A Comparative Analysis of Various Sampling Methods and MetaCost Learners to Improve Software Defect Prediction for Imbalanced Data

A project report submitted as a part of Major-II in the partial fulfilment of the requirement for the award of the degree

Of

Master of Technology in Software Engineering

By

Shine Kamal
2K15/SWE/16

Under the Guidance of:
Dr. Ruchika Malhotra
(Assistant Professor, Department of CSE)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

2015-2017

DELHI TECHNOLOGICAL UNIVERSITY

CERTIFICATE

This is to certify that the project report entitled "**A Comparative Analysis of Various Sampling Methods and Metacost Learners to Improve Software Defect Prediction for Imbalanced Data.**" is a bonafide record of the work carried out by **Shine Kamal (roll no. 2K15/SWE/16)** under my guidance and supervision during the academic session 2015-2017 in the partial fulfilment of the requirement for the award of degree of Master of Technology in Software Engineering from Delhi Technological University, Delhi.

To the best of my knowledge, the matter incorporated in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Dr. Ruchika Malhotra
Assistant Professor
Department of Software Engineering
Delhi Technological University
Delhi

ACKNOWLEDGEMENT

With due respect, I hereby take this opportunity to acknowledge the people who have supported me with their words and deeds in completion of my research work as a part of this course of Master of Technology in Software Engineering.

First of all I would like to thank the almighty, who has always guided me to work on the right path of the life. My greatest thanks to my parents who bestowed ability and strength in me to complete this work.

I owe a profound gratitude to my project guide, my motivator, **Dr. Ruchika Malhotra**, Assistant Professor, Department of Software Engineering, Delhi Technological University, who has been a constant source of inspiration to me throughout the period of this project. It was her competent guidance, constant encouragement and critical evaluation that helped me to develop a new insight into my project. Her calm, collected and professionally impeccable style of handling situations not only steered me through every problem, but also helped me to grow as a matured person. I am also thankful to her for trusting my capabilities to develop this project under her guidance.

Last but not the least, I would like to thank all the people who were directly or indirectly involved in the successful completion of my project.

SHINE KAMAL

Roll No. 2K15/SWE/16

TABLE OF CONTENTS

Certificate………………………………………………………………………………... i

Acknowledgement…………………………………………………………………….. ii

Table of Contents……………………………………………………………………... iii-iv

List of Tables………………………………………………………………………….. v

List of Figures…………………………………………………………………………vi

Abstract………………………………………………………………………………... vii

Chapter 1: Introduction………………………………………………………………… 1-7

1.1 Introduction……………………………………………………………………… 1-2

1.1.1 Software Metrics………………………………………………………… 2-3

1.1.2 Imbalanced Data Problem……………………………………………….. 3-4

1.2 Motivation of the work…………………………………………………………... 4-5

1.3 Goals of the Study……………………………………………………………….. 5-6

1.4 Thesis Organization…………………………………………………………........... 6-7

Chapter 2: Related Work………………………………………………………………..8-10

2.1 Defect Prediction Studies………………………………………………………… 8-9

2.2 Imbalanced Data Related Studies……………………………………………….... 9-10

Chapter 3: Experimental Design……………………………………………………….. 11-15

3.1 Dependent and Independent Variables…………………………………………... 11

3.2 Data Collection………………………………………………………………….... 11-13

3.3 Selection of Performance Measures……………………………………………… 13

3.4 Statistical Test Selection………………………………………………………… 14-15

3.5 Model Evaluation………………………………………………………………… 15

Chapter 4: Research Methodology……………………………………………………...16-26

4.1 Data Sampling Methods…………………………………………………………..16-25

4.1.1 SMOTE……………………………………………………………………... 16-17

4.1.2 Safe-Level-SMOTE……………………………………………………… 17-18

4.1.3 ADASYN………………………………………………………………… 18-19

4.1.4 SPIDER……………………………………………………………………... 19-21

4.1.5 SPIDER2…………………………………………………………………21-22

4.1.6 SPIDER3……………………………………………………………………22-24
4.1.7 MUTE…………………………………………………………………… 24
4.1.8 SPY……………………………………………………………………… 24
4.1.9 SpreadSubSample……………………………………………………….. 25
4.1.10 Resample…………………………………………………………………..25
4.2 MetaCost Learners……………………………………………………………….. 25
4.3 Machine Learning Classifiers…………………………………………………… 26
4.3.1 J48………………………………………………………………………... 26
4.3.2 Random Forest…………………………………………………………… 26
4.3.3 Naïve Bayes……………………………………………………………… 26
4.3.4 AdaboostM1……………………………………………………………….26
4.3.5 Bagging……………………………………………………………………... 26

Chapter 5: Empirical Results and Analysis…………………………………………. 27-55
5.1 Result Analysis for Oversampling Methods...………………………………… 27-40
5.2 Comparison of Oversampling Methods..……………………………………… 40-43
5.2.1 Friedman Test Analysis using AUC for Oversampling Methods……………... 41-42
5.2.2 Friedman Test Analysis using Sensitivity for Oversampling Methods……….. 42
5.2.3 Friedman Test Analysis using Precision for Oversampling Methods…………42-43
5.3 SPIDER2 vs SPIDER3……………………………………………………………43-44
5.4 Comparison of Undersampling and Resampling Methods……………………… 44-47
5.4.1 Friedman Test Analysis using AUC……………………………………... 45
5.4.2 Friedman Test Analysis using Sensitivity………………………………...45-46
5.4.3 Friedman Test Analysis using Precision…………………………………46-47
5.5 Comparison among Data Sampling Methods..…………………………………47-48
5.6 Result Analysis for MetaCost Learners..………………………………………. 48-55
5.7 Best Sampling Method vs MetaCost Learners…………………………………55

Chapter 6: Conclusion…………………………………………………………………..56-57
6.1 The Conclusions of the Work………………………………………………….. 56-57
6.2 Future Scope……………………………………………………………………... 57

References…………………………………………………………………………… 58-63

LIST OF TABLES

3.1 Static Code Metrics Description in NASA Datasets……………………………………12
3.2 Confusion Matrix…………………………………………………………………….. 14
3.3 Performance Metrics………………………………………………………………… 14
4.1 Parameter Selection for Sampling Algorithms………………………………………..17
5.1 Results for CM1 Dataset…………………………………………………………... 28
5.2 Results for JM1 Dataset…………………………………………………………..... 29
5.3 Results for KC2 Dataset…………………………………………………………..... 30
5.4 Results for KC3 Dataset…………………………………………………………..... 31
5.5 Results for MC1 Dataset…………………………………………………………... 32
5.6 Results for MC2 Dataset…………………………………………………………... 33
5.7 Results for MW1 Dataset………………………………………………………….. 34
5.8 Results for PC1 Dataset…………………………………………………………..... 35
5.9 Results for PC2 Dataset…………………………………………………………..... 36
5.10 Results for PC3 Dataset………………………………………………………….....37
5.11 Results for PC4 Dataset………………………………………………………….....38
5.12 Results for PC5 Dataset………………………………………………………….....39
5.13 Friedman Results using AUC for Oversampling Methods…………………………… 41
5.14 Friedman Results using Sensitivity for Oversampling Methods……………………… 42
5.15 Friedman Results using Precision for Oversampling Methods………………………...43
5.16 Friedman Results using AUC…………………………………………………………..45
5.17 Friedman Results using Sensitivity……………………………………………………46
5.18 Friedman Results using Precision…………………………………………………... 46
5.19 MC results for CM1 Dataset………………………………………………………... 49
5.20 MC results for JM1 Dataset…………………………………………………………49-50
5.21 MC results for KC2 Dataset………………………………………………………… 50
5.22 MC results for KC3 Dataset………………………………………………………… 50-51
5.23 MC results for MC1 Dataset………………………………………………………... 51
5.24 MC results for MC2 Dataset………………………………………………………... 51-52
5.25 MC results for MW1 Dataset……………………………………………………….. 52
5.26 MC results for PC1 Dataset…………………………………………………………52-53
5.27 MC results for PC2 Dataset…………………………………………………………53
5.28 MC results for PC3 Dataset…………………………………………………………53-54
5.29 MC results for PC4 Dataset…………………………………………………………54
5.30 MC results for PC5 Dataset…………………………………………………………54-55

LIST OF FIGURES

4.1 Safe-Level-SMOTE Algorithm for Imbalanced Data.. 18

4.2 ADASYN Algorithm for Imbalanced Data.. 19

4.3 SPIDER Algorithm for Imbalanced Data... 20

4.4 SPIDER2 Algorithm for Imbalanced Data.. 21

4.5 SPIDER3 Algorithm for Imbalanced Data.. 23

4.6 MUTE Algorithm for Imbalanced Data.. 24

4.7 SPY Algorithm for Imbalanced Data.. 24

Abstract

Data imbalancing is becoming a common problem to tackle in different fields like, defect prediction, change prediction, oil spills, medical diagnose etc. Various methods have been developed to handle imbalanced datasets in order to improve accuracy of the prediction models. Software defect prediction is important to identify defects in the early phases of software development life cycle. This early identification and thereby removal of software defects is crucial to yield a cost-effective and good quality software product. Though, previous studies have successfully used machine learning techniques for software defect prediction, these techniques yield biased results when applied on imbalanced data sets. An imbalanced data set has non-uniform class distribution with very few instances of a specific class as compared to that of the other class. Use of imbalanced data sets leads to off-target predictions of the minority class, which is generally considered to be more important than the majority class. Thus, handling imbalanced data effectively is crucial for successful development of a competent defect prediction model. Many studies have been carried out in the field of defect prediction for imbalanced datasets but most of them uses SMOTE oversampling method to handle the imbalanced data problem. There are many other oversampling methods which help to deal with imbalancing problem and are still unexplored particularly in the field of software defect prediction. This study evaluates the effectiveness of machine learning classifiers for software defect prediction on twelve imbalanced NASA datasets by application of nine sampling methods. We also propose a modified version (SPIDER3) of the existing oversampling method SPIDER2 and compare it with the original one. Furthermore, the work evaluates the performance of MetaCost learners on imbalanced datasets. The results show improvement in the prediction capability of machine learning classifiers with the use of sampling methods. MetaCost learners improves the sensitivity and helps to predict defects effectively. Moreover, they advocate the applicability of modified version of SPIDER2 oversampling method as it outperforms the original SPIDER2 method in majority of the cases.

CHAPTER 1

INTRODUCTION

1.1 Introduction

Defects in a software is a very common and frequently occurring problem. Software designing, coding, addition of new features, modification of a software etc. can lead to number of faults in the software. A software defect is defined as a bug that causes software failure and prevents it from producing the desirable outcomes. To minimize the chances of software failure, it becomes necessary to find faults in the software. As finding each and every fault is a sophisticated and impractical task, researchers focus on developing fault prediction models.

In recent years, many studies [16], [25], [26], [27], [28], [29], [30], [31], [32], [56] have successfully developed software defect prediction models. Software defect prediction involves determination of the probability of occurrence of a defect in the future or unseen versions of a software product. Since, a software defect may cause software failure and forbids the software to produce desirable outcomes, early detection of software defects is beneficial so that they can be corrected in the initial phases of software development life cycle. This helps in the development of a cost effective model because detection and correction of defects becomes difficult and costlier if they propagate to later phases.

Thus, software defect prediction aids in development of good quality software product with lower testing and maintenance costs and thereby satisfied customers. Software defect prediction models rely on past data and classify modules as defective or non-defective on the basis of this historically collected data. Previous studies have used various software metrics (section 1.1.1.) sets along with defect data, to build prediction models which have been proved reformative for predicting defect prone modules.

Previous research on defect prediction demonstrates that 80% of the defects occur in very few modules (20%) while the rest 80% of modules contains only 20% of the total defects [55]. This indicates that defective classes are present in minority (less number) as compared to non-defective classes, which results in imbalanced datasets. Imbalanced data is

the data in which distribution of classes is one-sided, which may result in incorrect prediction of the minority class instances.

Although, the minority class instances are low in number, but in majority of the cases they are important to be classified correctly. Incorrect prediction of defective classes might result in escape of critical errors leading to bad quality software and higher testing costs. Thus, misclassification of defective classes may lead to project scrap which can further harm the reputation of an organization. Therefore, it is important to address imbalanced data problem for software defect prediction to improve software quality, to reduce prediction error and for successful deployment of the software.

Below subsections demonstrates the use of software metrics and elaborates the occurrence of data imbalancing problem in defect prediction studies.

1.1.1. Software Metrics

A software metric is a measure of an extent to which a software system possess some characteristics. Software measurement is done through code coverage, cohesion, coupling, lines of code, cyclomatic complexity, Halstead complexity, function points etc. Metrics are defined at various levels for example, method level metrics, class level, file level, component level, quantitative metrics, product metrics and process metrics. Out of method level metrics, Halstead (1977) and McCabe (1976) are widely used metrics [26]. Now-a-days, class level metrics are also becoming popular but their use is confined to object oriented software only whereas method level metrics can be used for both structured as well as object oriented programming paradigm[26]. Popular class level metrics are CK metrics suite, MOOD, QMOOD and L&K [26]. Kaszycki (1999) observed that performance increases if we use process metrics as well [29]. The only difficulty with these metrics is that they change with the change of organization. So, it is required that model must be built from root again [29].

This thesis work is based on method level metrics i.e. Halstead and McCabe static code metrics suite. The study uses a set of 36 procedural metrics as independent variables. Procedural metrics consists of a set of traditional code metrics defined by Halstead [23] and McCabe [24] and lines of code (LOC) counts which are categorized under size metrics. Halstead metrics are used to measure complexities on the basis of number of operators and operands in a module [25] while McCabe metrics set is deduced using the flow graph information of a module.

Many previous defect prediction studies have used procedural metrics in order to conduct their experiments on defect datasets. For example, a study of Catal and Diri [28] used procedural metrics suite to assess results with respect to various machine learning techniques. Chug and Dhall [27] incorporated the use of static code metrics in their research regarding clusters and machine learning techniques. Lessmann et al. used Halstead and McCabe metrics to build defect prediction classification models. These are the metrics which have been used in a lot of defect prediction related studies.

1.1.2. Imbalanced Data Problem

Imbalanced data problem is a common problem in many machine learning (ML) and data mining related domains for example, network intrusion detection [52], medical diagnosis [51], fraud detection [53], hyperspectral image classification [54], software defect prediction [11] etc. A data set is called imbalanced when one of the classes i.e. the minority class is heavily under-represented in contrast to the majority class which have larger number of instances as compared to minority class [18]. This means imbalanced data results from biased distributions of classes.

Imbalanced data is considered as a serious problem in ML domain. It can cause adverse effect on the actual performance of various ML classifiers. In most of the cases, the accurate classification of minority class is more important than that of majority class as it is costly to misclassify instances from the minority class [18], [19]. For example, in case of medical diagnosis, cancer disease is less common but it is important to diagnose a person with cancer correctly otherwise it may lead to a loss of life. The traditional standard classifiers are built with the assumption that the input dataset is balanced with respect to various classes but when one class dominates the other, the classifiers tend to misclassify the minority class which results in the increase in prediction error [35]. This limitation of classifiers can lead to huge losses in terms of life and money.

There are four different characteristics that imbalanced data holds as explained by Ramyachitra and Manikandan [18]: small disjuncts, lack of density, noisy data and dataset shift. In this study, we are dealing with the fourth characteristic that is dataset shift. This is defined as the case where the dataset follow different distributions with respect to various classes and the minority class is mostly sensitive to prediction errors. This work focuses on binary class (defective and non-defective) imbalance problem. The defect prone classes are present in only 20% of the total modules but are very important to be predicted correctly.

There are many techniques to handle imbalanced datasets on various levels like data level, algorithm level, cost sensitive level, feature selection level and ensemble level [6]. These levels further encompass different methods and algorithms to handle the imbalanced data. This work explores data level and cost sensitive approach. The data level methods include oversampling methods, undersampling methods and resampling methods. These are further categorized into various data balancing techniques which we will discuss in the later chapters.

1.2 Motivation of the Work

Handling imbalanced datasets to obtain improved results is an important challenge in software defect prediction area. Various methods have been developed to deal with imbalanced data like data sampling methods, cost sensitive learning, ensemble methods etc. [6], [12], [14], [48]. As mentioned above, this study specifically focuses on data sampling methods and cost sensitive learning.

- *Data sampling methods* as mentioned by Lopez et al. [6] sample the data either by eradicating some of the majority class samples or by duplicating or adding new synthetic minority class samples. Ruling out some of the majority class samples is called the under sampling while addition of minority class samples (replicas or synthetic instances) is known as over-sampling technique.
- *Cost sensitive learning* balances the data distribution by considering the cost of misclassification. All misclassification errors may not be equal in terms of cost. A predictor tries to minimize the cost by making less number of costlier misclassification errors. In this work, we assess the performance of oversampling methods as well as MetaCost (MC) learners [22] for handling the imbalanced datasets.

Though, a number of studies in literature have explored imbalancing problem in software defect prediction domain [8], [9], [10], [11], [12], [13], [14], [15] yet only few sampling methods have been investigating in this domain. Amongst the explored sampling methods, Synthetic Minority Oversampling Technique (SMOTE) is a popular method while others like ADASYN, SPIDER, MUTE, SPY etc. are still novel to the area of defect prediction. This study investigates the performance of novel balancing techniques with use of NASA datasets. Furthermore, it implements these sampling methods together with the MC

learners for five ML classifiers to handle imbalanced data problem. We further perform statistical tests to compare their performances.

1.3 Goals of the study

This work examines nine sampling methods out of which five are oversampling techniques (SMOTE, ADAptive SYNthetic sampling technique (ADASYN), Safe-Level-SMOTE, Selective Preprocessing of Imbalanced Data (SPIDER) and SPIDER2), one is undersampling technique (Majority Undersampling TEchnique (MUTE)) while the remaining three are resampling techniques (SPY, SpreadSubSample and Resample) together with the MC learners with three different cost ratios by using five ML classifiers. The ML classifiers used in this study are decision trees (J48 and Random Forest (RF)), Naïve Bayes (NB), and two ensemble methods AdaboostM1 (AB) and Bagging (BG). Furthermore, in order to generalize the results we explore twelve defect prediction public NASA datasets. We implement nine existing balancing techniques in MATLAB. We also propose and implement an improved version of SPIDER2 i.e. SPIDER3. The results are appraised using Area Under the Receiver Operating Characteristic Curve (AUC), sensitivity, specificity and precision performance metrics. Furthermore, this study performs statistical comparison of the results using Friedman and Wilcoxon tests.

Thus, this study investigates the following research questions (RQ):

RQ1: Does balancing of datasets using sampling methods improve the performance of ML techniques for defect prediction?

RQ2: Which is the best oversampling method to improve the performance of ML techniques for software defect prediction in this study?

RQ3: What is the comparative performance of the proposed version of SPIDER2 technique i.e. SPIDER3 and the original SPIDER2 technique for software defect prediction?

RQ4: Which is the best sampling method among undersampling and resampling methods to improve the performance of ML techniques for software defect prediction in this study?

RQ5: Which sampling technique is the best among oversampling, undersampling and resampling techniques and why?

RQ6: What is the effect of using MC learners on imbalanced datasets for software defect prediction?

RQ7: What is the comparative performance of best sampling method and MC learners for software defect prediction?

1.4 Thesis Organization

This thesis work is bifurcated into six different chapters. Starting with the abstract, *Chapter 1* gives the brief introduction about the issues discussed in this study. The chapter explains the need and use of defect prediction models. It defines the defect related terminologies explaining how they affect the software systems and human life. It also addresses the imbalanced data problem, how it has been leading to the ignorance of defect prone classes in defect prediction area. The various software metrics used to develop prediction models are demonstrated and the goals of this empirical research are stated in the form of questions at the end of this chapter.

Chapter 2 sums up the related studies with respect to software defect prediction and imbalanced data problem. A lot of research has been carried out in defect prediction area in context of imbalanced data. This chapter summarizes the major contributions and findings of the previous studies. The literature survey conducted by the author in defect prediction finds out that imbalanced data is becoming a serious problem. Many studies [5], [9],[15], [13], [11] have been investigating in this field by using data balancing techniques. Most of the studies use SMOTE sampling method to handle imbalancing problem while other methods are still unexplored in the area of defect prediction. Only one defect prediction study [8] has used Safe-Level SMOTE oversampling method while the methods like ADASYN, SPIDER, SPY etc. are still novel. Furthermore, the related work describes the previous studies which have used procedural metrics and have applied various ML techniques for building models.

Chapter 3 provides the details regarding the experimental design of the study. It describes the dependent, independent variables used to carry out the research. The data collection method, different datasets and the various procedural metrics used in this study are mentioned in detail. The chapter further defines the performance metrics used to evaluate the prediction models and discusses the statistical test selection briefly. The 10-fold cross validation method used for model evaluation is explained in this section.

Chapter 4 describes the research methodology used in the experiment. It briefly discusses the various data sampling methods together with the detailed explanation of the algorithms to handle imbalanced data problem. A proposed oversampling method SPIDER3 is also discussed with full details. A detailed discussion is carried out regarding MetaCost

learners which is cost sensitive approach of dealing data imbalancing. Furthermore, this section defines various machine learning classifiers which are applied on balanced as well as imbalanced datasets to develop defect prediction models.

In *Chapter 5* the obtained results are stated and analysed using statistical tests. This chapter answers the above stated questions in chapter 1. We have performed an extensive comparison between various balancing methods using two non-parametric tests, Friedman and Wilcoxon. This chapter also states the advantageous use of the proposed method SPIDER3 and describes how it is better than the existing one (SPIDER2).

At last, *Chapter 6* concludes the final outcome of the study. It states which method performed the best and guides the researchers to make use of novel sampling techniques to further improve the performance of defect prediction models. The chapter also provides the future scope of the research.

CHAPTER 2

RELATED WORK

This section discusses the related work of this study. The section is further sub-divided into two parts. The first part discusses the existing studies in defect prediction domain which have used method level metrics i.e. static code metrics suite and NASA datasets. The second part mentions previous studies related to imbalanced data problem in defect prediction domain as well as in other areas.

2.1 Defect Prediction Studies

There are a number of previous studies which have used NASA data sets for defect prediction. Chug and Dhall analyzed various ML techniques and clusters for defect prediction on NASA data sets using static code metrics [27]. They found that RF outperforms all the other investigated ML techniques for software defect prediction.

Catal and Diri inquired the effect of dataset size and metrics set on software defect prediction [28]. They also used public NASA datasets and observed that RFs technique outperforms for large datasets and NB for small datasets. Another study by Catal and Diri observed that the most frequently used metrics in defect prediction are method-level metrics [26]. Also, ML techniques were found to be popular methods for defect prediction. A study by Catal [29] and another one by Malhotra [17], surveyed both ML and statistical techniques for defect prediction. According to their surveys, most of the studies used method level metrics and the defect prediction models were mostly developed using ML techniques. Moreover, the ML techniques outperformed the statistical methods in majority of the cases for developing software defect prediction models.

Gondra proposed an ML technique for selecting a subset of software metrics that are most likely to predict defects and used NASA datasets to obtain results [30]. The study concluded that the Support Vector Machine (SVM) performs better than that of Artificial neural networks (ANNs). Li and Reformat studied a fresh ML method ‘SimBoost’ to make the dataset more balanced in order to handle the skewness in data distributions in software defect prediction [31]. Although, the method attempted to balance the datasets but the accuracy of the prediction was still not acceptable.

The study by Hong carried out his research on RF classifier [32]. He proved that the RF model was better than the MultiLayer Perceptron neural network model and Support Vector Machine (SVM) model. Shanthini and Chandrasekaran analyzed the performance of ML models using traditional performance measures such as precision, recall and AUC [33]. Their results which were based on public domain NASA data set KC1 showed that the RF outperforms the other methods. Singh et al. also used public domain NASA data set KC1 to analyze that the SVM method predicts defective classes with high accuracy when evaluated using AUC [34]. Lessmann et al. used Halstead and McCabe metrics to build defect prediction classification models. These are the metrics which have been used in a lot of defect prediction related studies [27], [28], [30], [33], [34].

2.2 Imbalanced Data Related Studies

Some previous studies on defect prediction have inculcated the data pre-processing step by applying balancing techniques to get better results. The most popular and widely used method is SMOTE (Synthetic Minority oversampling technique) and its modified versions [5], [9], [12], [15]. A number of previous studies have used SMOTE for balancing the unbalanced data but there are more improved methods like SPIDER(selective preprocessing of imbalanced data), ADASYN (Adaptive synthetic sampling) which produce better results when compared to SMOTE. To the best of author’s knowledge, no work has been found in regard to these methods in software defect prediction.

Siers and Islam incorporated the oversampling methods, SMOTE and Safe-Level-SMOTE to optimize the cost of software defect prediction using decision forest [8]. The use of oversampling methods gave better results where number of defective examples was less than 100. To address the imbalanced data problem for software defect prediction, Liu et al. proposed a two-stage cost-sensitive learning (TSCS) method [10]. Their experimental results demonstrated that the TSCS methods outperformed single-stage cost-sensitive learning methods. Tan et al. applied the oversampling methods to improve the performance in online change classification [20]. Their results depicted that the oversampling methods improved the performance by significant percentage points. The study of Rodriguez et al. [7] compared cost-sensitive, sampling methods, hybrid techniques and ensembles to deal with imbalanced datasets. Their results showed that the algorithms to deal with imbalanced datasets enhanced the performance of prediction models.

The study of Wang and Yao compared the balancing techniques and concluded that the balanced random undersampling had a better defect prediction rate than the other methods [11]. To better estimate the cost, Khoshgoftaar and Gao used random undersampling (RUS) [13]. The results showed that the sampled data significantly out-performed the models that were constructed with the original, unsampled data. Kamei et. al. experimentally evaluated the effects of sampling methods (random over sampling, SMOTE, random under sampling and one-sided selection) on defect-prone ML models [14]. They discovered that sampling methods improved the prediction performance of the linear and logistic models, while the performance of neural network and classification tree models did not improve by the use of sampling methods.

Seliya and Khoshgoftaar used cost sensitive method to analyze the performance ML techniques in case of imbalanced datasets [48]. They considered misclassification cost as an important factor for making better models. Weiss et al. [49] and Galar et al. [50] also worked on imbalanced data in defect prediction. They used cost sensitive learning, sampling methods and ensemble methods to improve the performance of ML models.

Although these studies have worked on improving the performance of defect prediction models using imbalanced data but no study has explored all the above mentioned sampling methods to handle imbalanced data in the software defect prediction domain. Though, SMOTE has been popularly used in previous studies, this study analyzes the use of new and improved sampling methods like ADASYN and SPIDER etc. Furthermore, this study implements these nine sampling methods in the MATLAB environment and then uses them for developing better defect prediction models using ML techniques.

A previous work by Malhotra and Khanna in [21] analysed the performance of three sampling methods (sampling, SMOTE and Spread Subsample) along with MC learners on change prediction data. However, this work is different from author’s previous work as it investigates specifically the use of various oversampling methods on defect prediction data. Moreover, the various oversampling methods (apart from SMOTE) used in this work has been coded by the authors themselves. Also, the study proposes a new variant of an existing oversampling method SPIDER2. Our improved version of the SPIDER2 algorithm is more effective for handling imbalanced data than its original version. Moreover, this study uses twelve public NASA defect datasets as compared to only six data sets used in author's previous work on change prediction.

CHAPTER 3

EXPERIMENTAL DESIGN

This section provides the details regarding various design settings used in this study.

3.1 Dependent and Independent Variables

This study uses 'defect proneness' as a dependent variable [56]. Defect proneness is a binary variable which indicates the defective nature of the class. A class is said to be defect prone if there is a probability of detecting a fault in the class in future versions otherwise, a class is termed as non-defective. This binary variable is dependent on a number of other variables like Halstead and McCabe metrics.

The dependence of defect proneness over static code metrics is considered practical as they have helped in successful detection of the defect prone nature of the class in the past [25], [30], [33]. They are also helpful in deciding whether the module should go through manual inspections or not. According to the survey by Malhotra in [17] procedural metrics are widely used metrics in more than 51% of previous [25], [26], [30], [33], [34] defect prediction studies and can be calculated at reasonably low costs for both small and large systems. Table 3.1 describes the static code metrics, size metrics and other metrics which are a part of procedural metrics used in this study.

3.2 Data Collection

This study uses a set of 12 publically available NASA datasets. As observed by Malhotra in [17] more than 60% of the previous software defect prediction studies [26], [30], [31], [32], [34] used NASA datasets. They are available publically in NASA repository by NASA Metrics Data Programme. The NASA datasets used in this work are collected from the PROMISE repository. The NASA datasets used in this study are explored with the application of data sampling and ML techniques.

Table 3.1 Static Code Metrics Description in NASA Datasets

	Metric	NASA Dataset											
		CM1	JM1	KC2	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Halstead Metrics	*Level*	√	√	√	√	√	√	√	√	√	√	√	√
	Program time	√	√	√	√	√	√	√	√	√	√	√	√
	Volume	√	√	√	√	√	√	√	√	√	√	√	√
	Error estimate	√	√	√	√	√	√	√	√	√	√	√	√
	Length	√	√	√	√	√	√	√	√	√	√	√	√
	Content	√	√	√	√	√	√	√	√	√	√	√	√
	Difficulty	√	√	√	√	√	√	√	√	√	√	√	√
	Effort	√	√	√	√	√	√	√	√	√	√	√	√
	Num_operands	√	√	√	√	√	√	√	√	√	√	√	√
	Num_unique_operands	√	√	√	√	√	√	√	√	√	√	√	√
	Num_operators	√	√	√	√	√	√	√	√	√	√	√	√
	Num_unique_operators	√	√	√	√	√	√	√	√	√	√	√	√
McCabe Metrics	*Essential Complexity*	√	√	√	√	√	√	√	√	√	√	√	√
	Cyclomatic Complexity	√	√	√	√	√	√	√	√	√	√	√	√
	Design Complexity	√	√	√	√	√	√	√	√	√	√	√	√
	Cyclomatic Density	√	√	√	√	√	√	√	√	√	√	√	√
Size Metrics	*Number of lines*	√	-	-	√	√	√	√	√	√	√	√	√
	LOC total	√	√	√	√	√	√	√	√	√	√	√	√
	LOC executables	√	√	√	√	√	√	√	√	√	√	√	√
	LOC comments	√	√	√	√	√	√	√	√	√	√	√	√
	LOC code & comments	√	√	√	√	√	√	√	√	√	√	√	√
	LOC blanks	√	√	√	√	√	√	√	√	√	√	√	√
Others	*Branch count*	√	√	√	√	√	√	√	√	√	√	√	√
	Condition count	√	-	-	√	√	√	√	√	√	√	√	√
	Decision count	√	-	-	√	√	√	√	√	√	√	√	√
	Edge count	√	-	-	√	√	√	√	√	√	√	√	√
	Parameter count	√	-	-	√	√	√	√	√	√	√	√	√
	Modified condition count	√	-	-	√	√	√	√	√	√	√	√	√
	Multiple condition count	√	-	-	√	√	√	√	√	√	√	√	√
	Node count	√	-	-	√	√	√	√	√	√	√	√	√
	Decision density	√	-	-	√	-	√	√	√	√	√	√	-
	Design density	√	-	-	√	√	√	√	√	√	√	√	√
	Essential density	√	-	-	√	√	√	√	√	√	√	√	√
	Global data density	-	-	-	√	√	√	-	-	-	-	-	√
	Call pairs	√	-	-	√	√	√	√	√	√	√	√	√
	Maintenance severity	√	-	-	√	√	√	√	√	√	√	√	√
	Global data complexity	-	-	-	√	√	√	-	-	-	-	-	√
	Normalized cyclomatic complexity	√	-	-	√	√	√	√	√	√	√	√	√
	Percent comments	√	-	-	√	√	√	√	√	√	√	√	√
Number of code attributes		37	21	21	39	38	39	37	37	37	37	37	38
Total number of modules		344	7782	522	194	1988	125	253	759	1585	1077	1458	17186
Percentage of defected modules		12.21	21.48	20.5	18.55	2.31	35.2	10.67	8.03	1	12.44	12.20	3

Each dataset comprises of procedural metrics including static code metrics and size metrics. They also contains the defect proneness dependent variable. The value of dependent variable is set to ‘0’ or ‘no’ if the module is not defective otherwise it is set to ‘1’ or ‘yes’ in case of defective module. The datasets selected in this study are highly imbalanced with minority class (i.e. number of defective modules) percentage in the range of 1-35.5%.

The detailed description of 12 NASA datasets used in this study together with the procedural metrics used in each dataset is given in Table 3.1 which includes the total number of modules per dataset and percentage of defect prone modules.

3.3 Selection of Performance Measures

Performance of different defect prediction models can be evaluated using various performance metrics for example precision, recall, specificity, balance, AUC, F-measure, G-mean, accuracy etc. Researchers had been involved in a controversy over the use of performance measures while dealing with imbalanced data. The use of recall, precision and accuracy performance measures have been criticized by researchers [36], [37], [38] for the evaluation of prediction models while AUC, balance etc. are considered effective measures for the defect prediction models evaluation in case of imbalanced data [12], [25], [37]. This work evaluates the performance of ML classifiers using four performance metrics. We use AUC, specificity and two traditional performance metrics: recall and precision as well for evaluating the results of the prediction models. AUC is an important metric to be considered for evaluation as it shows the trade-off between correct and incorrect predictions made by a classifier [46].

The performance of the developed models is evaluated using confusion matrix. It consists of four variables out of which two are predicted class labels and other two are actual class labels. Two classes used in this paper are defective (whether the module is defective) and non-defective class. In matrix, TN (true negatives) is the number of non-defective samples of the dataset which are predicted as non-defective, TP (true positives) is the number of defective samples of the dataset predicted correctly as defective, FN (false negatives) implies to the number of defective samples predicted as non-defective and similarly FP (false positives) refers to the number of non-defective samples predicted falsely as defective. Table 3.2 shows the confusion matrix formation while various performance metrics used in this study are described with the help of definition along with formula in table 3.3.

Table 3.2 Confusion Matrix

Class	Predicted Negatives	Predicted Positives
Actual Negatives	TN	FP
Actual Positives	FN	TP

Table 3.3 Performance Metrics

Performance Metric	Definition
Area under ROC curve	Area Under the ROC Curve (AUC) is a combined measure of sensitivity and specificity. The ROC is a curve plotted between sensitivity and (1-specificity) on the y and x-coordinate axis respectively. The larger the area enclosed under the curve the better is the performance of the ML technique.
Sensitivity (Recall)	It is defined as a percentage of correctly predicted defective modules. $$Sensitivity = \frac{\text{TP} * 100}{\text{TP} + \text{FN}}$$
Specificity	It is defined as a percentage of correctly predicted non-defective modules. $$Specificity = \frac{\text{TN} * 100}{\text{TN} + \text{FP}}$$
Precision	It is defined as the ratio of correctly predicted defective modules to the total number of modules predicted as defective. $$Precision = \frac{\text{TP} * 100}{\text{TP} + \text{FP}}$$

3.4 Statistical Test Selection

In order to statistically evaluate the performance of data sampling methods and MetaCost learners, we use two statistical tests: Friedman test and Wilcoxon signed rank test. These tests are non-parametric tests and are conservative in nature. Unlike parametric tests, assumptions made in the non-parametric tests are not stringent and one may ignore the presence of outliers in the datasets, variance homogeneity, normal distributions etc [39]. Lessmann et al. ascertain that only few previous studies have used statistical tests for performance validation [25]. Deriving conclusions exclusively by manual inspection of empirical results might be misleading and can create inconsistency across more than one experiment performed on the same subject. To avoid this scenario, we use the two selected statistical tests to generate substantiated conclusions.

Friedman test assigns ranks to different methods under experiment on the basis of performance metrics used for evaluation. The lower the mean rank attained by any method, the better it is. The degree of freedom for the test is set to 6 and the alpha value to α=0.05. If

the results obtained by Friedman test are significant, we perform Wilcoxon signed rank test with Bonferroni correction. Bonferroni correction is used to remove family wise errors. The test ascertains whether the pairwise performance of two methods differs significantly or not. It compares two related scenarios (a vs b) using positive and negative ranks. Positive ranks indicate the number of times 'a' outperforms 'b' out of total number of instances while negative ranks indicate the number of times 'b' outperforms 'a'. If positive ranks are equal to negative ranks then the performance of both 'a' and 'b' is considered equal. We use $\alpha=0.05$ as a decision parameter for the acceptance or rejection of null hypothesis.

3.5 Model Evaluation

The study uses 10-fold cross validation method for model evaluation. The method works by randomly dividing the dataset into ten subsets. Ten iterations are performed where, during each iteration, one subset is taken as testing set while other nine subsets are considered as training sets. All the ten subsets are used as validation set exactly once. The final result is calculated by the average estimation of results generated during each iteration.

CHAPTER 4

RESEARCH METHODOLOGY

In this study, we have implemented nine existing sampling methods and have proposed a new and improved oversampling method. The techniques are applied on imbalanced NASA datasets along with the use of MC learners to handle imbalancing problem. Furthermore, we apply five ML classifiers on the balanced datasets in order to evaluate their performance by using four performance measures described in the previous section. The study uses WEKA (www.cs.waikato.ac.nz/ml/weka) for evaluation. The results are computed using default WEKA parameters. This section describes the various methods to handle imbalanced datasets and the ML techniques for defect prediction used in this study.

4.1. Data Sampling Methods

Data sampling methods attempt to balance data either by replicating the minority class samples or by generating new synthetic samples of the minority class or it can also be done by eliminating the noisy majority class instances. This study implemented five oversampling methods, one undersampling and three resampling methods in the MATLAB environment whose brief explanation is stated in this section below.

4.1.1 SMOTE

SMOTE, synthetic minority oversampling technique by Chawla et al. [4] is a widely used method. In SMOTE, for each minority class sample its k nearest neighbors are computed and are randomly chosen in order to compute synthetic samples close to each minority class sample. This study chooses seven different values of k depending on the requirement of each of the 12 NASA datasets used in this work. Selection of number of nearest neighbors depends upon the amount of oversampling (N) needed. The amount of oversampling required can be 100%, 200%, 500%, 1000% and so on. For example, 500% oversampling means five nearest neighbors are randomly chosen from k nearest neighbors. Amount of oversampling further depends on the percentage of minority class present in each dataset with respect to total number of instances present in the dataset. This study uses amounts of oversampling in the range of 200-9000%. Detailed description of k and N is provided in table 4.1 below. The synthetic sample for each minority class sample is generated by taking the difference between the particular minority class sample and its nearest

neighbor. The difference is then multiplied by a random number which belongs to the range from 0 to 1 and then it is finally added to that particular minority class sample under consideration. Detailed Algorithm of SMOTE can be referred in.

Table 4.1 Parameter Selection for Sampling Algorithms

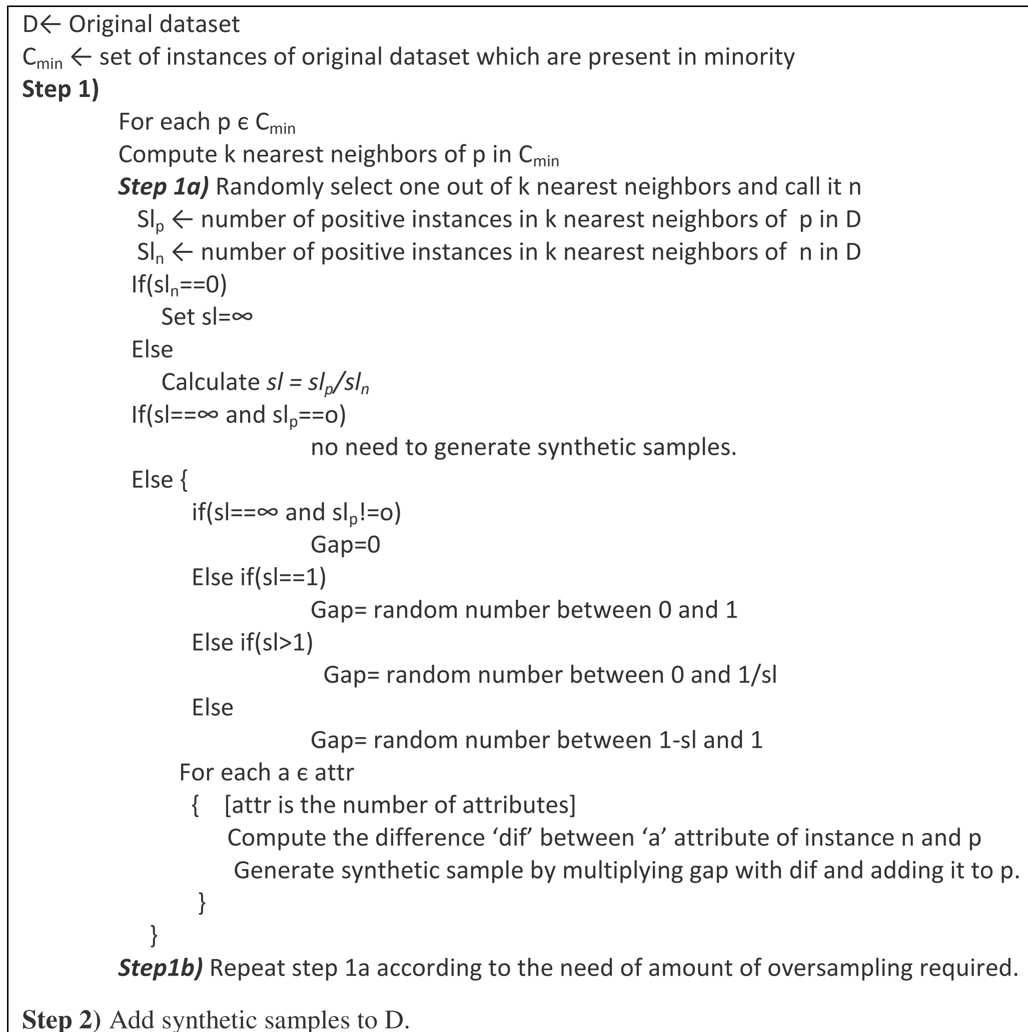
Dataset	SMOTE/Safe-Level-SMOTE		SPIDER/SPIDER2/SPIDER3	MUTE	SPY	
	K	N	K	k'	k"	Z
CM1	5	5	5	5	5	2
JM1	4	4	4	5	5	2
KC2	4	4	4	5	5	2
KC3	4	4	4	5	5	2
MC1	25	25	25	5	25	12
MC2	3	2	3	5	5	2
MW1	5	5	5	5	5	2
PC1	7	7	7	5	5	2
PC2	15	90	90	5	89	44
PC3	5	5	5	5	5	2
PC4	5	5	5	5	5	2
PC5	30	30	30	5	29	14

4.1.2. Safe-Level-SMOTE

Safe-level-SMOTE [2] is the modified version of SMOTE. It focuses on how the random number (used in SMOTE) will be chosen to generate synthetic minority samples. The minority class samples are assigned safe levels (sl) on the basis of k nearest neighbors in the dataset. In our experiment, we set the value of k to seven different values depending on the requirement of each of the 12 NASA datasets used in this work and the amount of synthetic samples to be generated are set in the range of 200-9000%. Detailed description of k and N is provided in table 4.1 above.

We find k nearest minority class neighbors for each minority class sample 'p'. One neighbor 'n' will be chosen randomly and then safe level ratio sl_p/sl_n (number of minority samples in k nearest neighbors of p in the dataset to the number of minority class samples in k nearest neighbors of n) will be calculated. On the basis of range of safe level ratio, random number is chosen accordingly. Then it will be used same as in SMOTE to generate synthetic samples. Difference between SMOTE and Safe-level-SMOTE is that SMOTE randomly generates equal synthetic samples for each minority class sample while it is not the case in

safe-level-smote. The generation of synthetic samples depends upon the gap variable. Figure 4.1 describes the Safe-Level-SMOTE algorithm.



D← Original dataset
C_{min} ← set of instances of original dataset which are present in minority
Step 1)
For each p ϵ C_{min}
Compute k nearest neighbors of p in C_{min}
Step 1a) Randomly select one out of k nearest neighbors and call it n
Sl_p ← number of positive instances in k nearest neighbors of p in D
Sl_n ← number of positive instances in k nearest neighbors of n in D
If(sl_n==0)
Set sl=∞
Else
Calculate $sl = sl_p/sl_n$
If(sl==∞ and sl_p==o)
no need to generate synthetic samples.
Else {
if(sl==∞ and sl_p!=o)
Gap=0
Else if(sl==1)
Gap= random number between 0 and 1
Else if(sl>1)
Gap= random number between 0 and 1/sl
Else
Gap= random number between 1-sl and 1
For each a ϵ attr
{ [attr is the number of attributes]
Compute the difference 'dif' between 'a' attribute of instance n and p
Generate synthetic sample by multiplying gap with dif and adding it to p.
}
}
Step1b) Repeat step 1a according to the need of amount of oversampling required.

Step 2) Add synthetic samples to D.

Fig. 4.1 Safe-Level-SMOTE Algorithm for Imbalanced Data

4.1.3. ADASYN

In Adaptive synthetic (ADASYN) sampling technique [3], the number of synthetic samples needed to be generated for each minority class sample is decided by the density distribution. Unlike SMOTE, ADASYN automatically calculates the number of synthetic samples which are needed to be generated to balance the data. We do not need to manually input the amount of oversampling in case of ADASYN. The only input is the imbalanced dataset which we need to give. Density distribution is the measure of weights which are given to each minority class sample according to their difficulty level of learning. The procedure of generating synthetic sample is same as that of SMOTE. The major difference between SMOTE and ADADSYN is that the former produces the number of synthetic samples as per the user demand and it generates equal amount of samples for each minority class sample

while the latter automatically decides that how many number of synthetic samples are needed to be generated on the basis of density distributions. The variable 'a' used in the algorithm decides the amount of balancing required with respect to number of majority class samples. This study uses a=1 which means fully balanced dataset will be generated post ADASYN application. Figure 4.2 represents pseudo code for ADASYN.

D← Original dataset
C_{min} ← set of instances of original dataset which are present in minority
m_s ← number of minority class samples
m_l ← number of majority class samples
d_{max} ← maximum tolerated degree of class imbalance
Step 1) Calculate class imbalance degree 'Deg'
Deg= m_s/m_l
Step 2) if Deg< d_{max}
Step 2a) Compute the number of synthetic samples that are needed to be generated for each minority class sample as
$T=(m_l-m_s)*a$
Where a ϵ [0,1] is a constant used to specify desired balance level.

Step 2b) For each s ϵ C_{min} do
Compute k nearest neighbors of s in D
Calculate ratio $r(i)=maj(i)/k$
Where 'maj' is the number of majority class samples in k nearest neighbors of s, i=1,2,3…..m_s.

Step 2c) Calculate the density distribution for each minority class sample 'i' as
$R(i)=r(i)/\sum_{i=1}^{ms} r(i)$

Step 2d) Calculate the number of synthetic samples need to be generated for each minority class sample 'i' as
$S(i)= R(i)*T$

Step 2e) Calculate each synthetic sample for 'i' as
Loop from 1 to S(i)
Randomly choose one minority sample 'n' from k nearest neighbors of s in C_{min}.
Generate synthetic sample as
$Synthetic=s+(n-s)*gap$
Where gap is the random number, gap ϵ [0,1]

End loop.

Fig. 4.2 ADASYN Algorithm for Imbalanced Data

4.1.4. SPIDER

Selective preprocessing of imbalanced data (SPIDER) proposed by Stefanowski and Wilk [1] consists of two phases. In the first phase, each sample from the given dataset is flagged as noisy or safe depending on the k- nearest neighbors. In our experiment, we fix the value of k depending upon the number of instances and the amount of oversampling required in case of each of the 12 NASA datasets. The detailed description of k values is described in

table 4.1. In the second phase, amplification of minority samples is done in three ways, that is, weak amplification, weak amplification & relabeling and strong amplification.

```
D← original dataset
C_min ← set of all samples in D which are present in minority
C_maj ← set of all samples in D which are present in majority
k← number of nearest neighbors

Step 1)   for each sample s ϵ D do
                If correct(data, s, k) then
                        type=safe
                Else
                        type=noisy

Step 2)   if amplication==weak then
              For each s ϵ flagged(data, C_min, noisy) do
                      replicate(data, s, k, maj, safe)
          else if amplication==weak & relabeling==true
              For each s ϵ flagged(data, C_min, noisy) do
                      replicate(data, s, k, maj, safe)
              For each s ϵ flagged(data, C_min, noisy) do
                      For each t ϵ C_maj in k nearest neighbors of s &
                      type==noisy do
                         Change class of t from C_maj to C_min
           else
              For each s ϵ flagged(data, C_min, safe) do
                      replicate(data, s, k, maj, safe)
              For each s ϵ flagged(data, C_min, noisy) do
                      If correct(data, s, k+2) then
                            replicate(data, s, k, maj, safe)
                      else
                            replicate(data, s, k+2, maj, safe)

Step 3) Remove all t ϵ D
```

Fig. 4.3 SPIDER Algorithm for Imbalanced Data

In weak amplification, the minority class samples which are flagged as noisy are amplified. For amplification, replicate them by as many numbers as there are safe majority class samples in k nearest neighborhood of each noisy minority class sample. In weak amplification & relabeling, one additional step is performed in which noisy majority class samples in the k nearest neighborhoods of noisy minority class sample are relabeled by modifying their class from majority to minority. Strong amplification amplifies all the examples of minority class whether flagged safe or noisy. But amplification of safe and noisy samples is done differently. Safe samples are replicated by as many numbers as there are safe majority samples in k nearest neighborhood. In case of noisy minority class samples, flagging is done yet again but this time by taking k+2 nearest neighbors. If the sample is flagged safe, it is amplified in its k nearest neighborhood otherwise in k+2 nearest neighborhoods. In this study we use strong amplification level in the second phase. Fig. 4.3 shows the detailed algorithm of SPIDER. Algorithm uses three functions which are: correct(data, s, k),

flagged(data, c, f) and replicate(data, s, k, maj, f). The first function classifies the sample 's' as safe or noisy using its k nearest neighbors. For safe it returns 1 else 0. The second function generates a set of those that are the part of class c and are flagged as f (noisy or safe). The third function replicates the copies of minority class sample 's' as many number of times as there are majority class samples in s's k nearest neighbors which are flagged as f.

4.1.5. SPIDER2

SPIDER2 is a modified version of SPIDER (Algorithm is described in figure 4.4) [44]. In this modified version, flagging of majority and minority class samples is done in different phases. In the first phase, only majority class samples are categorized as safe or noisy. Relabeling is also done in the first phase only. SPIDER2 either re-label all the noisy majority class samples or it removes them completely from the dataset depending upon the re-label option.

D← original dataset
C_{min} ← set of all samples in D which are present in minority
C_{maj} ← set of all samples in D which are present in majority
K← number of nearest neighbors

Step 1) for each sample s ϵ C_{maj} do
  If *correct(data, s, k)* then
    type=safe
  Else
    type=noisy
Step 2) if relabeling==true
  For each t ϵ *flagged(data, C_{maj}, noisy)* do
    Change class of t from C_{maj} to C_{min}
 else
  D←D – *flagged(data, C_{maj}, noisy)*

Step 3) for each sample s ϵ C_{min} do
  If *correct(data, s, k)* then
    type=safe
  Else
    type=noisy

Step 4) if amplication==weak then
  For each s ϵ *flagged(data, C_{min}, noisy)* do
    replicate(data, s, k, maj, safe)
 else
  For each s ϵ *flagged(data, C_{min}, noisy)* do
    If *correct(data, s, k+2)* then
      replicate(data, s, k, maj, safe)
    else
      replicate(data, s, k+2, maj, safe)

Fig. 4.4 SPIDER2 Algorithm for Imbalanced Data

In the second phase, samples of minority class are flagged as safe or noisy considering the changes that arise because of relabeling in the first phase. This is the major

difference between SPIDER and SPIDER2. The former blindly amplifies the minority class samples without taking into account the changes that arise in the dataset due to relabeling while the latter takes into considerations the changes made by relabeling option in the first phase and on the basis of those changes it flags the minority class samples as safe or noisy. After identification of noisy examples, SPIDER2 performs amplification operation on the relabeled dataset.

4.1.6. SPIDER3: A Modified Version of SPIDER2 (Proposed Method)

To add one more method in the family of SPIDER methods, we propose SPIDER3, a modified version of SPIDER2 technique. Pseudo code of SPIDER3 is presented in fig. 4.5 In this method we use three functions out of which two are same as used in SPIDER and SPIDER2 i.e. correct(data, s, k) and flagged(data, c, f). However, we modify the third function replicate(data, s, k, maj, f) by adding one more parameter into it. The modified replicate function is replicate(min, data, s, k, maj, f). This function generates new synthetic samples of minority class sample 's' as many number of times as there are majority class samples in s's k nearest neighbors which are flagged as f.

SPIDER3 consists of two phases. In the first phase, we identify majority class examples as safe or noisy on the basis of k nearest neighbors. Our method uses Euclidean distance instead of heterogeneous value difference metric (HVDM) distance function to compute k nearest neighbors. We used Euclidean distance because this study focuses on defect prediction imbalanced datasets which have all the numeric attributes. Only dependent variable is nominal. HVDM [45] distance function is useful when we deal with heterogeneous data which has both numeric and nominal attributes. It would not make much difference in case of homogeneous data. After identification, relabeling is performed on noisy majority class samples which lie in the nearest neighborhood of each corresponding minority class sample.

In the second phase, identification of minority class samples is done with reference to changes made in dataset by relabeling. Our modifications exist in the amplification phase. Instead of replicating the same minority sample SPIDER3 calculates the synthetic samples while amplifying the data. The synthetic samples are generated by using the same method used in the SMOTE method. Minority class is amplified as many numbers of times as there are safe examples of majority class in its k nearest neighbors. For each minority class sample

amplification, k nearest neighbors are computed in the minority class region only and then a synthetic sample is generated.

```
D← original dataset
Cmin ← set of all samples in D which are present in minority
Cmaj ← set of all samples in D which are present in majority
K← number of nearest neighbors

Step 1)   for each sample s є D do
              If correct(data, s, k) then
                  type=safe
              Else
                  type=noisy
Step 2)   if relabeling==true
            For each s є flagged(data, Cmin, noisy) do
              For each t є Cmaj in k nearest neighbors of s &
               type==noisy do
                  Change class of t from Cmaj to Cmin
        else
            For each s є flagged(data, Cmin, noisy) do
               For each t є Cmaj in k nearest neighbors of s &
                type==noisy do
                  D ← D – t
Step 3)   for each sample s є Cmin do
              If correct(data, s, k) then
                  type=safe
              Else
                  type=noisy
Step 4)  if amplication==weak then
            For each s є flagged(data, Cmin, noisy) do
                  replicate(Cmin, data, s, k, maj, safe)
        else
            For each s є flagged(data, Cmin, safe) do
                  replicate(Cmin, data, s, k, maj, safe)
             For each s є flagged(data, Cmin, noisy) do
                  If correct(data, s, k+2) then
                        replicate(Cmin, data, s, k, maj, safe)
                  else
                        replicate(Cmin, data, s, k+2, maj, safe)
```

Fig. 4.5 SPIDER3 Algorithm for Imbalanced Data

There is an exception which can arise while using SMOTE formula in SPIDER. As we are using the same value of k both for generating synthetic samples as well as for the other computations required by SPIDER3, it might be possible that the value of k exceeds the total number of minority class samples present in the dataset. For example, in case of PC2 dataset there are only 16 minority class examples. But according to the amount of oversampling needed which is 9000% (percentage of minority class is just 1% and hence, large number of samples are required to be generated to fully balance the dataset) we need to set k=90 (In SPIDER family there is no N, oversampling is done on the basis of k only) which exceeds total number of minority class samples. In such case, we simply set k for synthetic sample generation as the total number of minority class examples. Remaining

amount of oversampling will be done by replicating the minority sample. The advantage of our proposed method is that it generates new samples of minority class that is, a synthetic sample rather than just replicating the same sample again and again.

4.1.7. MUTE

Majority Undersampling Technique proposed in [57] is a replica of Safe-Level-SMOTE but with a difference that it generates safe levels for majority class instances while the latter generates them for minority class samples. The method declares a majority sample as safe if its safe level is zero else if safe level is equal to the total number of nearest neighbours then it is declared as noisy [57]. Hence, it removes the noisy majority instances in order to balance the data. The number of k' nearest neighbours taken for experimentation are described in table 4.1. Figure 4.6 describes the MUTE algorithm in detail.

D← original dataset
C_{maj} ← set of all samples in D which are present in majority
T ← minimum number of minority samples in the neighborhood of a majority sample which allow the removal of the majority sample
sl ← number of minority class instances in k-nearest neighbor of each majority class sample
Step 1) For each sample s є C_{maj} do
If sl >= T
Remove s from D

Fig. 4.6 MUTE Algorithm for Imbalanced Data

4.1.8. SPY

It's a novel method which tries to balance the data by changing the class labels of noisy majority class instances from majority to minority class [58]. The noisy majority class samples are those which lie at the boderline. The boderline samples are named as SPY samples in this method as they are noisy and required to be removed or renamed. Table 4.1 states the k" nearest neighbours and threshold value z required for renaming majority class samples to minority. The algorithm of SPY is described in figure 4.7.

D← original dataset
C_{min} ← set of all samples in D which are present in minority
T ← minimum number of minority samples in the neighborhood of a majority sample which allow the removal of the majority sample
k ← number nearest neighbors
Step 1) For each sample s є C_{min} do
C=Count the number of majority class instances in k-nearest neighborhood.
If C <= T
Change the class of the calculated majority instances to minority.

Fig. 4.7 SPY Algorithm for Imbalanced Data

4.1.9. SpreadSubSample

It is a resampling technique which generates a random subsample of a dataset [59]. The maximum spread between minority and majority is stipulated using this method. The distribution spread parameter is set as per the requirements of balancing the dataset.

4.1.10. Resample

It produces a random subsample of a dataset using either sampling with replacement or without replacement [60]. The amount of samples that are required to be replicated are needed to be manually given as input. The biasToUniformClass parameter is set as per the requirements of balancing the dataset.

4.2 MetaCost Learners

Making each classifier cost sensitive is a heavy task. In its contrast, a procedure was proposed by Domingos which was used for making classifiers cost sensitive called MC learner [22]. MC learner makes any ML classifier cost sensitive by applying cost minimizing procedure over it. MC learners do not require any information about how the individual classifier works. They can be applied to the datasets containing any number of classes and to any arbitrary cost matrix. This method uses Bayes optimal prediction to reduce the risk of achieving high overall cost which is called the conditional risk. The conditional risk R(r/x) computes the expected cost value of predicting a sample x as a part of the class ‘r’ when it actually belongs to the class ‘s’. The conditional risk is defined as the summation of product of C(r,s) and P(s/x) for each region ‘j’ where C(r,s) is the cost of predicting an example as a part of ‘r’ when it actually is the member of class ‘s’ and P(s/x) is the probability of predicting that sample x is a member of class ‘s’. The conditional risk partitions the sample space into ‘j’ regions such that least cost prediction i.e. ‘s’ falls into the region of its own class ‘s’.

In this way MC learners re-label the classes of each training instance according to their best predicted classes. MC is the parallel method to BG ensemble method. The difference between the two exists in choosing the size of resample. BG constructs the bootstrap resample by selecting ‘n’ samples with replacement from the training set of size ‘n’ while MC learners go well with the smaller resample size also. This nature of MC learners make them more efficient. The classifiers are then learned on each resample followed by the collection of votes from the ensemble. The majority vote decides the label of each example and hence, re-labels it to the optimal predicted class.

4.3 Machine Learning Classifiers

The ML classifiers used in this work to predict whether the module is defective or not, are described below.

4.3.1. J48

J48 is a decision tree classifier. It helps to classify the instances using information gain [27]. The attribute having higher information gain is selected as a root node and among the possible branches of the root, if there is any child for which all the instances are coming under same class label, we terminate that branch by assigning class target value to it otherwise we continue the above procedure.

4.3.2. Random forest

It is the forest of decision trees in which each tree is made up of randomly selected subsets of datasets using replacement [40]. The final result is given by the majority voting in which each decision tree gives out its own vote.

4.3.3. Naïve bayes

Bayesian learning is based on Bayes' theorem in which the classifier assumes that the effect of one attribute on a given class is independent of the other attributes which is called class conditional independence [41], [47].

4.3.4. AdaboostM1

AdaboostM1 is a boosting classifier [42] which trains various individual classifiers in a serial manner by using the whole dataset. In each iteration it focuses more on the difficult instances which are misclassified in the previous iteration in order to achieve the goal to correctly classify them in the next iteration. The difficulty of the instances is measured by weights which are increased for every misclassification and decreased for correctly classified instances.

4.3.5. Bagging

Bagging is a meta classifier which trains different classifiers using bootstrapped replicas of the original training dataset [43]. The instances are randomly selected with replacement from original dataset to form a new dataset which further trains each classifier.

CHAPTER 5

EMPIRICAL RESULTS AND ANALYSIS

In this section, we discuss and analyse the results obtained by applying ML techniques on sampled as well as original imbalanced datasets. In order to examine and equate the performance of different sampling methods as well as MC learners, we use four performance metrics: sensitivity (recall), specificity, AUC and precision. The results are assessed by using two non-parametric statistical tests: Friedman and Wilcoxon signed rank test. The investigation of results is carried out systematically by sequentially answering the research questions mentioned in chapter 1.

5.1 RQ1: Does balancing of datasets using sampling methods improve the performance of ML techniques for defect prediction?

Tables 5.1 to 5.12 provide the values of performance metrics calculated on the original imbalanced datasets (no sampling) as well as balanced datasets after oversampling, undersampling and resampling. The balanced datasets were obtained by correspondingly applying ten sampling methods: SMOTE, Safe-Level-SMOTE, ADASYN, SPIDER, SPIDER2, SPIDER3, SPY, MUTE, SpreadSubSample and Resample. The defect prediction models were developed by the application of five different ML techniques: J48, RF, NB, AB and BG. To carry out this work, ten sampling methods were implemented in the MATLAB environment where original imbalanced datasets were given as input and balanced datasets were generated as output. To obtain fully balanced datasets, we chose different values of k and N (refer table 4.1) depending on the requirement of each dataset. 10-fold cross validation method was used to develop models.

The empirical study conducted shows that in majority of the cases, the balancing of datasets using sampling methods improves the performance of ML techniques for developing defect prediction models when AUC, precision and sensitivity were used as an evaluation factor.

From table 5.1-5.12, it can be discovered that AUC values in case of sampling methods are better than those in the case when no sampling was performed in majority of the cases. The table results show a significant percentage increase of 2-120% in AUC values in -

Table 5.1 Results for CM1 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.594	0.894	0.893	0.920	0.937	0.873	0.881	0.594	0.594	0.61	0.948
	RF	0.763	0.955	0.938	0.962	1	0.984	0.951	0.773	0.773	0.745	0.992
	NB	0.694	0.927	0.895	0.943	0.711	0.708	0.904	0.694	0.694	0.654	0.767
	AB	0.717	0.965	0.952	0.966	0.736	0.781	0.946	0.717	0.717	0.726	0.863
	BG	0.727	0.960	0.945	0.967	0.994	0.976	0.946	0.754	0.754	0.737	0.959
Sensitivity (in %)	J48	26.2	84.5	84.2	87.7	99.6	95.1	83.6	26.6	26.6	61.9	97.6
	RF	0	83.3	79.8	85.6	100	95.1	82.8	0	0	71.4	97
	NB	33.3	84.5	74.9	89.7	31.4	27.5	86.6	33.3	33.3	33.3	40.6
	AB	0	84.1	80.8	86.3	92.7	82.2	81.3	0	0	73.8	85.5
	BG	0	83.7	79.8	86.0	100	95.5	82.1	2.4	2.4	66.7	93.9
Specificity (in %)	J48	93.7	91.1	91.7	92.4	87.4	82.9	90.1	93.7	93.7	61.9	91.1
	RF	99.0	98.3	98.3	97.4	94.4	92.2	97.3	98.3	98.3	69	90.5
	NB	89.1	89.1	88.7	89.7	89.1	89.8	86.7	89.1	89.1	85.7	90.5
	AB	100	100	98.7	99.3	54.3	59.7	97.3	100	100	71.4	68.7
	BG	99.0	100	99.0	98.3	86.8	88.1	97.3	99	99	66.7	85.5
Precision (in %)	J48	36.7	88.8	87.2	91.8	97.2	82.5	88.5	36.7	36.7	61.9	91
	RF	0	97.7	97.0	98.0	93.9	91.1	96.5	0	0	69.8	90.4
	NB	29.8	86.6	81.7	88.8	71.3	69.4	85.6	29.8	29.8	70	79.8
	AB	0	100	97.6	98.8	63.7	63.2	96.5	0	0	72.1	71.6
	BG	0	100	98.2	98.0	86.7	87.1	96.5	25.0	25.0	66.7	85.6

Table 5.2 Results for JM1 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.616	0.903	0.892	0.880	0.872	0.798	0.856	0.681	0.62	0.623	0.867
	RF	0.701	0.938	0.931	0.918	0.983	0.949	0.918	0.751	0.692	0.681	0.974
	NB	0.633	0.745	0.755	0.704	0.558	0.614	0.705	0.692	0.627	0.63	0.635
	AB	0.669	0.927	0.920	0.901	0.612	0.647	0.882	0.706	0.666	0.656	0.675
	BG	0.689	0.937	0.930	0.914	0.944	0.905	0.907	0.733	0.688	0.682	0.924
Sensitivity (in %)	J48	22.5	84.3	82.2	78.4	92.3	89.4	82.6	36.7	25	64.2	87.5
	RF	20.5	83.7	81.8	77.9	95.4	91.4	82.1	37.4	19.1	63.9	94.1
	NB	18.5	24.4	26.6	18.4	12.5	18.4	18.4	24.2	18.7	19.4	20.5
	AB	0	79.1	76.7	72.2	59.0	83.8	74.5	33	0	56.6	57.6
	BG	17.2	83.5	81.2	77.4	91.7	91.7	80.8	34.9	17.8	62.6	87.1
Specificity (in %)	J48	91.2	92.4	91.5	92.3	77.6	68.1	80.7	89.1	91.4	59.2	79.9
	RF	95.5	95.7	96.1	95.7	91.2	78.7	86.7	92.6	95.7	62	88.2
	NB	94.5	92.7	93.1	93.0	93.2	94.5	94.4	95.1	94.5	94.4	94
	AB	100	99.3	99.2	99.4	57.8	32.8	90.7	91.4	100	69.1	68.2
	BG	95.5	95.6	95.7	95.7	81.1	67.9	85.3	92.1	96	65	82.3
Precision (in %)	J48	41.0	93.8	92.1	91.2	79.5	79.9	86.7	54.9	44.4	61.1	81.3
	RF	55.3	96.4	96.2	94.8	91.0	85.9	90.4	64.7	54.8	62.7	88.9
	NB	48.1	82.1	82.2	72.7	63.4	82.6	83.4	64.1	48.3	77.5	77.5
	AB	0	99.4	99.1	99.2	56.8	63.9	92.4	58.1	0	64.7	64.5
	BG	51.3	96.3	95.8	94.8	82.0	80.2	89.3	61.4	55	64.1	83.1

Table 5.3 Results for KC2 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.704	0.927	0.924	0.906	0.838	0.862	0.886	0.818	0.757	0.709	0.877
	RF	0.825	0.965	0.963	0.955	0.930	0.937	0.952	0.89	0.819	0.8	0.953
	NB	0.832	0.841	0.858	0.786	0.707	0.763	0.766	0.893	0.833	0.818	0.82
	AB	0.784	0.947	0.946	0.941	0.745	0.773	0.920	0.873	0.794	0.76	0.833
	BG	0.825	0.961	0.963	0.951	0.885	0.904	0.943	0.89	0.826	0.821	0.917
Sensitivity (in %)	J48	49.5	89.7	87.5	84.4	77.7	79.9	85.6	67.4	56.1	74.8	86.3
	RF	47.7	89.5	88.7	86.8	84.9	87.8	89.2	68.9	45.8	78.5	91.7
	NB	42.1	46.2	46.9	34.7	24.3	38.4	39.0	62.2	43	46.7	36.1
	AB	43.9	86.7	82.9	80.1	61.0	77.4	83.5	69.6	48.3	76.6	89.5
	BG	43.0	89.2	88.3	86.8	77.7	80.8	87.7	71.9	44.9	77.6	90.3
Specificity (in %)	J48	89.6	91.8	92.8	90.6	82.7	83.2	87.9	92.2	87.4	75.7	86.5
	RF	92.5	92.3	92.8	91.1	85.5	83.2	89.6	93.5	92.3	72.9	86.5
	NB	94.2	92.3	92.5	92.5	92.5	94.0	94.0	94.6	94.9	94.4	96.7
	AB	91.1	92.8	94.9	94.2	79.5	70.6	86.5	90.7	90.3	76.6	69.8
	BG	94.2	91.1	92.0	90.8	84.1	83	89.3	94.6	93	80.4	83.7
Precision (in %)	J48	55.2	93.4	93.6	90.1	75.9	81.1	86.6	75.2	53.6	75.5	87.9
	RF	62.2	93.7	93.7	90.8	80.5	82.5	88.7	78.8	60.5	74.3	88.5
	NB	65.2	88.5	88.4	82.4	69.6	85.1	85.5	80	68.7	89.3	92.6
	AB	56.0	93.9	95.2	93.3	67.7	70.4	85.0	72.3	56.5	76.6	77
	BG	65.7	92.8	93.1	90.5	77.5	81	88.2	82.2	62.3	79.8	86.2

Table 5.4 Results for KC3 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.653	0.887	0.874	0.860	0.908	0.879	0.828	0.604	0.653	0.677	0.902
	RF	0.736	0.943	0.940	0.934	0.979	0.963	0.937	0.721	0.752	0.673	0.982
	NB	0.661	0.885	0.852	0.862	0.661	0.708	0.833	0.682	0.661	0.626	0.689
	AB	0.573	0.913	0.902	0.898	0.760	0.767	0.917	0.61	0.573	0.717	0.836
	BG	0.729	0.944	0.918	0.935	0.947	0.925	0.922	0.691	0.707	0.709	0.927
Sensitivity (in %)	J48	33.3	87.2	83.9	84.0	91.2	86.5	76.4	35.9	33.3	69.4	92.4
	RF	13.9	84.4	79.2	78.8	92.6	89	84.5	12.8	11.1	61.1	97.8
	NB	38.9	83.9	65.1	80.8	27.9	31	44.7	41	38.9	41.7	40.2
	AB	36.1	82.2	79.2	79.5	62.5	74.8	77.0	17.9	36.1	66.7	71.7
	BG	13.9	81.1	79.9	79.5	90.4	84.5	83.9	07.7	11.1	63.9	88
Specificity (in %)	J48	89.9	89.9	89.2	88.0	82.3	85.2	83.0	89	89.9	63.9	87.3
	RF	96.8	95.6	94.9	95.6	89.9	83	88.1	95.5	85.6	61.1	88.2
	NB	88.0	88.6	87.3	88.0	88.6	88.9	88.1	87.7	88	86.1	90.2
	AB	93.0	96.2	95.6	95.6	70.3	71.9	87.4	89.7	93	66.7	84.3
	BG	93.7	95.6	97.5	93.7	84.8	79.3	85.9	98.1	94.9	66.7	81.4
Precision (in %)	J48	42.9	90.8	88.0	87.3	81.6	87	84.2	45.2	42.9	65.8	86.7
	RF	50	95.6	93.7	94.6	88.7	85.7	89.5	41.7	36.4	61.1	88.2
	NB	42.4	89.3	82.9	86.9	67.9	76.2	81.8	45.7	42.4	75	78.7
	AB	54.2	96.1	94.4	94.7	64.4	75.3	87.9	30.4	54.2	66.7	80.5
	BG	33.3	95.4	96.7	92.5	83.7	82.4	87.7	50	33.3	65.7	81

Table 5.5 Results for MC1 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.521	0.984	0.980	0.989	0.992	0.992	0.983	0.521	0.521	0.632	0.987
	RF	0.883	0.997	0.998	0.998	1	1	0.998	0.885	0.885	0.847	1
	NB	0.708	0.936	0.939	0.947	0.754	0.754	0.941	0.708	0.708	0.732	0.727
	AB	0.842	0.992	0.992	0.995	0.889	0.889	0.993	0.842	0.842	0.798	0.89
	BG	0.860	0.992	0.993	0.995	0.999	0.999	0.993	0.854	0.854	0.822	0.999
Sensitivity (in %)	J48	8.7	96.8	96.0	98.1	100	100	96.5	08.7	08.7	73.9	100
	RF	19.6	97.1	96.4	98.1	100	100	97.0	174	17.4	80.4	100
	NB	32.6	97.4	95.7	96.2	37.8	37.8	98.2	326	32.6	43.5	37.5
	AB	0	96.3	95.9	97.7	69.5	69.5	96.4	0	0	71.7	84.2
	BG	0	96.3	95.9	97.8	100	100	96.4	0	0	67.1	100
Specificity (in %)	J48	99.5	99.4	99.6	99.4	97.8	97.8	99.9	99.5	99.5	65.2	96.7
	RF	99.5	99.5	99.6	99.6	99.4	99.4	99.7	99.6	99.6	84.8	99.5
	NB	90.2	75.1	80	80.9	89.9	89.9	75.3	90.2	90.2	89.1	89
	AB	100	99.7	99.7	99.7	84.1	84.1	99.7	100	100	71.7	80.2
	BG	100	99.8	99.8	99.8	98.5	98.5	99.7	100	100	69.6	98
Precision (in %)	J48	28.6	99.1	99.3	99.4	96.6	96.6	99.8	28.6	28.6	68	96.6
	RF	47.4	99.2	99.3	99.6	99.1	99.0	99.5	50	50	84.1	99.5
	NB	7.3	70.7	72.5	83.4	70.3	70.3	71.5	7.3	7.3	80	76.2
	AB	0	99.6	99.5	99.7	73.4	73.4	99.6	0	0	71.7	80
	BG	0	99.7	99.6	99.8	97.6	97.1	99.6	0	0	71.4	97.9

Table 5.6 Results for MC2 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.698	0.736	0.761	0.681	0.786	0.714	0.764	0.715	0.519	0.677	0.762
	RF	0.717	0.905	0.898	0.832	0.974	0.894	0.878	0.741	0.744	0.744	0.923
	NB	0.702	0.807	0.807	0.738	0.678	0.72	0.769	0.73	0.722	0.739	0.736
	AB	0.616	0.863	0.859	0.761	0.764	0.726	0.818	0.662	0.652	0.663	0.804
	BG	0.676	0.899	0.865	0.798	0.922	0.848	0.837	0.718	0.678	0.693	0.846
Sensitivity (in %)	J48	52.3	83.3	83.6	67.9	92.5	84.5	79.0	58.8	40.5	63.6	68.3
	RF	38.6	80.3	83.6	66.7	95.0	87.3	80.6	43.1	40.5	59.1	86.7
	NB	38.6	42.4	43.8	25.6	29.2	43.6	36.3	43.1	35.7	43.2	35
	AB	40.9	75.8	75.0	60.3	83.3	86.4	75.0	43.1	40.5	52.3	73.3
	BG	38.6	79.5	79.7	65.4	97.5	87.3	79.0	41.2	38.1	59.1	68.3
Specificity (in %)	J48	81.5	74.1	70.4	74.1	67.9	50.7	73.1	87.8	72.5	61.4	78.5
	RF	87.7	91.4	87.7	87.7	77.8	62.7	77.6	85.1	86.3	70.5	83.1
	NB	90.1	90.1	87.7	90.1	87.7	85.1	85.1	89.2	91.3	90.9	92.3
	AB	85.2	87.7	87.7	87.7	55.6	38.8	71.6	78.4	82.5	63.6	76.9
	BG	86.4	87.7	79.0	84.0	60.5	46.3	68.7	85.1	86.3	65.9	76.9
Precision (in %)	J48	60.5	84.0	81.7	71.6	81.0	73.8	84.5	76.9	43.6	62.2	74.5
	RF	63.0	93.8	91.5	83.9	86.4	79.3	87.0	66.7	60.7	66.7	82.5
	NB	68.0	87.5	84.8	71.4	77.8	82.8	81.8	73.3	68.2	82.6	80.8
	AB	60	90.9	90.6	82.5	73.5	69.9	83.0	57.9	54.8	59	74.6
	BG	60.7	91.3	85.7	79.7	78.5	72.7	82.4	65.6	59.3	63.4	73.2

Table 5.7 Results for MW1 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.449	0.906	0.862	0.932	0.947	0.92	0.908	0.449	0.449	0.513	0.944
	RF	0.716	0.956	0.936	0.974	1	0.992	0.958	0.711	0.711	0.774	0.99
	NB	0.728	0.896	0.863	0.930	0.764	0.76	0.884	0.728	0.728	0.712	0.737
	AB	0.711	0.949	0.934	0.969	0.836	0.868	0.961	0.695	0.695	0.735	0.863
	BG	0.705	0.954	0.927	0.966	0.985	0.972	0.946	0.698	0.698	0.739	0.976
Sensitivity (in %)	J48	14.8	88.3	82.9	90.6	100	96.6	87.8	14.8	14.8	63	94.1
	RF	18.5	87.0	82.9	91.5	100	96.6	86.2	25.9	25.9	74.1	97.5
	NB	55.6	67.9	65.8	70.1	56.7	57.1	69.6	55.6	55.6	59.3	62.2
	AB	29.6	84.0	81.2	89.3	73.0	73.7	84.5	29.6	29.6	55.6	73.1
	BG	11.1	84.6	79.5	88.8	97.8	95.4	84.0	11.1	11.1	66.7	94.1
Specificity (in %)	J48	95.6	96.5	96.0	96.5	88.9	85.6	93.7	95.6	95.6	55.6	88.8
	RF	96.0	97.3	96.9	96.9	93.4	93.2	96.8	96	96	66.7	93.3
	NB	84.5	88.9	89.4	91.6	82.7	82.4	89.2	84.5	84.5	85.2	84.3
	AB	94.7	99.6	99.1	98.7	81.4	82.9	96.4	94.7	94.7	74.1	77.6
	BG	98.7	99.1	99.1	100	92.9	90.1	96.8	98.7	98.7	77.8	92.5
Precision (in %)	J48	28.6	94.7	91.5	96.2	87.7	84.1	91.9	28.6	28.6	58.6	88.2
	RF	35.7	95.9	93.3	96.7	92.2	91.8	95.7	43.8	43.8	69	92.8
	NB	30	81.5	76.2	89.2	72.1	71.9	84.0	30	30	80	77.9
	AB	40	99.3	97.9	98.5	75.6	77.2	95.0	40	40	68.2	74.4
	BG	50	98.6	93.4	100	91.6	88.4	95.6	50	50	75	91.8

Table 5.8 Results for PC1 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.719	0.953	0.935	0.963	0.960	0.956	0.946	0.675	0.719	0.693	0.968
	RF	0.844	0.982	0.976	0.987	0.995	0.993	0.986	0.869	0.86	0.797	0.995
	NB	0.768	0.912	0.902	0.935	0.783	0.785	0.917	0.76	0.868	0.759	0.785
	AB	0.793	0.974	0.962	0.984	0.831	0.858	0.976	0.802	0.793	0.823	0.867
	BG	0.820	0.978	0.967	0.985	0.990	0.985	0.980	0.838	0.816	0.792	0.985
Sensitivity (in %)	J48	24.6	90.4	85.9	93.1	100	98.8	90.3	29.9	0.246	75.4	99.2
	RF	13.1	90.4	85.6	91.9	100	98.8	90.6	20.9	18	83.6	99.2
	NB	36.1	46.9	48.6	55.8	34.8	34.9	47.1	32.8	36.1	37.7	38.3
	AB	0	87.5	83.4	91.4	90.4	91.7	88.4	0	0	88.5	95
	BG	8.2	88.7	84.8	92.2	99.4	98.8	89.9	11.9	11.5	83.6	98.1
Specificity (in %)	J48	96.7	97.3	97.3	97.0	91.3	91.6	95.4	95.4	96.7	65.6	93.2
	RF	98.1	98.0	97.9	98.3	96.8	97.4	98.3	97.9	99.4	68.9	94.9
	NB	92.8	92.0	92.4	93.3	92.8	93.1	91.6	92.9	92.8	88.5	92.2
	AB	99.6	99.9	99.3	99.4	66.5	66.7	99.3	100	99.6	68.9	70.7
	BG	99.4	98.7	98.9	98.9	92.7	92.2	99.0	99.4	99.4	63.9	93.2
Precision (in %)	J48	39.5	95.9	94.2	96.9	89.9	89.8	93.9	38.5	39.5	68.7	93
	RF	38.1	96.9	95.4	98.2	96.1	96.6	97.6	45.2	50	72.9	94.7
	NB	30.6	80.4	76.9	89.4	79.0	79.0	81.5	31	30.6	76.7	81.8
	AB	0	99.8	98.4	99.4	67.6	67.3	99.0	0	0	74	74.8
	BG	55.6	98.0	97.5	98.8	91.3	90.5	98.6	66.7	63.6	69.9	93

Table 5.9 Results for PC2 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.448	0.993	0.975	0.994	0.990	0.990	0.993	0.448	0.448	0.938	0.988
	RF	0.836	0.999	0.994	0.999	1	1	0.999	0.845	0.845	0.949	1
	NB	0.877	0.995	0.985	0.996	0.896	0.896	0.996	0.877	0.877	0.895	0.896
	AB	0.914	0.999	0.995	0.999	0.976	0.976	0.999	0.914	0.914	0.918	0.962
	BG	0.828	0.998	0.990	0.997	1	0.999	0.998	0.831	0.831	0.965	0.999
Sensitivity (in %)	J48	0	99.0	95.3	99.0	100	100	98.9	0	0	93.8	100
	RF	0	98.9	95.3	99.0	100	100	98.9	0	0	93.8	100
	NB	31.3	90.2	95.9	90.9	37.4	37.4	89.1	31.3	31.3	62.5	47.6
	AB	6.3	98.9	95.3	99.0	97.4	97.4	98.9	06.3	06.3	85.7	100
	BG	0	98.9	95.3	99.0	100	100	98.9	0	0	85.7	100
Specificity (in %)	J48	99.7	99.7	99.7	99.8	97.8	97.8	99.9	99.7	99.7	93.8	97.7
	RF	99.9	100	100	99.9	99.7	99.8	99.9	99.9	99.9	81.3	99.4
	NB	96.1	99.7	99.5	99.6	95.5	95.5	99.7	96.1	96.1	87.5	95.5
	AB	99.8	100	99.9	100	90.2	90.2	100	99.8	99.8	81.3	88.3
	BG	100	100	100	100	98.9	98.9	100	100	100	81.3	97.7
Precision (in %)	J48	0	99.7	98.8	99.8	97.6	97.6	99.9	0	0	93.8	97.6
	RF	0	100	100	99.9	99.7	99.8	99.9	0	0	83.3	99.3
	NB	7.6	99.7	97.6	99.6	88.5	88.5	99.6	07.6	07.6	83.3	90.7
	AB	25.0	100	99.7	100	90.1	90.1	100	25	25	82.4	88.8
	BG	0	100	100	100	98.8	98.8	100	0	0	82.4	97.6

Table 5.10 Results for PC3 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.616	0.894	0.894	0.911	0.940	0.92	0.890	0.629	0.616	0.762	0.939
	RF	0.831	0.969	0.967	0.977	0.999	0.987	0.966	0.819	0.832	0.841	0.997
	NB	0.766	0.891	0.891	0.895	0.760	0.751	0.869	0.764	0.766	0.818	0.736
	AB	0.791	0.959	0.951	0.965	0.777	0.781	0.951	0.773	0.791	0.768	0.793
	BG	0.824	0.963	0.961	0.973	0.988	0.976	0.962	0.791	0.81	0.819	0.973
Sensitivity (in %)	J48	26.1	87.8	86.1	89.3	98.9	94.3	86.2	25.9	26.1	79.9	96.7
	RF	11.2	85.7	83.2	88.1	99.1	94.2	83.5	17	11.2	79.9	99.2
	NB	91.8	95.5	93.9	94.9	94.2	94.1	94.5	91.2	91.8	73.1	91.9
	AB	0	83.3	80.9	86.2	81.6	80.8	80.4	0	0	82.8	84.4
	BG	11.2	85.0	82.8	87.4	98.7	95.7	83.4	16.3	05.2	79.9	95.7
Specificity (in %)	J48	93.3	91.5	92.0	94.2	87.9	87.0	90.6	90.8	93	64.2	88.6
	RF	97.6	97.6	97.3	98.1	93.7	93.1	96.5	97.2	98	73.1	92.1
	NB	27.7	29.9	37.4	29.6	14.6	14.0	25.5	37.3	27.7	79.1	20.6
	AB	100	99.8	99.7	99.8	67.9	68.7	99.8	100	100	66.4	66.8
	BG	97.7	98.0	97.0	97.8	85.6	86.1	96.1	96.9	97.5	71.6	87.9
Precision (in %)	J48	35.7	89.8	89.0	94.1	87.4	85.7	89.1	30.6	35.7	69	88.3
	RF	39.5	96.8	95.9	97.9	93.1	91.9	95.5	49	44.1	74.8	91.8
	NB	15.3	53.7	52.8	58.2	48.3	47.6	531	18.7	15.3	77.8	50.8
	AB	0	99.7	99.5	99.8	68.2	68.2	99.7	0	0	71.2	69.4
	BG	40.5	97.3	95.4	97.6	85.3	85.1	95.0	45.3	22.6	73.8	87.6

Table 5.11 Results for PC4 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.777	0.948	0.940	0.955	0.955	0.937	0.944	0.747	0.777	0.834	0.966
	RF	0.945	0.991	0.989	0.992	0.999	0.99	0.986	0.935	0.946	0.92	0.997
	NB	0.836	0.916	0.908	0.930	0.840	0.818	0.887	0.789	0.836	0.813	0.849
	AB	0.913	0.986	0.982	0.988	0.917	0.925	0.967	0.879	0.913	0.909	0.94
	BG	0.920	0.988	0.985	0.989	0.990	0.981	0.982	0.911	0.927	0.908	0.987
Sensitivity (in %)	J48	48.9	91.3	88.8	93.2	99.2	95.3	91.6	46.4	48.9	87.6	98.7
	RF	37.6	90	87.4	91.8	99.8	96.1	89.5	37.1	37.1	94.9	99.9
	NB	38.2	80	77.6	86.8	44.8	43.9	74.6	27.3	38.2	40.4	41.9
	AB	23.0	90.7	88.7	92.3	89.0	92.5	80.9	22.2	23	92.7	96.2
	BG	46.1	91.6	88.3	91.9	99.3	95.9	90.1	39.2	44.4	92.7	98.2
Specificity (in %)	J48	93.6	94.0	94.6	94.4	91.0	91.5	93.6	93.9	93.6	79.2	92
	RF	98.1	98.0	98.0	98.4	94.7	94.5	97.7	97.9	98.1	81.5	93.7
	NB	93.8	87.3	88.0	85.1	92.7	90.6	85.5	92	93.8	92.1	93
	AB	98.4	94.9	94.6	94.5	79.1	79.2	98.1	97.9	98.4	75.8	78.4
	BG	96.4	96.1	97.0	96.6	90.2	91.0	95.0	96.8	96.7	82	90.6
Precision (in %)	J48	51.5	92.7	91.8	94.5	90.2	90.1	92.6	53.9	51.5	80.8	93
	RF	73.6	97.5	96.8	98.4	94.0	93.5	97.1	73.5	73.3	83.7	94.5
	NB	46.3	84.1	81.3	85.8	83.6	79.3	81.9	34.4	46.3	83.7	86.6
	AB	66.1	93.7	91.7	94.5	78.0	78.5	97.4	61.4	66.1	79.3	82.9
	BG	64.1	95.1	95.3	96.6	89.4	89.7	94.1	65.5	65.3	83.8	91.9

Table 5.12 Results for PC5 Dataset

Performance Metric	Classifier	No sampling	SMOTE	Safe-Level-SMOTE	ADASYN	SPIDER	SPIDER2	SPIDER3	SPY	MUTE	SpreadSub Sample	Resample
AUC	J48	0.817	0.989	0.988	0.988	0.986	0.986	0.985	0.91	0.811	0.944	0.988
	RF	0.977	0.999	0.999	0.999	0.999	0.999	0.999	0.992	0.974	0.973	0.999
	NB	0.937	0.944	0.945	0.945	0.913	0.921	0.933	0.966	0.939	0.946	0.941
	AB	0.959	0.999	0.999	0.999	0.945	0.95	0.997	0.981	0.957	0.957	0.964
	BG	0.975	0.999	0.999	0.999	0.996	0.996	0.999	0.991	0.972	0.963	0.996
Sensitivity (in %)	J48	46.3	98.2	98.4	98.2	97.8	97.1	97.5	79.4	47.4	94.8	99.3
	RF	43.4	98.1	98.0	98.3	98.4	97.6	97.4	79.3	42.5	96.9	99.3
	NB	44.8	63.6	64.8	63.8	47.2	46.7	52.7	76.8	43.9	56.2	48.5
	AB	14.5	97.3	97.6	97.0	86.2	85.3	95.5	76.1	19	90.1	92.3
	BG	39.7	98.0	98.0	98.1	97.7	97.5	97.1	76.3	36.5	94	96.2
Specificity (in %)	J48	99.0	98.9	98.8	99.0	97.6	97.8	99.0	99.4	98.8	92.4	96.7
	RF	99.3	99.4	99.3	99.3	97.9	98.3	99.4	99.7	99.4	93.4	97.4
	NB	98.0	97.0	97.0	96.6	95.2	96.5	96.7	97.1	98	97.7	97.6
	AB	99.7	99.0	98.4	99.5	90.8	91.6	99.6	99	99.5	91.7	90.9
	BG	99.4	99.3	99.3	99.4	97.2	97.3	99.5	99.7	99.4	92.6	99.3
Precision (in %)	J48	60.1	98.9	98.7	99.0	95.7	95.9	98.3	87.4	55.7	92.6	96.8
	RF	67.3	99.3	99.3	99.3	96.4	96.7	99.0	92.7	67	93.6	97.5
	NB	41.1	95.3	95.3	94.9	84.3	87.6	90.2	59.3	40.7	96	95.3
	AB	60	98.9	98.3	99.5	83.8	84.2	99.3	80.6	56.6	91.5	91
	BG	65.9	99.3	99.3	99.4	95.1	94.9	99.2	93.2	67.1	92.7	96.3

- majority of the cases. Sensitivity results show a significantly large percentage increase (6-1490%) in its values in the case of sampling methods as compared to the no sampling scenario. Similarly, precision values also showed a large improvement of 13-945% increase on the application of sampling methods. This improvement in the values of various performance metrics is due to the balancing in datasets. In case of imbalanced datasets, very few instances of defective class were present and hence, they were difficult to learn by the ML techniques which are developed with the assumption that the dataset used to train a particular classifier is balanced. The ML classifiers keep the tendency to classify non-defective examples correctly as they are present in majority. However, balancing in datasets using oversampling methods helps in overcoming the biased nature of the defect prediction models. The increase in defect prone examples made them easy to learn which can be clearly determined from the increased sensitivity values in table 5.1-5.12.

If we observe the specificity results regarding all the 12 datasets, we can conclude that values show a visible decrease of 1-30% in the specificity values. Specificity is the measure of number of non-defective examples which are correctly classified. Due to balancing in datasets, the defective and non-defective examples become equally important to learn while in case of imbalanced datasets, non-defective examples were dominating. The decrease in specificity performance can be considered a concerning factor as it would lead to the testing of some of the non-defective examples which would be a complete wastage of resources, time and effort. In order to develop a good classifier, a balance between sensitivity and specificity should be achieved. However, the balanced datasets provide overall improved performance whereas in case of imbalanced datasets, only specificity results were good. With balancing in datasets, balanced results have been obtained between sensitivity and specificity.

5.2 RQ2: Which is the best oversampling method to improve the performance of ML techniques for software defect prediction in this study?

In order to assess the superiority of various oversampling methods over the scenario when no oversampling method is used, we use Friedman test. The test statistically compares the performance of different oversampling methods. A lower mean rank of a sampling method indicates better comparative performance of that method. We apply Friedman test using AUC, sensitivity and precision performance measures. We do not use specificity for evaluation as it is more biased towards majority class instances and this work focuses more on minority class instances as chances of misclassifying them are high. Moreover, minority

class in this study represents the defective modules which are important to classify correctly because misclassification can lead to project failure and high cost to company. We first state the null and alternate hypothesis investigated by the Friedman test:

Null Hypothesis (H1, H2, H3): The (AUC, sensitivity or precision) results of the defect prediction models developed using five different ML classifiers (J48, RF, NB, ABM1 and BG) are same when no sampling method or six different oversampling methods (ADASYN, Safe-Level-SMOTE, SMOTE, SPIDER, SPIDER2 and SPIDER3) are used to balance the imbalanced datasets.

Alternate Hypothesis (H1a, H2a, H3a): The (AUC, sensitivity or precision) results of the defect prediction models developed using five different ML classifiers (J48, RF, NB, ABM1 and BG) are different when no sampling method or six different oversampling methods (ADASYN, Safe-Level-SMOTE, SMOTE, SPIDER, SPIDER2 and SPIDER3) are used to balance the imbalanced datasets.

5.2.1. Friedman Test Analysis using AUC for Oversampling Methods

Table 5.13 shows the results of Friedman test using AUC performance metric. The last column in the table is the p-value which decides whether the results are significant or not. The test results show that in all the twelve cases, the scenario where no sampling is done (i.e. the case of imbalanced datasets) shows worst results when compared to oversampling methods. Thus, the oversampling methods significantly outperformed the scenario where no sampling was used.

Table 5.13 Friedman Results using AUC for Oversampling Methods

Datasets	Rank1	Rank2	Rank3	Rank4	Rank5	Rank6	Rank7	p-value
CM1	ADASYN	SPIDER	SMOTE	SPIDER2	SPIDER3	S-L-SMOTE	No Sampling	0.009
JM1	SMOTE	S-L-SMOTE	ADASYN	SPIDER	SPIDER3	SPIDER2	No Sampling	0.02
KC2	SMOTE	S-L-SMOTE	ADASYN	SPIDER3	SPIDER2	SPIDER	No Sampling	0
KC3	SMOTE	SPIDER	SPIDER2	S-L-SMOTE	ADASYN	SPIDER3	No Sampling	0.026
MC1	ADASYN	SPIDER	SPIDER2	SPIDER3	S-L-SMOTE	SMOTE	No Sampling	0.016
MC2	SMOTE	S-L-SMOTE	SPIDER	SPIDER3	SPIDER2	ADASYN	No Sampling	0.005
MW1	ADASYN	SPIDER	SPIDER3	SPIDER2	SMOTE	S-L-SMOTE	No Sampling	0.008
PC1	ADASYN	SPIDER	SPIDER3	SPIDER2	SMOTE	S-L-SMOTE	No Sampling	0.006
PC2	ADASYN	SPIDER3	SMOTE	SPIDER	SPIDER2	S-L-SMOTE	No Sampling	0.009
PC3	ADASYN	SPIDER	SMOTE	SPIDER2	S-L-SMOTE	SPIDER3	No Sampling	0.093
PC4	ADASYN	SMOTE	SPIDER	S-L-SMOTE	SPIDER3	SPIDER2	No Sampling	0.001
PC5	ADASYN	SMOTE	S-L-SMOTE	SPIDER3	SPIDER2	SPIDER	No Sampling	0.003

Furthermore, the test describes that ADASYN outperforms all other oversampling methods in majority of the cases (8 out of 12). SMOTE outperforms other oversampling methods in case of four datasets: JM1, KC2, KC3 and MC2 when AUC performance measure is used for evaluation. It can be ascertained that our purposed method SPIDER3 which is an enhancement in SPIDER2, significantly outperforms SPIDER2 in eight out twelve cases. Hence, it can be used as a balancing filter for imbalanced datasets. As all the tests show significant results, we can safely reject the null hypothesis H1.

5.2.2. Friedman Test Analysis using Sensitivity (Recall) for Oversampling Methods

Table 5.14 states the results of Friedman test on sensitivity measure. The last column is the p-value which decides whether to approve or disapprove the null hypothesis. The test results prove that in all the twelve cases, oversampling methods significantly outperform the results of original datasets. Thus, we can safely rule out the null hypothesis H2. Furthermore, the test resulted in the mixed outcomes in case of six oversampling methods. ADASYN outperforms all other oversampling methods in five out of twelve cases while SMOTE and SPIDER family achieves the best rank in six out of twelve cases: KC2, KC3 and CM1, JM1, MC2, PC3 datasets respectively. Safe-Level-SMOTE shows best performance in only one case i.e. PC5. As ADASYN outperforms in majority of the cases when compared to the number of times SPIDER family and SMOTE variants achieved the best rank, we can say that ADASYN is comparatively a better method.

Table 5.14 Friedman Results using Sensitivity for Oversampling Methods

Datasets	Rank1	Rank2	Rank3	Rank4	Rank5	Rank6	Rank7	p-value
CM1	SPIDER	ADASYN	SPIDER2	SMOTE	SPIDER3	S-L-SMOTE	No Sampling	0.008
JM1	SPIDER2	SMOTE	SPIDER	S-L-SMOTE	SPIDER3	ADASYN	No Sampling	0.027
KC2	SMOTE	S-L-SMOTE	SPIDER3	ADASYN	SPIDER2	SPIDER	No Sampling	0
KC3	SMOTE	SPIDER	SPIDER2	SPIDER3	ADASYN	S-L-SMOTE	No Sampling	0.092
MC1	ADASYN	SPIDER	SPIDER2	SPIDER3	SMOTE	S-L-SMOTE	No Sampling	0.015
MC2	SPIDER2	SPIDER	S-L-SMOTE	SMOTE	SPIDER3	ADASYN	No Sampling	0.001
MW1	ADASYN	SPIDER	SPIDER2	SMOTE	SPIDER3	S-L-SMOTE	No Sampling	0.011
PC1	ADASYN	SPIDER	SPIDER2	SPIDER3	SMOTE	S-L-SMOTE	No Sampling	0.013
PC2	ADASYN	SPIDER	SPIDER2	SMOTE	SPIDER3	S-L-SMOTE	No Sampling	0.012
PC3	SPIDER	ADASYN	SMOTE	SPIDER2	SPIDER3	S-L-SMOTE	No Sampling	0.001
PC4	ADASYN	SPIDER	SPIDER2	SMOTE	SPIDER3	S-L-SMOTE	No Sampling	0.003
PC5	S-L-SMOTE	ADASYN	SMOTE	SPIDER	SPIDER3	SPIDER2	No Sampling	0

5.2.3. Friedman Test Analysis using Precision for Oversampling Methods

Table 5.15 shows the results of Friedman test on precision metric. In all the twelve cases, the scenario where no sampling is done shows worst results when compared to all the

other oversampling methods. Furthermore, the test describes that ADASYN significantly outperforms all the other oversampling methods in seven out of twelve cases while SMOTE achieves the best rank in four out of twelve cases: JM1, KC3, MC2 and PC2. Safe-Level-SMOTE shows best performance in only one case i.e. KC2. Hence, we can rule out the null hypothesis H3 as the test shows significant results.

Table 5.15 Friedman Results using Precision for Oversampling Methods

Datasets	Rank1	Rank2	Rank3	Rank4	Rank5	Rank6	Rank7	p-value
CM1	ADASYN	SMOTE	S-L-SMOTE	SPIDER3	SPIDER	SPIDER2	No Sampling	0.001
JM1	SMOTE	S-L-SMOTE	ADASYN	SPIDER3	SPIDER2	SPIDER	No Sampling	0.001
KC2	S-L-SMOTE	SMOTE	ADASYN	SPIDER3	SPIDER2	SPIDER	No Sampling	0
KC3	SMOTE	S-L-SMOTE	ADASYN	SPIDER3	SPIDER2	SPIDER	No Sampling	0
MC1	ADASYN	SPIDER3	S-L-SMOTE	SMOTE	SPIDER	SPIDER2	No Sampling	0
MC2	SMOTE	S-L-SMOTE	SPIDER3	SPIDER	ADASYN	SPIDER2	No Sampling	0
MW1	ADASYN	SMOTE	SPIDER3	S-L-SMOTE	SPIDER	SPIDER2	No Sampling	0
PC1	ADASYN	SMOTE	SPIDER3	S-L-SMOTE	SPIDER	SPIDER2	No Sampling	0
PC2	SMOTE	SPIDER3	ADASYN	S-L-SMOTE	SPIDER2	SPIDER	No Sampling	0
PC3	ADASYN	SMOTE	SPIDER3	S-L-SMOTE	SPIDER	SPIDER2	No Sampling	0
PC4	ADASYN	SMOTE	SPIDER3	S-L-SMOTE	SPIDER	SPIDER2	No Sampling	0
PC5	ADASYN	SMOTE	S-L-SMOTE	SPIDER3	SPIDER2	SPIDER	No Sampling	0

According to above Friedman test results on AUC, sensitivity and precision, it can be noticed that ADASYN outperforms in majority of the cases as compared to other oversampling methods. This is due to the adaptive nature of ADASYN method. As the name suggests, ADAptive SYNthetic minority oversampling, this method adapts itself according to the need to generate synthetic minority samples. This method automatically chooses the value of k (nearest neighbor) and n (amount of oversampling) on the basis of position of each minority sample in the dataset. Unlike SMOTE method and its variants, ADASYN does not generate equal amount of synthetic samples for each minority sample. It focuses more on those minority samples which lie in the safe region and ignores those which are noise. Unlike the other oversampling methods, this method doesn't require user to input the values of k and n. The original dataset is the only requirement as input. Thus, this method turns out to be the best among all the oversampling methods.

5.3 RQ3: What is the comparative performance of the proposed version of SPIDER2 technique i.e. SPIDER3 and the original SPIDER2 technique for software defect prediction?

Although it can be noticed from the above stated Friedman results that SPIDER3 outperforms SPIDER2 in majority of the cases but still to further justify the result we apply

Wilcoxon signed rank test with Bonferroni correction where alpha is set to α=0.05. The hypothesis H4 for Wilcoxon test is stated below.

Null Hypothesis H4: Defect prediction models developed using five different ML classifiers: J48, RF, NB, AB and BG are same when two oversampling methods: SPIDER2 and SPIDER3 are used to balance the imbalanced datasets when AUC, sensitivity and precision performance measures were taken for evaluation.

Alternate Hypothesis H4a: Defect prediction models developed using five different ML classifiers: J48, RF, NB, AB and BG are different when two oversampling methods: SPIDER2 and SPIDER3 are used to balance the imbalanced datasets when AUC, sensitivity and precision performance measures were taken for evaluation.

The Wilcoxon signed rank test performs pairwise comparison of SPIDER2 and SPIDER3 on the performance metric (AUC, sensitivity and precision) values of the defect prediction models developed by all the investigated ML techniques together on all the datasets used in the study. The test depicts that SPIDER3 outperforms SPIDER2 significantly in case of AUC and precision while in case of sensitivity both the methods show comparitive performance. Thus, the results show that SPIDER3 has improved the performance of two important performance measures namely AUC and precision. The improvement is due to the use of SMOTE method in SPIDER3. SPIDER3 uses SMOTE to find synthetic samples for each minority class sample while SPIDER2 simply replicates the existing minority class samples. Hence, the Wilcoxon test results confirms that our proposed method i.e. the modified version of SPIDER2 shows better results when compared to the original method.

5.4 RQ4: Which is the best sampling method among undersampling and resampling methods to improve the performance of ML techniques for software defect prediction in this study?

To find out the best sampling method among various undersampling and resampling methods, we again use Friedman test. We apply Friedman test using AUC, sensitivity and precision performance measures where lower mean rank indicates better performance. The null and alternate hypothesis taken for Friedman test are as follows:

Null Hypothesis (H5, H6, H7): The (AUC, sensitivity or precision) results of the defect prediction models developed using five different ML classifiers (J48, RF, NB, ABM1 and BG) are same when no sampling method or four different sampling methods (SPY, MUTE, SpreadSubSample and Resample) are used to balance the imbalanced datasets.

Alternate Hypothesis (H5a, H6a, H7a): The (AUC, sensitivity or precision) results of the defect prediction models developed using five different ML classifiers (J48, RF, NB, ABM1 and BG) are different when no sampling method or four different sampling methods (SPY, MUTE, SpreadSubSample and Resample) are used to balance the imbalanced datasets.

5.4.1 Friedman Test Analysis using AUC

The test results in table 5.16 show that in all the twelve cases, Resample method significantly outperforms all the other resampling and undersampling methods except for one case i.e. JM1. In JM1 dataset, SPY method shows best results. However, MUTE, SPY and SpreadSubSample shows mixed results leading to the average performance in some cases. On an average, the four sampling methods significantly outperformed the scenario where no sampling was used where resample performs the best. Hence, it can be used as a balancing filter for imbalanced datasets. As majority of the tests show significant results, we can safely reject the null hypothesis H5.

Table 5.16 Friedman Results using AUC

Dataset	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	p-value
CM1	Resample	SPY	MUTE	SpreadSub Sample	No sampling	0.013
JM1	Resample	SPY	No sampling	MUTE	SpreadSub Sample	0.003
KC2	SPY	Resample	MUTE	No sampling	SpreadSub Sample	0.006
KC3	Resample	No sampling	SpreadSub Sample	MUTE	SPY	0.031
MC1	Resample	SPY	MUTE	No sampling	SpreadSub Sample	0.562
MC2	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.007
MW1	Resample	SpreadSub Sample	No sampling	SPY	MUTE	0.005
PC1	Resample	MUTE	SPY	No sampling	SpreadSub Sample	0.041
PC2	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.020
PC3	Resample	SpreadSub Sample	No sampling	MUTE	SPY	0.166
PC4	Resample	MUTE	No sampling	SpreadSub Sample	SPY	0.003
PC5	Resample	SPY	No sampling	SpreadSub Sample	MUTE	0.017

5.4.2. Friedman Test Analysis using Sensitivity (Recall)

Table 5.17 states the results of Friedman test on sensitivity measure. The test results prove that in all the twelve cases, resample method outperforms all the other methods in the first place while SpreadSubSample method achieves the second rank. In six out of twelve cases the no sampling scenario achieves worst rank while the other methods significantly improve the performance of prediction models. Hence, we can safely rule out the null hypothesis H6. Furthermore, the test resulted in the mixed outcomes in case of SPY and MUTE sampling methods. However, SPY achieves rank third in majority of the cases.

Table 5.17 Friedman Results using Sensitivity

Dataset	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	p-value
CM1	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.001
JM1	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.002
KC2	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.015
KC3	Resample	SpreadSub Sample	No sampling	SPY	MUTE	0.007
MC1	Resample	SpreadSub Sample	SPY	No sampling	MUTE	0.013
MC2	Resample	SpreadSub Sample	SPY	No sampling	MUTE	0.011
MW1	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.001
PC1	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.002
PC2	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.001
PC3	Resample	SpreadSub Sample	No sampling	SPY	MUTE	0.011
PC4	Resample	SpreadSub Sample	No sampling	MUTE	SPY	0.001
PC5	Resample	SpreadSub Sample	SPY	No sampling	MUTE	0.002

5.4.3. Friedman Test Analysis using Precision

Table 5.18 shows the results of Friedman test on precision metric. In seven out of the twelve cases, the scenario where no sampling is done shows worst results when compared to all the other sampling methods. Furthermore, the test describes that resample, SpreadSubSample and SPY method significantly outperforms the no sampling scenario at first second and third rank respectively. However, MUTE could not perform well in four out of twelve cases.

Table 5.18 Friedman Results using Precision

Dataset	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	p-value
CM1	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.001
JM1	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.001
KC2	Resample	SpreadSub Sample	SPY	No sampling	MUTE	0.001
KC3	Resample	SpreadSub Sample	SPY	No sampling	MUTE	0.002
MC1	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.001
MC2	Resample	SpreadSub Sample	SPY	No sampling	MUTE	0.007
MW1	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.001
PC1	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.002
PC2	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.000
PC3	Resample	SpreadSub Sample	SPY	MUTE	No sampling	0.002
PC4	Resample	SpreadSub Sample	No sampling	SPY	MUTE	0.002
PC5	Resample	SpreadSub Sample	SPY	No sampling	MUTE	0.002

According to above Friedman test results on AUC, sensitivity and precision, it can be noticed that Resample method outperforms in majority of the cases as compared to other sampling methods. This is due to the biasToUniformClass parameter which helps to resample the dataset so as to achieve a good balanced ratio between minority and majority class instances. However, an average performance of MUTE is due to its strict rule to eliminate

majority class samples. MUTE method removes majority class instance and declares it noisy if and only if all its k nearest neighbors are minority class samples. This makes it difficult to achieve balance between two classes as it is not possible for maximum of the majority samples to have all its k nearest neighbors as minority samples because of latter being less in number. Researchers should decrease the threshold value for noisy samples to a reasonable digit so as to achieve better performance with respect to MUTE method.

5.5 RQ5: Which sampling technique is the best among oversampling, undersampling and resampling techniques and why?

From the above Friedman tests, it can be noticed that ADASYN and Resample are the best methods among oversampling and resampling techniques respectively. In order to further justify which method is the best between the two we apply Wilcoxon signed rank test with Bonferroni correction where alpha is set to α=0.05. The hypothesis H8 for Wilcoxon test is stated below.

Null Hypothesis H8: Defect prediction models developed using five different ML classifiers: J48, RF, NB, AB and BG are same when two sampling methods: ADASYN and Resample are used to balance the imbalanced datasets when AUC, sensitivity and precision performance measures were taken for evaluation.

Alternate Hypothesis H8a: Defect prediction models developed using five different ML classifiers: J48, RF, NB, AB and BG are different when two sampling methods: ADASYN and Resample are used to balance the imbalanced datasets when AUC, sensitivity and precision performance measures were taken for evaluation.

The Wilcoxon signed rank test performs pairwise comparison of ADASYN and Resample on the performance metric (AUC, sensitivity and precision) values of the defect prediction models developed by all the investigated ML techniques together on all the datasets used in the study. The test depicts that both the techniques are efficient in their own ways. ADASYN outperforms Resample in case AUC and precision while Resample outperforms ADASYN in case of sensitivity. ADASYN dominates precision and AUC test results significantly while Resample dominates recall results.

This is due to the biasness removing nature of the resample method. The biasToUniformClass parameter helps to achieve an effective balance between the two classes. This further helps to improve the correct prediction of minority class instances to a significant

level. Thus, correct prediction of minority class leads to the better sensitivity results. Furthermore, ADASYN's best performance in case of AUC and precision is due to its adaptive nature which helps to balance the data by multiplying each minority class sample using the synthetic sample generation method. It distributes the synthetic samples among each minority class sample on the basis of density distribution function. Density distribution function determines the number of synthetic samples that should be produced corresponding to each minority class sample.

5.6 RQ6: What is the effect of using MC learners on imbalanced datasets for software defect prediction?

This work also uses cost sensitive learning to handle the imbalanced data problem in software defect prediction. We use three different cost ratios: 10, 30 and 50 to cost sensitize the various ML classifiers used in this study. We compare the use of MC learners with the scenario in which the original learners are used to build ML models. Tables 5.19-5.30 describes the results obtained by using different cost ratios in MC learners as well as those obtained on original datasets.

It can be observed from the tables 5.19-5.30 that the average performance of ML techniques improved (1-47%) in 6 out of 12 datasets in terms of AUC when MC learners were used in comparison to the original scenario. However, the average performance of ML techniques decreased in the remaining datasets. The results show that the MC with cost ratio 10 outperforms the other MC learners in majority of the cases when AUC was used while in case of sensitivity MC with cost ratio 50 outperforms the other MC learners as well as the original scenario. In fact, MC learners with all the cost ratios outperforms the original dataset in case of sensitivity with percentage increase of 10-600%. This outcome is because the cost values of MC learners were set in such a way so as to decrease the number of false negative predictions. The lower the number of FNs the higher will be the chance of correctly classifying defective modules. Thus, penalizing the classifier for false negatives lead to the increase in sensitivity in all the datasets. But in order to develop the better predictive models the balance between specificity and sensitivity must be achieved. Table 5.19-5.30 also show that the precision values decreased in majority of the cases. However, the average performance of ML techniques improved in case of two datasets: CM1 and PC2. But overall decrease in precision values is a concerning factor and it should be researchers' aim to achieve balance among all the performance metrics.

The study also performed Wilcoxon signed rank test in order to make a pairwise comparison between MC learners and the original dataset scenario. The test was performed on AUC, sensitivity and precision values of the defect prediction models developed by the application of five ML techniques (J48, RF, NB, AB and BG) on all the datasets (original as well as MC learners) of the study. It was observed that original scenario outperformed MC learners in case of AUC but non-significantly while in case of sensitivity MC learners significantly outperformed the original dataset. As the defect prone classes are important to learn correctly, MC learners help improve the prediction of defective modules and hence improves the quality of the product. However, they may not always yield improved results.

Table 5.19 MC results for CM1 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.594	0.581	0.657	0.678
	RF	0.763	0.780	0.738	0.727
	NB	0.694	0.686	0.684	0.688
	AB	0.717	0.754	0.753	0.736
	BG	0.727	0.760	0.711	0.671
Sensitivity (in %)	J48	26.2	59.5	59.5	73.8
	RF	0	71.4	90.5	95.2
	NB	33.3	52.4	64.3	64.3
	AB	0	90.5	95.2	97.6
	BG	0	78.6	92.9	95.2
Specificity (in %)	J48	93.7	67.2	67.9	55.3
	RF	99.0	71.5	39.7	30.8
	NB	89.1	71.2	64.6	64.2
	AB	100.0	56.3	39.4	27.8
	BG	99.0	64.2	36.8	22.5
Precision (in %)	J48	36.7	20.2	20.5	18.7
	RF	0	25.9	17.3	16.1
	NB	29.8	20.2	20.1	20.0
	AB	0	22.4	17.9	15.8
	BG	0	23.4	17.0	14.6

Table 5.20 MC results for JM1 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.616	0.623	0.535	0.483
	RF	0.701	0.689	0.650	0.613
	NB	0.633	0.650	0.650	0.650
	AB	0.669	0.500	0.500	0.500
	BG	0.689	0.682	0.587	0.500
Sensitivity (in %)	J48	22.5	75.2	94.6	99.7
	RF	20.5	79.4	97.2	99.1
	NB	18.5	27.8	28.8	28.8

	AB	0	100.0	100.0	100.0
	BG	17.2	90.0	99.8	100.0
Specificity (in %)	J48	91.2	48.4	96	70.0
	RF	95.5	44.9	82	024
	NB	94.5	90.7	90.4	90.3
	AB	100.0	0	0	0
	BG	95.5	27.1	70.0	0
Precision (in %)	J48	41.0	28.5	22.3	21.5
	RF	55.3	28.3	22.5	21.7
	NB	48.1	45.0	45.0	44.8
	AB	0	21.5	21.5	21.5
	BG	51.3	25.3	21.6	21.5

Table 5.21 MC results for KC2 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.704	0.728	0.418	0.471
	RF	0.825	0.812	0.723	0.702
	NB	0.832	0.830	0.831	0.832
	AB	0.784	0.804	0.570	0.499
	BG	0.825	0.811	0.548	0.500
Sensitivity (in %)	J48	49.5	79.4	92.5	100.0
	RF	47.7	84.1	92.5	92.5
	NB	42.1	57.0	57.0	57.0
	AB	43.9	85.0	96.3	98.1
	BG	43.0	86.0	96.3	100.0
Specificity (in %)	J48	89.6	72.8	11.8	0
	RF	92.5	69.6	45.3	34.9
	NB	94.2	89.4	89.2	89.2
	AB	91.1	69.9	12.8	024
	BG	94.2	71.3	13.5	0
Precision (in %)	J48	55.2	42.9	21.3	20.5
	RF	62.2	41.7	30.4	26.8
	NB	65.2	58.1	57.5	57.5
	AB	56.0	42.1	22.2	20.6
	BG	65.7	43.6	22.3	20.5

Table 5.22 MC results for KC3 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.653	0.670	0.692	0.517
	RF	0.736	0.692	0.644	0.646
	NB	0.661	0.663	0.656	0.652
	AB	0.573	0.624	0.528	0.506
	BG	0.729	0.630	0.498	0.500
Sensitivity (in %)	J48	33.3	69.4	75.0	86.1
	RF	13.9	72.2	94.4	97.2
	NB	38.9	50.0	52.8	52.8
	AB	36.1	75.0	97.2	97.2

	BG	13.9	75.0	97.2	100.0
Specificity (in %)	J48	89.9	60.8	54.4	10.8
	RF	96.8	58.9	18.4	10.1
	NB	88.0	71.5	69.0	68.4
	AB	93.0	47.5	032	019
	BG	93.7	43.0	013	0
Precision (in %)	J48	42.9	28.7	27.3	18.0
	RF	50.0	28.6	20.9	19.8
	NB	42.4	28.6	27.9	27.5
	AB	54.2	24.5	18.6	18.4
	BG	33.3	23.1	18.3	18.6

Table 5.23 MC results for MC1 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.521	0.682	0.604	0.643
	RF	0.883	0.917	0.877	0.868
	NB	0.708	0.704	0.713	0.708
	AB	0.842	0.793	0.816	0.812
	BG	0.860	0.787	0.853	0.834
Sensitivity (in %)	J48	8.7	30.4	30.4	37.0
	RF	19.6	39.1	63.0	71.7
	NB	32.6	58.7	65.2	71.7
	AB	0	26.1	63.0	76.1
	BG	0	19.6	56.5	76.1
Specificity (in %)	J48	99.5	96.1	96.3	95.8
	RF	99.5	98.1	91.4	85.8
	NB	90.2	72.9	68.4	66.6
	AB	100.0	96.7	81.5	73.2
	BG	100.0	98.3	89.9	81.9
Precision (in %)	J48	28.6	15.7	16.3	17.2
	RF	47.4	32.7	14.8	10.7
	NB	7.3	4.9	4.7	4.8
	AB	0	15.8	7.5	6.3
	BG	0	21.4	11.7	9

Table 5.24 MC results for MC2 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.698	0.575	0.575	0.566
	RF	0.717	0.576	0.534	0.498
	NB	0.702	0.676	0.674	0.674
	AB	0.616	0.588	0.510	0.499
	BG	0.676	0.502	0.500	0.500
Sensitivity (in %)	J48	52.3	75.0	88.6	97.7
	RF	38.6	93.2	100.0	100.0
	NB	38.6	47.7	47.7	47.7
	AB	40.9	97.7	100.0	100.0
	BG	38.6	100.0	100.0	100.0

Specificity (in %)	J48	81.5	29.6	27.2	8.6
	RF	87.7	18.5	0	0
	NB	90.1	80.2	77.8	76.5
	AB	85.2	14.8	025	0
	BG	86.4	0	0	0
Precision (in %)	J48	60.5	36.7	39.8	36.8
	RF	63.0	38.3	35.2	35.2
	NB	68.0	56.8	53.8	52.5
	AB	60.0	38.4	35.8	35.2
	BG	60.7	35.2	35.2	35.2

Table 5.25 MC results for MW1 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.449	0.583	0.661	0.562
	RF	0.716	0.726	0.709	0.733
	NB	0.728	0.711	0.705	0.703
	AB	0.711	0.668	0.688	0.678
	BG	0.705	0.745	0.700	0.500
Sensitivity (in %)	J48	14.8	40.7	59.3	92.6
	RF	18.5	55.6	74.1	81.5
	NB	55.6	59.3	63.0	63.0
	AB	29.6	55.6	85.2	88.9
	BG	11.1	63.0	88.9	100.0
Specificity (in %)	J48	95.6	77.4	68.1	14.6
	RF	96.0	82.3	53.5	35.4
	NB	84.5	73.5	72.6	72.6
	AB	94.7	81.9	47.8	33.6
	BG	98.7	80.1	21.7	0
Precision (in %)	J48	28.6	17.7	18.2	11.5
	RF	35.7	27.3	16.0	13.1
	NB	30.0	21.1	21.5	21.5
	AB	40.0	26.8	16.3	13.8
	BG	50.0	27.4	11.9	10.7

Table 5.26 MC results for PC1 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.719	0.691	0.801	0.803
	RF	0.844	0.848	0.850	0.842
	NB	0.768	0.717	0.716	0.718
	AB	0.793	0.802	0.804	0.810
	BG	0.820	0.816	0.792	0.776
Sensitivity (in %)	J48	24.6	55.7	72.1	88.5
	RF	13.1	49.2	93.4	98.4
	NB	36.1	42.6	45.9	47.5
	AB	0	85.2	95.1	96.7
	BG	8.2	63.9	93.4	96.7
Specificity	J48	96.7	81.1	77.9	71.3

(in %)	RF	98.1	88.5	64.8	56.2
	NB	92.8	80.1	78.2	77.7
	AB	99.6	67.0	56.4	53.4
	BG	99.4	78.7	56.3	49.1
Precision (in %)	J48	39.5	20.5	22.2	21.3
	RF	38.1	27.3	18.8	16.4
	NB	30.6	15.8	15.6	15.7
	AB	0	18.4	16.0	15.4
	BG	55.6	20.7	15.7	14.3

Table 5.27 MC results for PC2 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.448	0.652	0.662	0.620
	RF	0.836	0.814	0.892	0.927
	NB	0.877	0.867	0.862	0.860
	AB	0.914	0.826	0.880	0.880
	BG	0.828	0.849	0.860	0.858
Sensitivity (in %)	J48	0	18.8	25.0	18.8
	RF	0	063	37.5	56.3
	NB	31.3	62.5	62.5	62.5
	AB	6.3	37.5	68.8	75.0
	BG	0	18.8	43.8	62.5
Specificity (in %)	J48	99.7	97.7	97.1	96.8
	RF	99.9	99.2	94.5	92.0
	NB	96.1	84.2	82.1	81.6
	AB	99.8	97.6	93.3	91.3
	BG	100.0	99.5	95.3	92.5
Precision (in %)	J48	0	7.7	8.2	5.7
	RF	0	7.1	6.5	6.0
	NB	7.6	3.9	3.4	3.4
	AB	25.0	14.0	9.5	8.1
	BG	0	27.3	8.8	7.8

Table 5.28 MC results for PC3 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.616	0.710	0.744	0.733
	RF	0.831	0.819	0.807	0.795
	NB	0.766	0.564	0.548	0.531
	AB	0.791	0.809	0.787	0.789
	BG	0.824	0.806	0.768	0.774
Sensitivity (in %)	J48	26.1	68.7	77.6	82.8
	RF	11.2	78.4	94.8	96.3
	NB	91.8	94.8	94.8	95.5
	AB	0	85.8	93.3	96.3
	BG	11.2	85.8	91.8	97.8
Specificity (in %)	J48	93.3	72.0	70.5	66.9
	RF	97.6	74.4	53.6	43.5

	NB	27.7	099	067	069
	AB	100.0	63.8	45.8	35.4
	BG	97.7	68.6	50.3	38.1
Precision (in %)	J48	35.7	25.8	27.2	26.2
	RF	39.5	30.3	22.5	19.5
	NB	15.3	13.0	12.6	12.7
	AB	0	25.2	19.7	17.5
	BG	40.5	28.0	20.8	18.3

Table 5.29 MC results for PC4 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.777	0.859	0.850	0.857
	RF	0.945	0.918	0.890	0.877
	NB	0.836	0.728	0.730	0.734
	AB	0.913	0.881	0.855	0.874
	BG	0.920	0.900	0.858	0.833
Sensitivity (in %)	J48	48.9	84.8	88.2	91.0
	RF	37.6	91.6	98.9	99.4
	NB	38.2	44.4	47.2	47.8
	AB	23.0	97.2	97.8	97.8
	BG	46.1	94.4	97.8	97.8
Specificity (in %)	J48	93.6	83.8	81.5	81.1
	RF	98.1	80.4	67.7	60.8
	NB	93.8	83.8	82.3	81.9
	AB	98.4	72.3	68.7	65.0
	BG	96.4	77.4	69.6	68.8
Precision (in %)	J48	51.5	42.1	39.8	40.1
	RF	73.6	39.4	29.9	26.1
	NB	46.3	27.6	27.0	26.8
	AB	66.1	32.8	30.3	28.0
	BG	64.1	36.8	30.9	30.3

Table 5.30 MC results for PC5 Dataset

Performance Metric	Classifier	Original	MC(10)	MC(30)	MC(50)
AUC	J48	0.817	0.871	0.908	0.906
	RF	0.977	0.971	0.965	0.964
	NB	0.937	0.930	0.929	0.929
	AB	0.959	0.960	0.959	0.956
	BG	0.975	0.966	0.968	0.963
Sensitivity (in %)	J48	46.3	77.1	83.1	85.3
	RF	43.4	88.2	94.4	96.3
	NB	44.8	70.5	70.9	70.9
	AB	14.5	87.2	91.9	92.1
	BG	39.7	88.4	93.4	95.3
Specificity (in %)	J48	99.0	96.5	96.0	95.4
	RF	99.3	95.8	93.8	92.8
	NB	98.0	93.8	93.7	93.7

	AB	99.7	94.2	91.1	90.0
	BG	99.4	95.6	93.9	93.0
Precision (in %)	J48	60.1	40.6	39.2	36.7
	RF	67.3	39.3	32.0	29.2
	NB	41.1	26.1	26.0	25.9
	AB	60.0	31.7	24.2	22.2
	BG	65.9	38.4	32.0	29.7

5.7 RQ7: What is the comparative performance of best sampling method and MC learners for software defect prediction?

According to the findings in RQ5, ADASYN is the overall best sampling method to handle imbalanced datasets. This RQ compares ADASYN with MC learners using Wilcoxon signed rank test. The result of the pairwise comparison of ADASYN and MC learners was evaluated on AUC, sensitivity and precision using all the datasets of the study together where models were developed by the application of five ML techniques (J48, RF, NB, AB, BG). According to the results, ADASYN method significantly outperformed the MC learners in case of AUC and precision. However, MC learners outperform ADASYN non-significantly in case of sensitivity. This is due to the fact that MC learners focus more on correct prediction of defective class instances by cost sensitizing the classifiers for false negatives leading to the increase in sensitivity values.

However, the overall best performance of ADASYN in all the cases is because of its property to balance the data using density distribution method. The synthetic samples to be generated may vary in number for every minority class sample depending upon the value of density distribution function. Density distribution helps in defining the region in which synthetic samples should be yielded for each minority class sample. This property makes ADASYN works automatically while in other sampling methods as well as in case of MC learners, we are supposed to set one or more parameters (k, n or cost ratios) manually.

CHAPTER 6

CONCLUSION

This study ascertains if balancing the datasets improves the performance of ML techniques in software defect prediction. The study uses five ML classifiers namely J48, RF, NB, AB and BG to develop defect prediction models. In order to handle imbalanced data, the study uses nine existing sampling methods: SMOTE, ADASYN, Safe-Level-SMOTE, SPIDER, SPIDER2, MUTE, SPY, SpreadSubSample and Resample. As only the SMOTE method is used in most of the previous studies, we implemented all the above mentioned sampling methods in MATLAB environment in order to perform our analysis. We also proposed a modified version of SPIDER2 i.e. SPIDER3 and implemented the same. Furthermore, MC learners were also evaluated to ascertain their effectiveness in improving the results of the developed defect prediction models on imbalanced datasets. Moreover, a comparative analysis between MC learners and sampling methods was also performed. The empirical validation was done using AUC and three traditional metrics: sensitivity, precision and specificity and the outcomes of the study were statistically assessed.

6.1 The Conclusions of the Work

- A significant improvement was observed in the performance of ML techniques when sampling methods were used to handle imbalancing in datasets. Moreover, ADASYN was observed to be the best oversampling method among others due to its adaptive nature and capability to balance the data automatically using density distributions. In addition, other oversampling methods also performed well. SMOTE and SPIDER showed comparative results followed by other techniques.
- Resample method outperformed among resampling and undersampling methods. It shows the best sensitivity results among all the sampling methods including oversampling. This is due to its unbiased nature to resample the data by setting biasToUniformClass parameter to an optimum value.
- MUTE undersampling method can be improved for developing better prediction models by relaxing the threshold value used to identify noisy majority class samples to a reasonable amount. MUTE method removes majority class instance and declares it noisy

if and only if all its k nearest neighbors are minority class samples. This makes it difficult to achieve balance between two classes leading to the poor development of prediction models.

- The proposed method SPIDER3 i.e. the modified version of SPIDER2 significantly outperforms SPIDER2 method in case of AUC and precision while it showed comparative results when evaluated using sensitivity measure. The modified version only improved the performance of existing one by generating synthetic samples per each defective class sample rather than just replicating them.
- MC learners are another effective way to handle the imbalancing problem. They cost sensitize the classifier in order to improve predictable nature of the predictive models by using different cost ratios for various misclassification errors. They outperformed the results of original datasets in case of sensitivity. However, AUC and precision results were average. They showed lower performance in comparison to original non cost-sensitized learners i.e. oversampling methods when evaluated using AUC and precision.
- A pairwise comparison between ADASYN and MC learners with best cost ratio concluded that ADASYN is the better method to handle imbalancing problem as compared to MC learners. ADASYN significantly outperformed in case of AUC and precision. Although MC learner with cost ratio 50 outperformed ADASYN in few cases when evaluated using sensitivity but the results were non-significant. Moreover, ADASYN provides balanced results among various performance measures, which is in fact necessary for better development of defect prediction models.

6.2 Future Scope

The analysis performed in this study can be used to develop efficient defect prediction models in case of imbalanced data problem. The future work will focus on another category of balancing methods i.e. ensemble methods using inter-cross validation method. Furthermore, the future research may include a statistical comparison between sampling and ensemble methods for the betterment of defect prediction models in case of imbalanced data.

REFERENCES

[1] J. Stefanowski, Sz. Wilk, “Selective Pre-processing of Imbalanced Data for Improving Classification Performance” In Proc. of 10th Int. Conference DaWaK 2008, LNCS vol. 5182, Springer Verlag,, 283 292, 2008.

[2] C. Bunkhumpornpat, K. Sinapiromsaran, C. Lursinsap, “safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem”, in: Advances in Knowledge Discovery and Data Mining,, pp. 475–482, 2009.

[3] H. He, Y. Bai, E. A. Garcia, and S. Li, “ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning”, International Joint Conference on Neural Networks (IJCNN 2008).

[4] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Oversampling Technique,” Journal of Artificial Intelligence Research, vol. 16, pp. 321-357, 2002.

[5] A. Syaripudin and M. L. Khodra, “A Comparison for Handling Imbalanced Datasets”, International Conference of Advanced Informatics: Concept, Theory and Application (ICAICTA), 2014.

[6] V. Lopez, A. Fernandez, S. Garcia, V. Palade, F. Herrera, “An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics”, Information Sciences 250, journal homepage: www.elsevier.com/locate/ins, pp. 113–141, 2013.

[7] D. Rodriguez, I. Herraiz and R. Harrison, “Preliminary Comparison of Techniques for Dealing with Imbalance in Software Defect Prediction”, Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, Article No. 43, 2014.

[8] M.J. Siers, 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 Volume 51, Pages 62–71, July 2015.

[9] J. Chen, S. Liu and W. Liu, “A Two-Stage Data Preprocessing Approach for Software Fault Prediction”, Software Security and Reliability, 2014 Eighth International Conference, 15 September 2014.

[10] M. Liu, L. Miao and D. Zhang, “Two-Stage Cost-Sensitive Learning for Software Defect Prediction”, IEEE Transactions on Reliability, Volume: 63, Issue: 2, June 2014.

[11] S. Wang and X. Yao, “Using Class Imbalance Learning for Software Defect Prediction”, Reliability, IEEE Transactions, pg 434-443, June 2013.

[12] R. Shatnawi, “Improving software fault-prediction for imbalanced data”, Innovations in Information Technology (IIT), 2012 International Conference, pg 54-59, March 2012.

[13] T. M. Khoshgoftaar and K. Gao, “Feature Selection with Imbalanced Data for Software Defect Prediction”, Proceedings of the 2009 International Conference on machine learning and Applications, p.235-240, December 13-15, 2009.

[14] Y. Kamei, A. Monden and S. Matsumoto, “The Effects of Over and Under Sampling on Fault-prone Module Detection”, Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium, Oct 2007.

[15] J. C. Riquelme, R Ruiz, D Rodr´ıguez, and J Moreno, “Finding Defective Modules from Highly Unbalanced Datasets”, Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos, Vol. 2, No. 1, 2008.

[16] R. Malhotra and A. Jain, "Fault Prediction Using Statistical and machine learning Methods for Improving Software Quality," Journal of Information Processing Systems, vol. 8, no. 2, pp. 241-262, 2012.

[17] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction”, Applied Soft Computing 27, 504-518, 2015.

[18] D. Ramyachitra and P. Manikandan, “Imbalanced dataset classification and solutions: a review”, International Journal of Computing and Business Research (IJCBR), ISSN (Online): 2229-6166, Volume 5 Issue 4 July, 2014.

[19] S. Kotsiantis, D. Kanellopoulos and P. Pintelas, "Handling imbalanced datasets: A review," GESTS International Transactions on Computer Science and Engineering, Vol.30, 2006.

[20] M. Tan, L. Tan, S. Dara and C. Mayeux, "Online Defect Prediction for Imbalanced Data", IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015.

[21] R. Malhotra and M. Khanna, "An empirical evaluation for software change prediction using imbalanced data", Automated Software Engineering, Springer, August 2016.

[22] P. Domingos, "MetaCost: A General Method For Making Classifiers Cost Sensitive", In Proc. of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, CA, 155–164, 1999.

[23] M. Halstead, "Elements of Software Science", Elsevier, New York, 1977.

[24] T. McCabe, "A complexity measure", IEEE Trans. Softw. Eng. 2, 308–320, 1976.

[25] S Lessmann, B. Baesans,C. Mues,S. Pietsch, "Benchmarking classification models for software defect prediction: a proposed framework and novel finding" IEEE Trans on Softw Eng,Vol 34, 485–496, july/august 2008.

[26] C. Catal and B. Diri, "A systematic review of software fault prediction studies," Expert Systems with Applications,vol. 36, pp. 7346-7354, 2009.

[27] A. Chug, S. Dhall , "Software Defect Prediction Using Supervised Learning Algorithm and Unsupervised Learning Algorithm", Confluence 2013: The Next Generation Information Technology Summit (4th International Conference), pg 5.01, 2013.

[28] C. Catal, B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem", Information Sciences 179, 1040–1058, 2009.

[29] C. Catal, "Software fault prediction: a literature review and current trends", Expert Syst. Appl. 38, 4626–4636, 2011.

[30] I. Gondra, "Applying machine learning to software fault-proneness prediction," The Journal of Systems and Software, vol. 81, pp. 186-195, 2008.

[31] Z. Li and M. Reformat, "A practical method for the software fault-prediction", Information Reuse and Integration, IRI. IEEE International Conference, IEEE, Las Vegas, IL, August, 2007.

[32] E. Hong, "Software fault-proneness Prediction using random forest", International Journal of Smart Home Vol. 6, No. 4, October, 2012.

[33] A. Shanthini and R. M. Chandrasekaran, "Applying machine learning for Fault Prediction Using Software Metrics", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 6, June 2012.

[34] Y. Singh, R. Malhotra, A. Kaur, "Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines", Int. Journal of System Assur. Eng. Management, 1(3):269–281, July-Sept, 2010.

[35] P. Jeatrakul, K. W. Wong, and C. C. Fung, "Classification of Imbalanced Data by Combining the Complementary Neural Network and SMOTE Algorithm", Springer-Verlag Berlin Heidelberg, ICONIP, Part II, LNCS 6444, pp. 152–159, 2010.

[36]T. Menzies, A. Dekhtyar, J. Distefance, J. Greenwald, "Problems with precision:a response to comments on 'data mining static code attributes to learn defectpredictors", IEEE Trans. Softw. Eng. 33 637–640, 2007.

[37] H. He and E. A. Garcia, "Learning from imbalanced data." IEEE Trans on Knowledge Data Eng Vol 21, 1263–1284, 2009.

[38] K. Gao, T. M. Khoshgoftaar, A. Napolitano, "Combining feature subset selection and data sampling for coping with highly imbalanced software data.", In Proc. of 27th International Conf. on Software Engineering and Knowledge Engineering, Pittsburgh, 2015.

[39]J. Demšar, "Statistical comparisons of classifiers over multiple data sets.", J Mach Learn Res Vol 7, 1–30, 2006

[40] L. Breiman, "Random Forests.", Machine Learning Vol 45, 5–32, 2001.

[41] K. P. Murphy, "Naïve Bayes classifiers", Technical Report, 2006.

[42] I. H. Witten, E. Frank, M. A. Hall, "Data mining: practical machine learning, tools and techniques", 3rd edition. Morgan Kaufmann, San Francisco, 2011.

[43] L. Breiman, "Bagging predictors." Machine Learning Vol 24, 123–140, 1996.

[44] K. Napierala, J. Stefanowski, S. Wilk, "Learning from Imbalanced Data in Presence of Noisy and Borderline Examples", Springer-Verlag Berlin Heidelberg, RSCTC, LNAI 6086, pp. 158–167, 2010.

[45] D. R. Wilson and T. R. Martinez, "Reduction Techniques for Instance-Based Learning Algorithms", Machine learning, 38, 257–286, 2000.

[46] T. Fawcett, “An introduction to ROC analysis.”, Pattern Recogn Lett Vol 27, 861–874, 2006.

[47] G. J. Pai, J. B. Dugan, “Empirical analysis of software fault content and fault proneness using Bayesian methods.”, IEEE Trans on Softw Eng, Vol 33, 675–686, 2007.

[48] N. Seliya and T.M. Khoshgoftaar, “The use of decision trees for costsensitive classification: an empirical study in software quality prediction.” Wiley Interdiscip Rev: Data Min Knowl Disc 1, pg 448–459, 2011.

[49] G.M. Weiss, K. McCarthy and B. Zabar, “Costsensitive learning vs. sampling: which is best for handling unbalanced classes with unequal error costs?”, In Proc. of International Conf. on Data Mining, pg 35–41, 2007.

[50] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince and F. Herrera, “A review on ensembles for the class imbalance problem: Bagging, boosting, and hybridbased approaches.”, IEEE Trans Syst Man Cybern Part C Appl Rev 42, pg 463–484, 2012.

[51] M. Zięba, “Service-oriented medical system for supporting decisions with missing and imbalanced data”, IEEE Journal of Biomedical and Health Informatics, Vol 18, pg 1533 – 1540, May. 2014.

[52] Z. Miao, L. Zhao, W. Yuan, “Multiclass imbalanced learning implemented in network intrusion detection”, Computer Science and Service System (CSSS), IEEE International Conference, 2011.

[53] T. M. Padmaja, N. Dhulipalla, R. S. Bapi, P. R. Krishna, “Unbalanced data classification using extreme outlier elimination and oversampling methods for fraud detection”, 15th International Conference on Advanced Computing and Communications (ADCOM), pg 511 – 516, 2007.

[54] J. Li, Q. Du, W. Li, Y. Li, “Representation based hyperspectral image classification with imbalanced data”, IEEE International Geoscience and Remote Sensing Symposium (IGARSS), pg 3318 – 3321, 2016.

[55] T. Zimmermann, N. Nagappan and A. Zeller, “Predicting bugs from history”, Software Evolution, Springer, pp 69-88, 2008.

[56] R. Malhotra, N. Pritam and Y. Singh, "On the applicability of evolutionary computation for software defect prediction", International Conference on Advances in Computing Communications and Informatics (ICACCI), 2014.

[57] C. Bunkhumpornpat, K. Sinapiromsaran and C. Lursinsap, “MUTE: Majority Under-sampling Technique”, ICICS, 2011.

[58] X. T. Dang, D. H. Tran, O. Hirose, K. Satou, “SPY: a novel resampling method for improving classification performance in imbalanced data”, Seventh International Conference on Knowledge and Systems Engineering, 2015.

[59] http://weka.sourceforge.net/doc.stable/weka/filters/supervised/instance/SpreadSubsamp-le.html

[60] http://weka.sourceforge.net/doc.stable/weka/filters/supervised/instance/Resample.html