[154] S. O. Olatunji and A. Ajasin, "Sensitivity-based linear learning method and extreme learning machines compared for software maintainability prediction of object-oriented software systems," *ICTACT Journal On Soft Computing*, vol. 3, no. 03, 2013.

[155] T. M. Khoshgoftaar and R. M. Szabo, "Improving code churn predictions during the system test and maintenance phases." in *ICSM*, vol. 94, 1994, pp. 58–67.

[156] T. M. Khoshgoftaar, "Improving neural network predictions of software quality using principal components analysis," in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, vol. 5. IEEE, 1994, pp. 3295–3300.

[157] R. Malhotra and A. Chug, "Application of group method of data handling model for software maintainability prediction using object oriented systems," *International Journal of System Assurance Engineering and Management*, vol. 5, no. 2, pp. 165–173, 2014.

[158] M. O. Elish, H. Aljamaan, and I. Ahmad, "Three empirical studies on predicting software maintainability using ensemble methods," *Soft Computing*, vol. 19, no. 9, pp. 2511–2524, 2015.

[159] S. Manchanda and A. Chug, "Cfs based feature subset selection for software maintenance prediction," *International Journal of Advance Foundation and Research in Computer (IJAFRC)*, vol. 2, 2015.

[160] S. Mishra and A. Sharma, "Maintainability prediction of object oriented software by using adaptive network based fuzzy system technique," *International Journal of Computer Applications*, vol. 119, no. 9, 2015.

[161] L. Kumar and S. K. Rath, "Neuro–genetic approach for predicting maintainability using chidamber and kemerer software metrics suite," in *Recent Advances in Information and Communication Technology 2015*. Springer, 2015, pp. 31–40.

[162] H. Sharma and A. Chug, "Dynamic metrics are superior than static metrics in maintainability prediction: An empirical case study," in *2015 4th Inter-*

national Conference on Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions). IEEE, 2015, pp. 1–6.

[163] L. Kumar, D. K. Naik, and S. K. Rath, "Validating the effectiveness of object-oriented metrics for predicting maintainability," *Procedia Computer Science*, vol. 57, pp. 798–806, 2015.

[164] L. Kumar and S. K. Rath, "Predicting object-oriented software maintainability using hybrid neural network with parallel computing concept," in *Proceedings of the 8th India software engineering conference*, 2015, pp. 100–109.

[165] D. Chandra, "Support vector approach by using radial kernel function for prediction of software maintenance effort on the basis of multivariate approach," *International Journal of Computer Applications*, vol. 51, no. 4, 2012.

[166] S. Tarwani and A. Chug, "Predicting maintainability of open source software using gene expression programming and bad smells," in *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2016, pp. 452–459.

[167] S. K. Dubey, A. Rana, and Y. Dash, "Maintainability prediction of object-oriented software system by multilayer perceptron model," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 5, pp. 1–4, 2012.

[168] R. Malhotra and K. Lata, "An exploratory study for predicting maintenance effort using hybridized techniques," in *Proceedings of the 10th Innovations in Software Engineering Conference*, 2017, pp. 26–33.

[169] S. S, "Software maintainability prediction using neural networks," *Int J Eng Res Appl (IJERA)*, vol. 2, no. 2, pp. 750–755, 2012.

[170] L. Kumar and S. K. Rath, "Software maintainability prediction using hybrid neural network and fuzzy logic approach with parallel computing concept," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 2, pp. 1487–1502, 2017.

[171] M. A. Ahmed and H. A. Al-Jamimi, "Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model," *IET software*, vol. 7, no. 6, pp. 317–326, 2013.

[172] R. Malhotra and K. Lata, "On the application of cross-project validation for predicting maintainability of open source software using machine learning techniques," in *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2018, pp. 175–181.

[173] A. Kaur and K. Kaur, "Statistical comparison of modelling methods for software maintainability prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 06, pp. 743–774, 2013.

[174] S. Bhutani and A. Chug, "Software maintainability prediction using hybrid neural network and fuzzy logic approach with parallel computing concept," *International Journal of Computer Science and Communication Networks*, vol. 5, no. 2, pp. 92–95, 2015.

[175] A. Jain, S. Tarwani, and A. Chug, "An empirical investigation of evolutionary algorithm for software maintainability prediction," in *2016 IEEE Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*. IEEE, 2016, pp. 1–6.

[176] V. Kumar, R. Kumar, and A. Sharma, "Maintainability prediction from project

metrics data analysis using artificial neural network: an interdisciplinary study," *Middle-East J. Sci. Res*, vol. 19, no. 10, pp. 1412–1420, 2014.

[177] S. D. Conte, H. E. Dunsmore, and Y. Shen, *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., 1986.

[178] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Information and Software Technology*, vol. 54, no. 1, pp. 41–59, 2012.

[179] D. T. Campbell and J. C. Stanley, *Experimental and quasi-experimental designs for research*. Ravenio Books, 2015.

[180] M. Kubat, R. C. Holte, and S. Matwin, "Machine learning for the detection of oil spills in satellite radar images," *Machine learning*, vol. 30, no. 2-3, pp. 195–215, 1998.

[181] T. Fawcett and F. Provost, "Adaptive fraud detection," *Data mining and knowledge discovery*, vol. 1, no. 3, pp. 291–316, 1997.

[182] P. L. Braga, A. L. Oliveira, and S. R. Meira, "Software effort estimation using machine learning techniques with robust confidence intervals," in *7th international conference on hybrid intelligent systems (HIS 2007)*. IEEE, 2007, pp. 352–357.

[183] G. Catolino and F. Ferrucci, "An extensive evaluation of ensemble techniques for software change prediction," *Journal of Software: Evolution and Process*, vol. 31, no. 9, p. e2156, 2019.

[184] R. Malhotra and K. Lata, "An empirical study on predictability of software maintainability using imbalanced data," *Software Quality Journal*, pp. 1–34, 2020.

[185] S. Kotsiantis, D. Kanellopoulos, P. Pintelas *et al.*, "Handling imbalanced datasets: A review," *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pp. 25–36, 2006.

[186] H. Han, W. Wang, and B. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *2005 International Conference on Intelligent Computing(ICIC05)*, ser. Lecture Notes on Computer Science, vol. 3644. Springer-Verlag, 2005, pp. 878–887.

[187] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining(PAKDD09)*, ser. Lecture Notes on Computer Science, vol. 5476. Springer-Verlag, 2009, pp. 475–482.

[188] H. He, Y. Bai, E. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 International Joint Conference on Neural Networks(IJCNN08)*, 2008, pp. 1322–1328.

[189] G. Batista, R. Prati, and M. Monard, "A study of the behavior of several methods for balancing machine learning training data," *SIGKDD Explorations*, vol. 6, no. 1, pp. 20–29, 2004.

[190] J. Stefanowski and S. Wilk, "Selective pre-processing of imbalanced data for improving classification performance," in *10th International Conference in Data Warehousing and Knowledge Discovery(DaWaK2008)*, ser. Lecture Notes on Computer Science, vol. 5182. Springer, 2008, pp. 283–292.

[191] K. Napierala, J. Stefanowski, and S. Wilk, "Learning from imbalanced data in presence of noisy and borderline examples," in *7th International Conference*

on Rough Sets and Current Trends in Computing(RSCTC2010), 2010, pp. 158–167.

[192] P. Hart, "The condensed nearest neighbour rule," *IEEE Transactions on Information Theory*, vol. 14, no. 5, pp. 515–516, 1968.

[193] J. Laurikkala, "Improving identification of difficult small classes by balancing class distribution," in *8th Conference on AI in Medicine in Europe(AIME01)*, ser. Lecture Notes on Computer Science, vol. 2001. Springer Berlin / Heidelberg, 2001, pp. 63–66.

[194] K. Yoon and S. Kwek, "An unsupervised learning approach to resolving the data imbalanced issue in supervised learning problems in functional genomics," in *5th International Conference on Hybrid Intelligent Systems(HIS05)*, 2005, pp. 303–308.

[195] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, 2017.

[196] P. Zhang, X. Zhu, J. Tan, and L. Guo, "Classifier and cluster ensembles for mining concept drifting data streams," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 1175–1180.

[197] A. Topchy, B. Minaei-Bidgoli, A. K. Jain, and W. F. Punch, "Adaptive clustering ensembles," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 1. IEEE, 2004, pp. 272–275.

[198] S. Wang and X. Yao, "Diversity analysis on imbalanced data sets by using ensemble models," in *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*. IEEE, 2009, pp. 324–331.

[199] C. W. Yohannese, T. Li, M. Simfukwe, and F. Khurshid, "Ensembles based combined learning for improved software fault prediction: A comparative study," in *Intelligent Systems and Knowledge Engineering (ISKE), 2017 12th International Conference on*. IEEE, 2017, pp. 1–6.

[200] R. Malhotra and K. Lata, "Using ensembles for class-imbalance problem to predict maintainability of open source software," *International Journal of Reliability, Quality and Safety Engineering*, p. 2040011, 2020.

[201] R. Malhotra and K. Lata, "Tackling the imbalanced data in software maintainability prediction using ensembles for class imbalance problem," in *In International Conference on Recent Trends in Engineering, Technology and Business Management (ICRTETBM)*. Springer, 2019.

[202] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "Smoteboost: Improving prediction of the minority class in boosting," in *European conference on principles of data mining and knowledge discovery*. Springer, 2003, pp. 107–119.

[203] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 40, no. 1, pp. 185–197, 2010.

[204] S. Hu, Y. Liang, L. Ma, and Y. He, "Msmote: improving classification performance when training data is imbalanced," in *Computer Science and Engineering, 2009. WCSE'09. Second International Workshop on*, vol. 2. IEEE, 2009, pp. 13–17.

[205] H. Guo and H. L. Viktor, "Learning from imbalanced data sets with boosting

and data generation: the databoost-im approach," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 30–39, 2004.

[206] M. Galar, A. Fernández, E. Barrenechea, and F. Herrera, "Eusboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling," *Pattern Recognition*, vol. 46, no. 12, pp. 3460–3471, 2013.

[207] J. Błaszczyński, M. Deckert, J. Stefanowski, and S. Wilk, "Integrating selective pre-processing of imbalanced data with ivotes ensemble," in *International Conference on Rough Sets and Current Trends in Computing*. Springer, 2010, pp. 148–157.

[208] C. Drummond, R. C. Holte *et al.*, "C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," in *Workshop on learning from imbalanced datasets II*, vol. 11. Citeseer, 2003, pp. 1–8.

[209] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems and Man and and Cybernetics and Part B*, vol. 39, no. 2, pp. 539–550, 2009.

[210] S. Hosseini, B. Turhan, and M. Mäntylä, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction," *Information and Software Technology*, vol. 95, pp. 296–312, 2018.

[211] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *Empirical software engineering and verification*. Springer, 2010, pp. 1–59.

[212] M. Harman and B. F. Jones, "Search-based software engineering," *Information and software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[213] M. Harman, "The relationship between search based software engineering and predictive modeling," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010, pp. 1–13.

[214] M. Harman and J. Clark, "Metrics are fitness functions too," in *10th International Symposium on Software Metrics, 2004. Proceedings.* Ieee, 2004, pp. 58–69.

[215] R. Malhotra, M. Khanna, and R. R. Raje, "On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions," *Swarm and Evolutionary Computation*, vol. 32, pp. 85–109, 2017.

[216] R. Malhotra and K. Lata, "An empirical study to investigate the impact of data resampling techniques on the performance of class maintainability prediction models," *Neurocomputing*, 2020.

[217] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 742–762, 2009.

[218] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Genetic programming for effort estimation: an analysis of the impact of different fitness functions," in *2nd International Symposium on Search Based Software Engineering*. IEEE, 2010, pp. 89–98.

[219] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013.

[220] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–61, 2012.

[221] C. Grosan and A. Abraham, "Hybrid evolutionary algorithms: methodologies, architectures, and reviews," in *Hybrid evolutionary algorithms*. Springer, 2007, pp. 1–17.

[222] R. Malhotra and K. Lata, "Using hybridized techniques for prediction of software maintainability using imbalanced data," in *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, 2020, pp. 787–792.

[223] R. Malhotra and K. Lata, "Improving software maintainability predictions using data oversampling and hybridized techniques," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–7.

[224] R. Malhotra and K. Lata, "Modified safe level synthetic minority oversampling techniques for software maintainability prediction," *IEEE Transactions of Emerging Topics in Computational Intelligence*, 2020.

[225] R. Malhotra and M. Khanna, "The ability of search-based algorithms to predict change-prone classes," *Software Quality Professional*, vol. 17, no. 1, p. 17, 2014.

[226] B. A. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 316–329, 2007.

[227] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value

of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[228] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, 2013, pp. 252–261.

[229] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 91–100.

[230] R. Malhotra, V. Gupta, and M. Khanna, "Applicability of inter project validation for determination of change prone classes," *Int. J. Comput. Appl*, pp. 0975–8887, 2014.

[231] R. Malhotra and K. Lata, "On the application of cross-project validation for predicting maintainability of open source software using machine learning techniques," in *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2018, pp. 175–181.

Supervisor's Biography

Dr. Ruchika Malhotra
Associate Head & Associate Professor
Discipline of Software Engineering
Department of Computer Science & Engineering
Delhi Technological University
Email: ruchikamalhotra@dtu.ac.in

Educational Qualifications:

Postdoc (Indiana University-Purdue University Indianapolis, USA), Ph.D (Computer Applications)

Ruchika Malhotra is an Associate Professor in Discipline of Software Engineering, Department of Computer Science Engineering, Delhi Technological University, Delhi, India. She is Associate Dean in Industrial Research and Development, Delhi Technological University. She has been awarded prestigious UGC Raman Postdoctoral Fellowship by the Indian government for pursuing postdoctoral research from the Department of Computer and Information Science, Indiana University-Purdue University, Indianapolis, USA. She was an Assistant Professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She has received IBM Faculty Award 2013. She is the recipient of Commendable Research Award by Delhi Technological University for her research in the year 2017, 2018 and 2019. She is the author of book titled "Empirical Research in Software Engineering" published by CRC press and co-author of a book on "Object Oriented Software Engineering" published by PHI Learning. Her research interests are in software testing, improving software quality, statistical and adaptive prediction models, software metrics and the definition and validation of software metrics. She has published more than 190 research papers in international journals and conferences. Her h-index is 28 as reported by Google Scholar.

Author's Biography

Kusum Lata
Research Scholar
Department of Computer Science & Engineering
Assistant Professor
University School of Management and Entrepreneurship
Delhi Technological University
Email: er.kusum82@gmail.com

Educational Qualifications:
M.E (CTA), B.Tech (CSE)

Kusum Lata is currently pursuing her doctoral degree from Delhi Technological University. She is currently working as Assistant Professor in University School of Management and Entrepreneurship, Delhi Technological University. She completed her master's degree in Computer Technology and Applications in 2010 from the Delhi College of Engineering, University of Delhi, India. She received her bachelor degree in Computer Science and Engineering in 2003 from Beant College of Engineering and Technology, Punjab Technical University, India. Her research interests are in software quality improvement, validation of software metrics and predictive modeling in software engineering domain.