CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

An Effective Undersampling Approach to Deal with Class Imbalance Problem in Software Defect Prediction

by

Syed Fawad-ul-Hassan Gillani

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2021

My dissertation work is devoted to my loving Parents, My Teachers and My Friends. I would like to pay special gratitude to my supervisor who guided me throughout my work.

CERTIFICATE OF APPROVAL

. An Effective Undersampling Approach to Deal with Class Imbalance Problem in Software Defect Prediction

by

Syed Fawad-ul-Hassan Gillani

(MCS173033)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Yaser Hafeez	Arid Agriculture University
(b)	Internal Examiner	Dr. Saima Nazir	CUST, Islamabad
(c)	Supervisor	Dr. Aamer Nadeem	CUST, Islamabad

Dr. Aamer Nadeem
Thesis Supervisor
May, 2021

Dr. Nayyer Masood
Head
Dept. of Computer Science
May, 2021

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
May, 2021

Author's Declaration

I, **Syed Fawad-ul-Hassan Gillani** hereby state that my MS thesis titled "**An Effective Undersampling Approach to Deal with Class Imbalance Problem in Software Defect Prediction**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

(Syed Fawad-ul-Hassan Gillani)
Registration No: MCS173033

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "**An Effective Undersampling Approach to Deal with Class Imbalance Problem in Software Defect Prediction**" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

(Syed Fawad-ul-Hassan Gillani)
Registration No: MCS173033

Acknowledgements

First, I thank Allah for blessing me this wonderful opportunity to pursue my MS thesis.

I would like to thank my supervisor Dr. Aamer Nadeem, for all his encouragement and support throughout this journey. Once again, I would like to thank my supervisor, Dr. Aamer Nadeem, whose patience and relentless commitment have rendered me a much stronger researcher. I am particularly grateful to the members of my CSD committee for their help and valuable advice throughout this journey

I cannot thank you sufficiently, Sir Muhammad Rizwan, for all that my have accomplished. You were a promoter, a friend and, most importantly, a great person, I am grateful to you forever.

To my mother, father and brothers thank you very much for your help on this journey. You all taught me what a real sacrifice is. Without your sincere generosity, your devotion, help, inspiration and advice I would not have been here. I feel so fortunate to have such a wonderful family. Thank you to my father, brother, sisters and my wife for there every step of my journey. That thesis I dedicate all of you.

(Syed Fawad-ul-Hassan Gillani)

Abstract

Software testing is the process of finding faults in software by executing it. Many software development tasks are carried out by humans, which can lead to numerous software bugs arising over the course of development. The results of the testing are used to find and correct faults. Software defect prediction estimates where faults are likely to occur in source code. Therefore, in the initial stages of testing, the prediction of software defects has become a primary concern in software engineering. The results from the defect prediction can be used to optimize testing and ultimately improve software quality. The most important ability of the software testing process is to identify software defects and the most important thing is reducing software cost and enhancing the overall reliability and quality of the required software. Machine learning is used for software defect prediction to identified the defect modules. Machine learning, which concerns computer programs learning from data, is used to build prediction models which then can be used to classify data. In Machine Learning class imbalance problem is an important issue. Class imbalance problem has become an important issue and many authors research on software defects prediction to solve the class imbalance problem. In the class imbalance problem most instances belong to one class, this class is known as the majority class and the label of the majority class is negative. And the other class has very few instances and this class is known as minority class and the label of minority class is positive.

In the literature, different techniques have been proposed for class imbalance problem. These techniques have been divided into the data level, algorithm level, and the combination of both data and algorithm level techniques. Random over-sampling and random under-sampling are the basic sampling strategics in the data-level approaches for class imbalance problem. Under-sampling reduces the dataset and deletes the instances from the majority class, while over-sampling expands the dataset and adds the duplicates and synthetic instances of minority class in the dataset. Both sampling approaches are used for balancing the dataset, under-sampling removes the instances from the training dataset and may also remove useful information the model has to learn from. Oversampling will cause

an increased training dataset, due to this increased size of the training dataset over-fitting may occur.

There are two main objectives for this thesis. First to achieve an effective under-sampling with minimum loss of useful information. Second to address the relationship between Structured Under-Sampling and Imbalance Ratio (IR).

In this thesis, we propose a new technique known as Structured Under-sampling (SUS). With the help of SUS, we try to address the problem of loss of information due to under-sampling. The problem with Random Under-Sampling is that it removes the instances randomly and due to randomness, loss of useful information may occur. Many different techniques try to solve the loss of information problem. In under-sampling information is lost, but in the proposed under-sampling technique we systematically remove instances and we delete inconsistent/noisy instances, repeated / redundant instances, and most similar instances are removed with minimum loss of information.

We used the C4.5 classifier and compared the performance of Structured Under-sampling with other Under-sampling techniques and measure the performance difference between sampling methods using F-Measure and ROC. We compared the performance of the proposed method Structured Under-sampling with Tomek Links, Random under-sampling, and Tomke Random under-sampling. The results of our investigation have shown good performance using Structured Under-sampling as compared to other existing under-sampling techniques. Software defect datasets showed the superiority of method Structured Under-sampling after using Machine learning algorithms C4.5. Some datasets have shown comparable performance using all under-sampling methods. Imbalance ratio affects the performance of Structured Under-Sampling, when the Imbalance ratio is > 5 SUS outperformed in all results. In future work, our effort is to improve the SUS performance and find the best combination of SUS with any other oversampling approach, and improve software defect prediction performance for SDP.

Contents

List of Figures

List of Tables

Abbreviations

CI	**C**lass **I**mbalance
CIP	**C**lass **I**mbalancen **P**roblem
IDS	**I**mbalanced **D**ata **S**ets
IR	**I**mbalance **R**atio
MiC	**M**inority **C**lass
MjC	**M**ajority **C**lass
ML	**M**achine **L**earning
ROS	**R**andom **O**ver-**S**ampling
RUS	**R**andom **U**nder-**S**ampling
SDP	**S**oftware **D**efect **p**rediction

Chapter 1

Introduction

Software demand has grown significantly in recent past[1], however software systems do not behave as expected in many scenarios. Software testing is very important for improving the reliability and quality of software. Improving the reliability and quality of software is a challenging task for software engineers. The goals of the software testing process is to identify software defects and enhancing the overall reliability and quality of the software [2]. However, as far as resources are concerned, software testing is a time consuming and expensive task. Software Defect Prediction (SDP) helps in testing process and guides the testing team to concentrate more efforts on the fault-prone modules.

The method of predicting modules that may be fault-prone in software is called SDP. In previous research, several machine learning algorithms have been used (also known as classifiers) for SDP [3, 4]. Decision trees, classification rules, neural networks, and probabilistic classifiers are members of these families [3].

1.1 Software Defect Prediction

In the SDP process, machine learning models learn via previous project datasets and predict the defects in the required software system [5]. Datasets have been created from software repositories including defect monitoring systems, source

code changes, data extraction, and version control systems. Datasets consist of samples that may be modules of applications, directories, classes, functions, and modules. When datasets are created for SDP, before SDP using preprocessing methods such as noise detection and reduction [6], data normalization [7] the collected datasets are cleaned. In SDP after pre-processed datasets are used to create a machine learning model that predicts whether or not new instances contain defects. Software metrics such as code metrics and process metrics are used in SDP. In SDP software metrics have a very important role and these metrics are created with software systems [8].

Software metrics can be defined as a quantitative calculation that assigns the characteristics of expected instances to symbols or numbers [8]. These characteristics or attributes characterize several features, such as software products' reliability, effort, complexity, and consistency. In creating an efficient software defect indicator, these metrics play a key role. They can be divided into two major categories: code metrics and process metrics [9].

Code metrics: are explicitly obtained from the source code. These metrics measure the complexity of source code based on the assumption that complex software components are more likely to contain bugs. Throughout the history of software engineering, various code metrics have been used for software defect prediction [3, 10–12].

Process metrics: are metrics based on past changes over time in the source code. These metrics can also be derived from the Source Code Management System and include, for example, the number of code changes and removals, the number of different committees, and the number of lines that have been changed. Numerous prediction studies subsequently applied this metric for SDP [3, 13–15].

In the literature, several studies have indicated that software defect predictors work better when making use of machine learning models to learn from past datasets [3]. Machine learning is a mathematical study that explores the creation and analysis of methods that allow computer programs to learn from data without being

specifically programmed [5].

Usually, machine learning algorithms are classified as : supervised and unsupervised learning [16]. Methods used in unsupervised learning learn predictors from unlabeled data [3]. Besides that, supervised learning, consist of a collection of data input with label information it learns prediction from labeled data. The outputs can be real numbers in regression in supervised learning or class labels in classification.

Supervised learning is also known as classification, as it classifies input into two or more classes. The labeling approach has been commonly used in the literature to classify instances of software modules as defect or defect-free. In table 1.1 we mention some different machine learning algorithms that are use for SDP.

TABLE 1.1: Summary of Classifiers

Abbr	**Classification Algorithm**	**Ref**
LR	Logistic Regression	[17, 18]
NB	Naive Bayes	[19]
C4.5	Decision tree	[13]
IBk	Instance based kNN	[20]
Ripper	Rule based Ripper	[21]
SVM	Support vector machine (SMO)	[22]
RF	Random Forest	[1, 23]

Measuring classifier performance is completed with four measures of classification correctness. These four measures are True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). These four measures represent the confusion matrix as we mention in table 1.2.

These four measures within the confusion matrix define a facet of the correctness of classification results [3]. Different performance measures are uses to evaluate the machine learning model performance, in the table 1.3 we mention some performance measures.

TABLE 1.2: Confusion Matrix

	Actual	
	Defective	**Non-Defective**
Predicted defective	TP	FN
Predicted non-defective	FN	TN

TABLE 1.3: Summary of Performance Measures

Name	**Ref**
Accuracy	[24, 25]
False Positive rate	[26]
Recall	[26]
Precision	[12, 24]
Balance	[27]
ROC	[24–26]
F-Measure	[6, 12, 24, 28]

Due to the imbalance problem, the output of the machine learning model may be biased. The imbalance problem is often considered to be a class imbalance problem. The machine learning model focuses on one class during classification and ignores the other class because of the problem of class imbalance [3].

1.2 Class Imbalance Problem

Class Imbalance problem (CIP) refers to imbalanced dataset (IDS). The dataset is considered as imbalanced when there are too few instances of one class this class is known as minority class (MiC) and the label of MiC has positive. The other class has many more instances this class is known as Majority Class (MjC) and the label of MjC is negative. Many real-world datasets face the CIP, and due to the occurrence of CIP, the output of the classifier is biased. CIP has been existing

in many real-world datasets such as text analysis, detection of fraud, identification of oil leaks, medical diagnosis, credit card account services, and software defect prediction, etc. [3, 29].

For example, during the classification when one class (MiC) has few instances and the other class has much more instances (MjC) the machine learning model ignores the MiC and focuses on MjC. When the classifier can not identify the occurrence of unique instances (MiC) due to this result can be more serious consequences in real-world datasets IDS as we mention above.

With the SDP scenario, misinterpretation of faulty and non-faulty instances leads to very serious software risks. In CIP, Imbalance Ratio (IR) is generally used as a measure to compute data imbalance.
As shown in Eq. 1.1, IR is a well-known term in IDS. In software defect prediction, IR is defined as the ratio of non-defective (MjC) samples in the number of defective (MiC) [3]. In Eq. 1.1, IR represents the Imbalance Ratio, MjC and MiC represent majority, and minority class instances respectively.

$$IR = \frac{MjC}{MiC} \tag{1.1}$$

In the literature, many sampling techniques have been proposed to deal with CIP.

These approaches are divided into two as groups data level and the algorithm level approaches [3].

1.3 Sampling Approaches for CIP

Random over-sampling (ROS) and Random under-sampling (RUS) are the basic data level sampling technique. RUS reduces the dataset and randomly deletes the instances of MjC while ROS increases the dataset and randomly adds the duplicate instances of MiC. Under-sampling discards data from the dataset and in reduced datasets useful information can be lost. Over-sampling increases the dataset and increased training dataset when used as training set may cause over-fitting.

1.3.1 Under-Sampling Approach

Under-sampling (US) is a simple data-level approach for sampling training dataset [30]. Under-sampling eliminates the MjC instances from the training dataset until the IR reaches specified level between the MiC and MjC. According to the literature, the issues with RUS it cannot manage which data sample is removed from the MjC. Particularly, very important data sample about the decision boundary between the MiC and MjC may be eliminated and useful information may be lost. In literature, different under-sampling techniques have been proposed such as Condensed Nearest Neighbor (CNN) [31], One-Sided selection (OSS)[32], DBSCAN Under-sampling [33], NearMiss Under-sampling [34].

1.3.2 Over-Sampling Approach

Over-sampling (OS) is also a data-level approach in this method the size of MiC is increased with random selection or with adaptive approach [30]. This method has several advantages: low computational complexity, no distortion to MiC distributions, and natural generalization to the multi-class case. However, the method has a drawback, it gets many instances with the same points, and also causes over-fitting. In literature, different over-sampling approaches have been proposed such as SMOTE [35], MSMOTE [36], Borderline-SMOTE [37], ADASYN [30].

1.4 Problem Statement

For the challenge of class imbalance problem, researchers have applied different data preprocessing methods. The under-sampling method is one of them to over the class imbalance problem.

In chapter 2, we discuss different under-sampling techniques that are used to balance datasets. Most of the techniques try to manage the boundary between MjC and MiC by expanding boundaries to the majority and giving more room to the minority. The main challenging task while performing under-sampling is

to select the decision boundary between MiC and MjC because it risks losing sample points that contain valuable information for the classification. During under-sampling process useful information may be lost. To overcome challenge, we propose a new under-sampling technique as discussed in chapter 3.

1.5 Objective and Research Questions

There are two main objectives for this thesis. First to achieve an effective under-sampling with minimum loss of useful information. Second to address the relationship between SUS and Imbalance Ratio (IR), and find out which IR impacts the performance of SUS.

RQ1: How can we achieve an effective under-sampling with minimum loss of information as compared to the other existing under-sampling approaches?

RQ2: Does the proposed under-sampling approach improve the performance over existing under-sampling approaches?

RQ3: How does imbalance ratio affect the performance of proposed under-sampling approaches?

1.6 Outline of Thesis

The rest of this thesis is structured as follows.

Chapter2

This chapter presents related work about under-sampling techniques. We discuss and explain existing the under-sampling approaches.

Chapter 3

This chapter presents the proposed approach. It includes a diagram, and explains the three different phases of the proposed approach.

Chapter 4

This chapter describes experiments followed by analysis, evaluation and the design of evaluation framework, and also discusses the tool that is used in experiments.

Chapter 5

This chapter contains a description of results, analysis and also presents performance difference between proposed sampling method SUS and other sampling methods.

Chapter 6

This chapter discusses conclusion and future work and answers the research questions, as well as provides a possible direction for future research.

Chapter 2

Literature Review

In the literature, many solutions have been proposed for CIP. Proposed solutions are divided into two groups data and algorithm level solutions. Hybrid solutions is the combination of data and algorithm level solutions. In section 1.3 we have mention different Under-Sesampling (US) and Over-Sampling (OS) methods. In section 2.1, we have discussed only under-sampling methods.

As we mentioned in section 1.3 data-level solutions are divided into two groups US and OS. Under-sampling is an important resampling method [30]. In the literature several under-resampling methods are applied to balance class distribution. Although US and OS have shown good performance, OS still faces the over-fitting problem and the US still faces the problem of information loss. According to the literature, one of the main issues with the US is that we cannot manage which data simples are removed from the training dataset due to this reason, useful information can be lost [16]. Section 2.1, discusses the under-sampling approaches and also discusses some sampling approaches used to reduce the noise or outliers.

2.1 Under-Sampling Technique

Laurikkala proposed a new method Neighborhood Cleaning rule (NCL) in 2001 [38]. The Neighborhood Cleaning Rule algorithm was created for the binary case,

it cleans border and noise by deleting wrong points among 3-nearest neighbors. This algorithm seems simple but performs very well. The proposed Neighborhood Cleaning Rule is outperforms RUS and OOS methods in experiments with ten data sets. All reduction methods improve the identification of small classes (20-30 percent), but the differences were insignificant. Machine learning model C4.5 results suggest that Neighborhood Cleaning Rule is a useful method for improving the modeling of difficult small classes, and for building classifiers to identify small classes from the real-world data.

Mani and Zhang proposed a new method NearMiss Under-sampling in 2003 [34]. NearMiss Under-sampling is a binary under-sampling algorithm that uses average distances between a given point and also the nearest or farthest points of an opposite class. There are four variants of the NearMiss approach.
In NearMiss-1, need to pick the MjC points up to a given percentage of the MjC size, which is close to some of the MiC points. NearMiss-1 suggests picking out MjC points with the smallest average distance to the three nearest points from the MiC. In NearMiss-2, need to pick the MjC points up to a given percentage of the MjC size, which is close to all points of the MiC. NearMiss-2 means to select the MjC points with the smallest average distance to the three farthest points from the MiC. In NearMiss-3 for each minor class point, and pick a given number of the nearest MjC points.In most distant negative points (NearMiss-4) need to pick the major class points (up to a given percentage of total) with the largest average distance to the three closest points of the minor class.

Liu et al. proposed two algorithms to solve majority-class issue in 2008 [39]. This paper proposes two methods, one straightforward method is to sample several subsets independently from N (the majority class), use these subsets to train classifiers separately, and combine the trained classifiers. Another method is to use trained classifiers to guide the sampling process for subsequent classes.The experiment result shows the good performance with such datasets car, ionosphere, letter, phoneme, sat, and wdbc, Ada achieves very high AUC values, which are all greater than 0.95.

Yen and Lee proposed Cluster-based under-sampling in 2009 [40]. In the proposed approach, they first divided all the training dataset samples into several clusters. The basic principle is various clusters have many samples including MjC and MiC samples, and each cluster tends to have distinct characteristics. When a cluster contains more groups of MjC and fewer MiC Samples, they should act like samples of the MjC. On the other hand, if a cluster contains more members of MiC, it does not have the property of samples of the MjC but performs more like MiC samples. Cluster-based under-sampling picks an appropriate number of majority level samples of each cluster by a ratio from MjC groups to minority percentages in-cluster class samples. Cluster-based under-sampling has good results on two real applications that include the issue of imbalanced class distribution and it requires less time than other methods for choosing the training samples.

Tahir et al. proposed inverse random under-sampling (IRUS) in 2012 [41]. In this paper, a novel IRUS approach is introduced for CIP. In IRUS the cardinalities of the MjC and MiC are reversed. The new methodology is utilized in 22 UCI data sets. Experimental findings indicate a substantial performance improvement relative to other current methods of teaching and learning with an imbalanced class. This paper presents results for multi-label classification. Classification of multi-label IDS is a challenging research issue in many modern applications including SDP. This paper claims that this limit has the potential to delineate the MjC more efficiently than traditional learning solutions. T-tests (level of significance 0.05) were performed using AUC, F1, and G-mean showing WIN – TIE – LOSE, respectively. Together with other approaches like the state-of-the-art EasyEnsemble, the paired t-test shows that IRUS is superior in most data sets.

Ng et al. proposed Diversified Sensitivity-Based Under-Sampling (DSUS) method in 2014 [42]. DSUS selects useful samples and avoids selecting samples around the decision boundary. The DSUS outperformed other methods with a statistical significance. The DSUS provided the best performance in terms of all AUC, FM, and G-Mean. In these datasets Pima, breast, post-op, CMC, newthyroid, and optdigits, the DSUS does the best for the FM, except the Haberman dataset.

Beckmann et al. proposed KNN under-sampling (KNN-Und) algorithm in 2015 [43]. This work is focused on the data adjusting algorithms and a proposal of a KNN under-sampling (KNN-Und) algorithm. The KNN- Und is a very simple algorithm, and basically, it uses the neighbor count to remove instances from the MjC. The KNN-Und algorithm was developed as a preprocessing plugin in the WEKA platform. It can also be used to solve the CIP, commonly associated with IDS. In highly skewed IDS, the algorithm can be used with some other sampling method to improve the results. In this paper, AUC was used as a performance measure and C4.5 as machine learning algorithm. The experiments were performed on 19 datasets. The KNN-Und outperformed in 11 of 15 datasets.

Liu et al. proposed a novel threshold-based clustering algorithm (NTC) as a two-stage data preprocessing approach in 2015 [44]. The main contribution of the paper is a novel two-stage data preprocessing approach which performs both feature selection and instance reduction in sequence. This paper used three classification models which are commonly used in SDP and AUC was used as a performance measure. This paper also uses the Friedman test to determine whether the performance measures used in the experiments are sufficient or not. This paper uses random under-sampling in the sample reduction stage to achieve the balance between faulty and non-faulty instances. According to this paper experiments, datasets were selected from software projects of Eclipse and NASA. The proposed approach was compared with other approaches to certain classical baseline approaches and further researched the contributing variables in the approach. The final results indicate the proposed method's efficacy and include a guide for achieving cost-effective preprocessing of data by using a two-stage approach.

Yang et al. proposed Under-Sampling Conditional Field (UCRF) in 2015 [45]. In this paper, studies have been conducted on under-sampling, and CRF Model. UCRF has two stages, in the first stage it handles CIP that is the key reason accounting for the poor performance of certain machine learning methods.
In this paper authors propose to balance the training dataset by mean-shift clustering approach. The mean-shift clustering approach removes the MjC samples to achieve the required IR in the training dataset. In the second stage, UCRF

adopts the CRF model which has the ability to handle complex features without any change in training procedure in the above-balanced data set. The CRF model can be easily applied to recognition because it is a modeling technique for state and mark sequences. According to the results of this paper, UCRF outperforms all other approaches.

Chen et al. proposed a noise-filtered under-sampling system, called the EE-IPF in 2016 [46]. This article discusses a noise filtering strategy for imbalanced classification in the sense of an under-sampling algorithm. The author borrows the active Partitioning Filter to boost EasyEnsemble(EE) efficiency. To reduce the risk of failure reducing a minority class instance, authors suggest upon under-sampling takes multiple IPF filters.

Kang et al. proposed Noise-filter Under-Sampling Scheme (NUS) in 2016 [7]. This paper combines noise filter with methods of under-sampling and discusses multiple experiments to check the NUS approach by using ML Repository comparison data sets from the University of California Irvine (UCI). This paper has utilized well-known strategies of RUS, US, RUS Boost (RUSB), Under-sampling + Adaboost (UA), UnderBagging (UB). The proposed context in which a K-nearest cousin (KNN)-dependent noise filter, named KF for short, is applied, and the efficiency metric used to test ML algorithm output with AUC, F-measure, and G-mean is focused on that.

Elhassan and Aljurf proposed Tomek-Link combined with Random Under-Sampling (TL-RUS) in 2016 [47]. Tomek Link algorithm was used in the preprocessing phase as a method of data cleaning to remove noise. After filtering the IDS it uses other under-sampling methods on the filtered dataset. In this paper, Tomek Link is combined with other sampling methods such as RUS, ROS, and SMOTE to maintain a balanced class distribution in IDS. The classification uses several ML algorithms such as ANN, RF, and LR. After reducing noise from IDS, it applies different resampling methods including RUS, ROS, and SMOTE. SMOTE and RUS showed good performance among various resampling methods. Under-sampling techniques showed superior efficiency compared with other sampling techniques for all classifiers tested.

Sowah et al. proposed Cluster Under-Sampling Technique (CUST) in 2016 [48]. This paper presents a new under-sampling technique referred to as CUST, that has the capability of raising the performance of classification algorithms by learning from imbalanced datasets. The performance of CUST was evaluated by using sixteen class imbalanced datasets before building classification models using the C4.5 decision tree algorithm. In CUST experiment C4.5 classification algorithm was used and two performance measures AUC and G-mean were used. The experimental results showed that CUST improved the performance result as compared to SMOTE, RUS, ROS, OSS, and NONE.

2.2 Distance Function Based Data Reduction Method

Condensed Nearest Neighbor (CNN) Rule is formed for the binary case [31]. Beginning with original dataset S, tend to produce associate initial under-sampled data set C with only the MiC and one random purpose from the MjC. Then classify S by 1-nearest neighbor classifier using the points in C and move all misclassified examples of S into C. The complexity of this algorithm is O(m.n) where m is the size of the MiC and n is the size of the MjC. This algorithm has a drawback created by the random initialization, where generally a selected point appears on the border of the major class, and deleted points unexpectedly distort the data distribution.

DBSCAN Under-sampling is a clustering-based under-sampling algorithm [33]. DBSCAN clustering is fast as O(n) and it detects outliers that can be used for the border and noise-cleaning. The radius of the neighborhood parameter can be found as a quintile of within-class distances. This needs to compute and sort distances of the class points which is costly, so for the big size major classes, it is possible to sample a small portion of instances for this task.

One-Sided selection (OSS) under-sampling is a combination of Condensed Nearest Neighbor Rule in the first stage and Tomek links in the second stage [32]. Tomek

links are used as a resampling tool to exclude instances of majority class noise to borderline. Borderline instances may be deemed 'unsafe', because a slight amount of noise may cause them to fall on the wrong side of the decision boundary. The US-CNN seeks to exclude from the plurality class cases which are far from the boundaries of the decision. The remaining instances, i.e. "free" instances of a majority class, and all examples of minority groups are used for learning.

In literature, many different approaches have been proposed that use distance function to compute the similarity between the instances, such as Tomek links [49], Condensed Nearest Neighbor Rule [31], and One-Sided Under-Sampling [32], Tomek Random Under-sampling [47].
Data cleaning methods are used for Under-sampling. The main goal of these methods is to identify possible noisy examples or overlapping regions and then decide on the removal of examples. One of those methods uses Tomek links [49], which consists of points that are each other's closest neighbors but do not share the same class label. This method allows for two options: only remove Tomek links examples belonging to the MjC or eliminate Tomek links examples of both classes.

The notion of the Condensed NearestNeighbour Rule (CNN) was also applied to perform undersampling [40]. CNN is used to find a subset of examples consistent with the training set, that is, a subset that correctly classifies the training examples using a 1-nearest-neighbor. The CNN and Tomek links methods were combined in a strategy called One-SidedSelection (OSS).

In literature, distance function-based approaches have been presented as under-sampling methods as we mentioned in the above section. But according to Lemaitre et al. [50], all approaches used for data cleaning methods do not achieve specific IR. According to the literature, the Euclidean distance function is frequently used for data cleaning.

In table 2.1 we have presented the summary of sampling approaches. According to our best knowledge, these techniques show good performance in a specified scenario as mentioned in table 2.1. In past, some researchers used distance function as a

data reduction method, data filter method, and noise filter. In the next section 2.2, we discuss existing approaches that have used distance function for under-sampling.

2.3 Euclidean Distance Function

Danielsson has proposed Euclidean Distance Function (EDF) [51]. The EDF is one of the most commonly known distance function and its has used in previous methods. The Euclidean distance function has been used in the previous under-sampling techniques as a distance measure.
The Euclidean distance formula is used to calculate the distance between two data points as shown in Eq. 2.1.

$$d\left(x,y\right)=\sqrt{\sum_{i=1}^{n}\left(xi-yi\right)^{2}} \tag{2.1}$$

2.4 Summary

CIP affect the ML algorithms performance, in addition to the CIP, there are other well-known problems of data quality such as inconsistent/noisy instances, repeated / redundant instances, and outliers that can adversely impact the performance of classification algorithms where large numbers are present in the training results [52].

Many authors have been proposed many solutions to deal with the CIP for SDP. In table 2.1, we summarize the existing sampling approaches. According to table 2.1, many techniques use clustering to address the CIP in IDS [40, 42, 44, 48].The central issue in clustering techniques is that these are dependent on the number of clusters and the clustering method is dependent on the optimal number of clusters and the quality of clusters.

Table 2.1: Summary of Under-Sampling Techniques

Year/ article	Proposed Technique	Comparison with	Used Classifier	Datasets used	Datasets Name	Performance Measures	Results
2016 [48]	(CUST)	RUS, SMOTE, OSS, and Cluster-Based Under-sampling.	C4.5	16	CM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2,	ROC, G-mean	HSD test on the mean performance of the classification algorithms at Alpha=0.05 showed that CUST statistically performed better.
2016 [7]	Noise-filtered Under-sampling Scheme (NUS),	(UA, RUSB, UB, or EE)	Ada-Boost	16	Pima, poker, wine-red, balance, wilt, page-block	AUC, F-measure and G-mean	X-KFs shows that the EE-F works the highest
2016 [46]	EE-IPF	SMOTE-IPF, SMOTE, EE	Ada-Boost	11	Cmc, wdbc, wpbc, latter, housing, pima, balance	AUC, F-measure, and G-mean	The corresponding EE-IPF outperforms the EE and SMOTE-IPF.
2016 [47]	T-RUS	RUS, ROS, SMOTE, TLink/RUS, TLink/ROS and TLink/SMOTE	SVM, ANN, RF, and LR.	223	The data set contains 223 families with a mean number of siblings equal to 3 siblings per family.	F-statistic, G-mean, sensitivity and weighted accuracy	Performance measures such as F-statistic,
2015 [45]	UCRF	RCRF	SVM and neural network	11	CM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2,	PD, PN, G-mean	UCRF is almost 4% better than that of RCRF.
2015 [44]	NTC	ROS, RUS	NB, C4.5, and IB1	13	CM1, KC1, KC3, MC2, MW1, PC1, PC2	AUC	RUS performs better than other instance sampling techniques, such as ROS for software fault prediction. The proposed algorithm NTC can be further improved software fault prediction.
2015 [43]	KNN-Und	RUS, SMOTE, ENN, ECL	C4.5	33	GlassBWNFP,EcoliCP-IM, Pima, Habermann, New-Thyroid	AUC and G-Mean.	KNN-Und outperformed in 11 of 15 datasets as compared with other approaches.
2014 [42]	DSUS	RUS	SVM	14	pima, breast, post op, cmc, newthyroid, and optdigits	AUC, F1, and G-Mean.	In 153 out of 252 (60.71%) experiments, the DSUS outperforms other methods with a statistical significance
2012 [41]	IRUS	RUS, ROS, SMOTE	C4.5	22	WPC,Pima, Breast-cancer, Glass, Hepatitis	F1, G-mean, AUC	T-test tests (level of significance 0.05) use AUC, F1, and G-mean showing WIN – TIE – LOSE, respectively. Together with other approaches like state-of-the-art EasyEnsemble, the paired t-test shows that IRUS is superior in most data sets
2009 [40]	Cluster-based under-sampling	NearMiss-2, SBCNM-1, SBCNM-2, SBCNM-3, SBCMD, and SBCMF	artificial neural network, k-means clustering algorithm	4	DS4E10DN, DS4E10D20, DSiEjDk, DSiEjDN	precision , recall and F-measure	Classification on the DS4E10DN dataset shows the best result when IR is 1:1, and the F-measures are above 80%.
2008 [39]	Balance Cascade ,	With 15 techniques	Ada-Boost	16	Car, cmc, wpbc, wdbc, pima	ROC ,F-measure and G-mean	Car, ionosphere, letter, phoneme, sat, and wdbc, Ada achieves very high AUC values, which are all greater than 0.95.

In IDS, MiC instances may overlap the MjC instances and these data samples may affect the classification performance. According to Shepperd et al., inconsistent instances have an impact on classification performance [52]. In section, 2.1 discuss under-sampling methods that minimize noise or inconsistency [7, 46, 47]. In CIP when noisy data are used for SDP, it has a bad affect on classifier performance. Typically the noisy data is correlated with outliers. Outliers may arise as a result of measuring differences in datasets, or perhaps as a consequence of experimental error. Noise filtering, as reported by [52], has also been reported as an efficient way of removing noise in a dataset.

In this thesis, we propose a new under-sampling technique called Structured Under-sampling (SUS). Our proposed approach is based on the EDF. The key difference between SUS and other under-sampling methods [31, 32, 34, 47, 49] is the proposed SUS removes 3 different kinds of instances from MjC. Our proposed SUS is not dependent on any clustering algorithm. In the next chapter, we discuss the proposed SUS methodology and explain the working of the proposed SUS technique.

Chapter 3

Proposed Approach

In this chapter, we have presented our proposed methodology known as Structured Under-sampling (SUS). In this chapter, we explain the working of the proposed technique. We also explain the flowchart of the proposed methodology.

Structured Under-sampling identifies inconsistent samples between MiC and MjC in IDS. After identification of inconsistent samples, we remove these samples from MjC. Since SUS is an under-sampling technique, we remove only MjC instances. According to Shepperd et al, inconsistent instances impact classification performance [52]. In the proposed approach we try to minimize the problem of information loss during under-sampling and also handle the problem of inconsistent instances.

The SDP research community is continuously proposing and using an under-sampling technique to address CIP. However, existing approaches are suffered from either removing more informative instances like a diverse instance or keeping less informative instances like inconsistent and duplicate instances.
This reflects the sub-optimum performance of ML models. Therefore, there is a need to propose an under-sampling technique that balances the imbalance dataset by identifying and dropping the least informative instances.

In this chapter, we explain the proposed approach into three different phases as we shows in figure 3.1, 3.2 and 3.3.

3.1 Structured Under-sampling (SUS)

In the first phase, identify the inconsistent samples. Two samples are inconsistent if they are identical but have different labels. These kind of instances affect the classification process. So in SUS, first we handle inconsistent instances problem and remove all inconsistent instances from MjC.

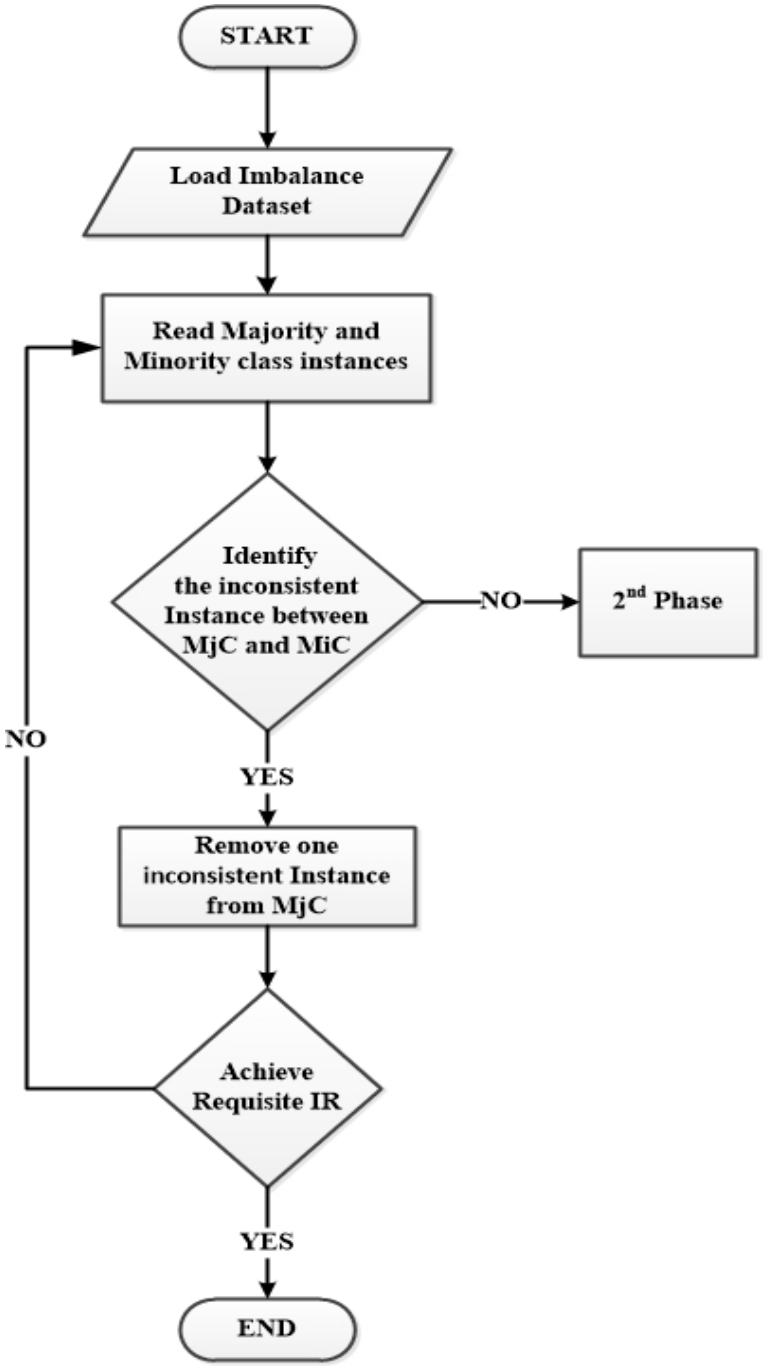

FIGURE 3.1: SUS 1^{st} Phase Flowchart

In the second phase, we identify the duplicate instances in IDS. Duplicate instances are two different samples having same information with same label, these samples also affect the classification performance. So in SUS, the second phase removes all duplicate instances from MjC.

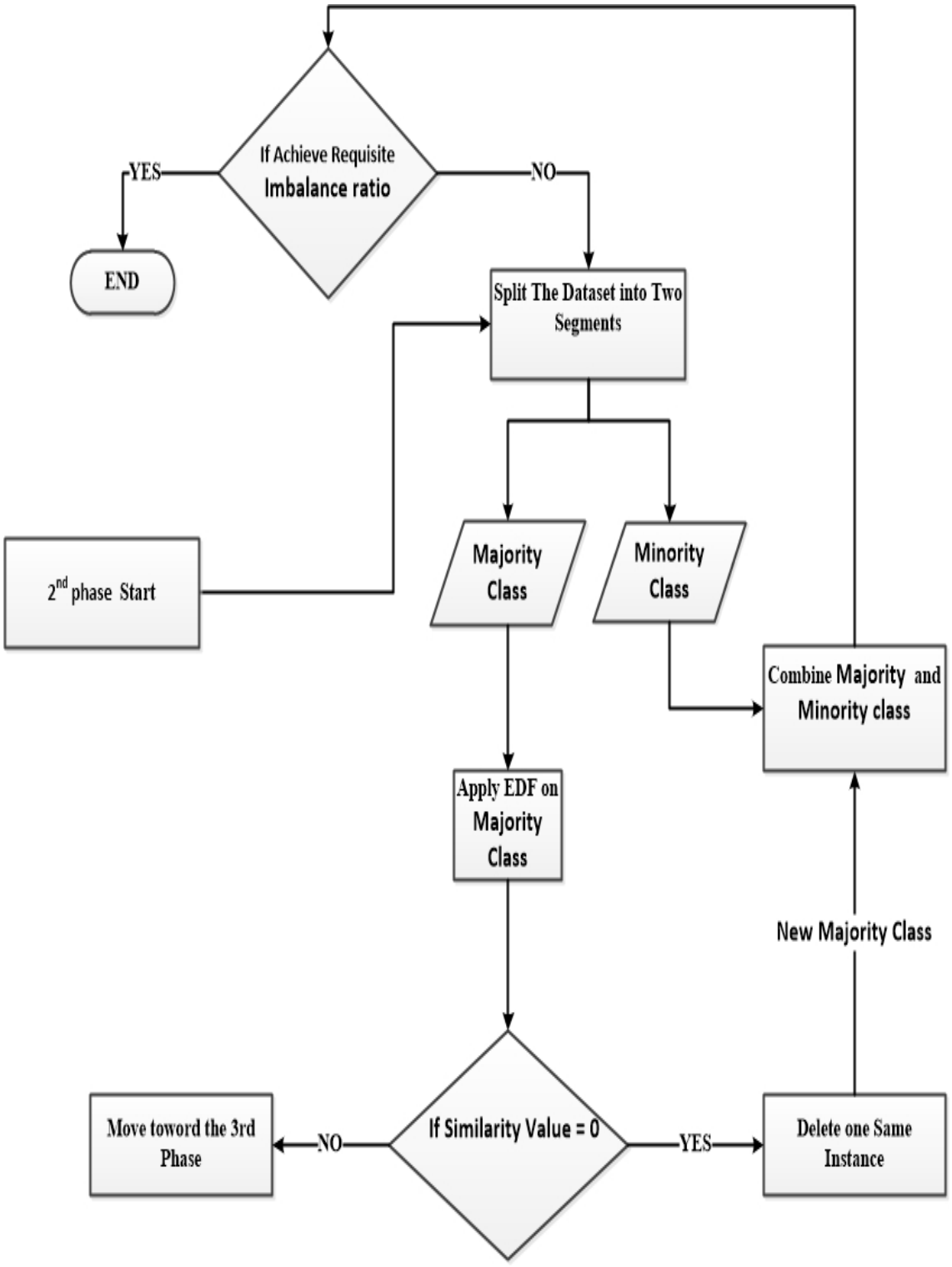

FIGURE 3.2: SUS 2^{nd} Phase Flowchart

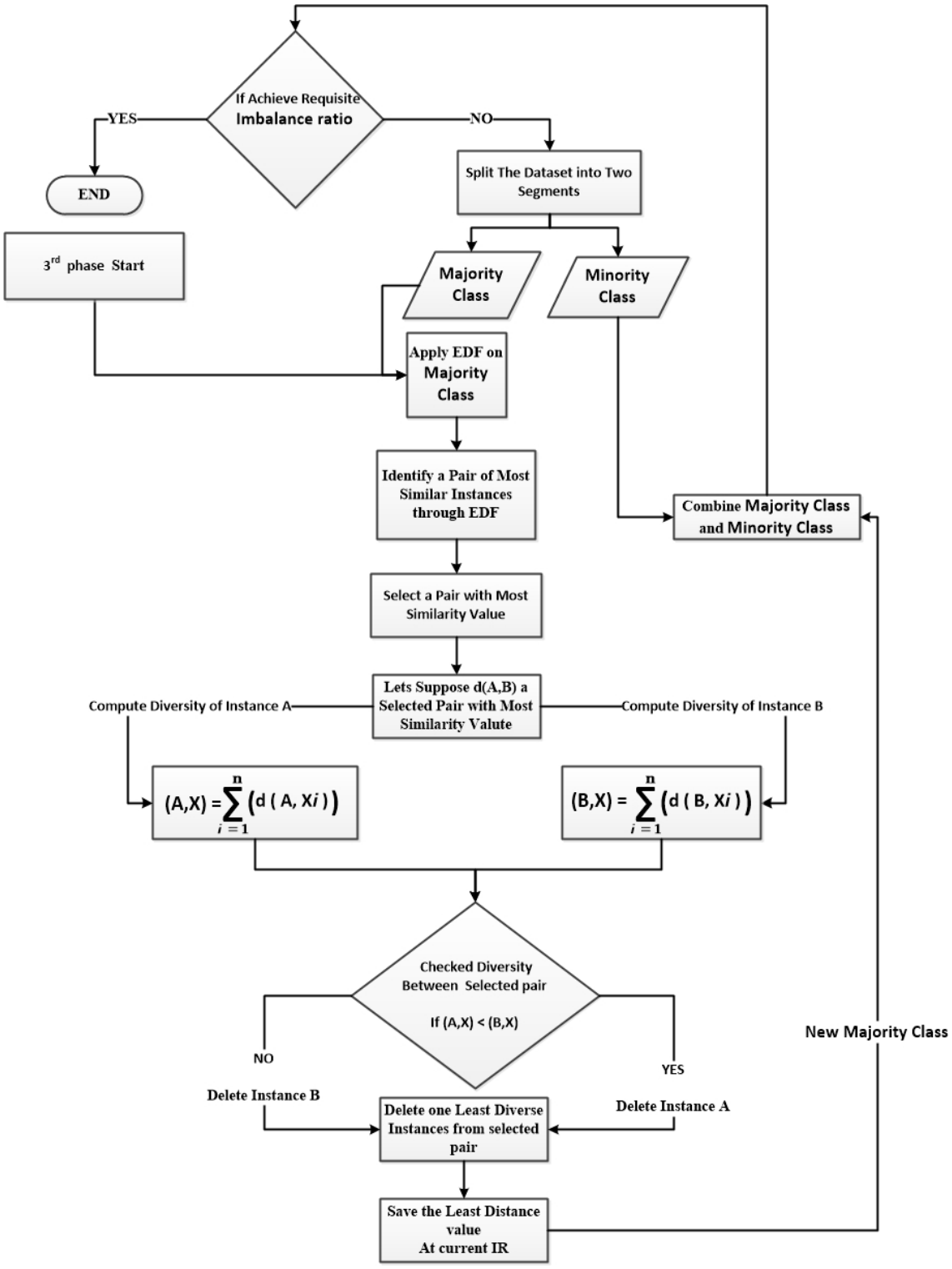

FIGURE 3.3: SUS 3^{rd} Phase Flowchart

In the third phase, SUS removes least diverse instances from MjC based on the EDF. With help of EDF, SUS computes the similarity distance between all pairs of MjC instances and selects one pair with least similarity value.

When we selects one pair of two instances with lest similarity on the based of EDF value, then computes the diversity of selected pair with all other instances of MjC. When computes the diversity, then removes the least diverse instance from selected pair.

There is an important difference between the first and second phases. In the first phases, under-sampling is done between the classes but in second phase only MjC and computes the similarity within the MjC instances.

lets suppose **d(A,B)** a selected pair of two instances with lest similarity value. With help of EDF, we computes the similarity of instance **A** with all other instances and add the all pairs similarity values as we shows in Eq. 3.1. In Eq. 3.1, **'A'** show the instance A of selected pair and $'X_i'$ show the all other instances of MjC. We also computes the similarity of instance **B** with all other of MjC and add the all pairs similarity values as we shows in Eq. 3.2. In Eq. 3.2, **'B'** show the instance B of selected pair and $'X_i'$ show the all other instances of MjC.

When we complete the process for selected pair, then we checked the diversity of instance A and B. If (A,X) < (B,X) then removed instance A, If (B,X) < (A,X) then removed instance B.

According to the figure 3.3 when a instance is removes from MjC then we checked the IR, if we not achieved the required IR then SUS again repeat third phase until SUS reached at required IR. In our experiment, SUS stopping criteria is 1:1 IR.

$$(A, X) = \sum_{i=1}^{n} (A - X_i) \tag{3.1}$$

$$(B, X) = \sum_{i=1}^{n} (B - X_i) \tag{3.2}$$

In figure 3.1, we shows the flowchart of the SUS 1^{st} phase. In figure 3.2 we shows the flowchart of the SUS 2^{nd} phase, and in figure 3.3 we shows the flowchart of the SUS 3^{rd} phase. Following are the detail explanation of phases of Structured Under-sampling (SUS).

- 1^{st} **Phase**

  Step 1: An imbalanced dataset is initially loaded.

  Step 2: When IDS loads, SUS read MiC and MjC instances then identify the inconsistent instances from IDS.

  Step 3: If the inconsistent instance is identify, then we removes the instances from MjC. Else, the inconsistent instance is not identified then SUS moves in the 2^{nd} phase as we shows in 3.2.

  Step 4: When we removes one inconsistent instance from MjC instance then cheeked the IR.

  Step 5: If we achieved the required IR, then we stopped the processing of SUS and end the SUS process.

  Step 6: If we not achieve the required IR, then we goes on step 2.

  Step 7: We repeats the step 2 to 6 until we removes all inconsistent instances from MjC.

- 2^{nd} **Phase**

  Step 1; We split the loaded dataset into two segments of MiC and MjC instances.

  Step 2: WE read only the MjC instances.

  Step 3: We apply EDF on MjC, with help of EDF we computes the similarity of all pairs of MjC as shows in figure 3.2.

  Step 4: On the based of EDF value we identify the duplicate instances from MjC.

  Step 5: If the result of decision box **similarity value = 0** is **YES** then moves into the second phase. Else, the result of decision box **similarity value = 0** is **NO** then we goes on 3^{rd} phase.

  Step 6: When decision box **similarity value = 0**, then we selects a pair of two duplicate instances on the based of EDF value.

Step 7: When we identify a pair of duplicate instances from MjC, then we removes one duplicate instance from the selected pair.

Step 8: When we removes one duplicate instance from MjC , then we combine the MjC and MiC insatances .

Step 9: When we combine the MjC and MiC instances, then checked IR.

Step 10: If we achieve the require IR, then we stop the process. Else we move on next step.

Step 11: If we not achieve the required IR, then repeats step 1 to step 10.

- 3^{rd} **Phase**

Step 1: We read the MjC instances.

Step 2: We apply EDF on MjC instances.

Step 3: With help of EDF, we Computes similarity between all pairs of MjC instances.

Step 4: On the based of EDF value, we identify a pair of two instances with least similarity.

Step 5: When we identify a pair with least similarity value, then select this pair and move on next step.

Step 6: On the based of EDF value, when we selects a pair with least similarity then checked the diversity between selected pair.

Step 7: We computes the diversity of selected pair with all other instances of MjC.

Step 8: Let's suppose d(A,B) a selected pair with least similarity value as we shows in figure in 3.3.

Step 9: For Instance A: d(A,X) = $\sum_{i=1}^{n} (A - X_i)$**.**

In step 9, A show the value of instance A of selected pair, and X_i show all other instances of MjC.

Step 10: With the help of EDF, we computes the similarity of instance A with all other instances of MjC as we shows in Eq. 3.1 and figure 3.3.

Step 11: We compute the similarity, then we add the all similarity value of all pairs of MjC as we shows in step 9.

Step 12: For Instance B: d(B,X) = $\sum_{i=1}^{n} (B - X_i)$.

In step 12, B show the value of instance B of selected pair, and X_i show all other instances of MjC.

Step 13: With the help of EDF, we computes the similarity of instance B with all other instances of MjC as we shows in Eq. 3.2 and figure 3.3.

Step 14: We compute the similarity, then we add the all similarity value of all pairs of MjC as we shows in step 12.

Step 15: When we complete the process for selected pair, then we have two different for d(A,X) and d(B,X).

Step 16: On the based of d(A,X) and d(B,X) value, we identify the diver instance.

Step 17: If d(A,X) < d(B,X), then we instance A from selected pair. Else move on next step.

Step 18: If d(B,X) < d(A,X), then we instance B from selected pair.

Step 19: When we removes least diver instance from MjC, the we save the least similarity distance at the current state and move on next step.

Step 20: Now we combine MjC and MiC.

Step 21: checked the IR.

Step 22: If IR = 1:1, we go on step 26. Else we go on next step.

Step 23: If we not achieve the required IR, then move on next step.

Step 24: We split the dataset into two segments MjC and MiC instances.

Step 25: We repeats the step 1 to step 24.

Step 26: End the process.

3.2 Euclidean Distance Function

As we discuss in chapter 2, many previous techniques use EDF for data cleaning and also used for under-sampling. In our proposed approach SUS we use EDF for under-sampling and we removes samples from MjC in systematic way. In Eq. 3.3, we shows how to compute the similarity of MjC instances.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2) + ... + (p_n - q_n)^2} \tag{3.3}$$

According to Eq. 3.3, p and q shows as two instances of dataset and we compute the similarity between these two instances samples. According to Eq. 3.3 p_1 and q_1, we shows the first feature value of instance p_1 and q_1, p_2 and q_2 shows the feature value of second feature, and p_n and q_n shows the value pf last feature. According to the Eq. 2.1 and Eq. 3.3, with help of EDF we compute the similarity between the instances and we performed the under-sampling in systematic way and we try to minimize the problem of loss of information.

3.3 Example

In sample dataset table 3.1, we shows some instances of MjC and MiC. According to sample dataset table 3.1, total number of instances is 11 and MjC have 8 instances and MiC have 3 instances, IR between MjC and MiC is 2.67. With the help of sample dataset table 3.1, we explain the working of SUS and we removes three kind of instances from MjC.

According to SUS working as shown in figure 3.1, first we identify inconsistent instances and we removes them. According to figure 3.2, we identify duplicate instances and we removes them. According to figure 3.3 we identify the lest diverse instances and we removes them. According to the figures 3.1, 3.2 and 3.3, we checked IR, when a single instance remove from MjC. With the help of EDF, we have done under-sampling as we shows in Eq. 3.3 and Eq. 2.1.

TABLE 3.1: Sample Dataset

Instance NO.	Features A	Features B	Features C	Label
1	3	2	1	YES
2	3	3	2	NO
3	3	2	3	NO
4	2	3	4	NO
5	2	1	1	NO
6	2	1	1	NO
7	3	2	3	NO
8	3	2	1	NO
9	3	3	3	YES
10	3	3	2	YES
11	5	3	4	NO

1^{st} Phase

In first phase, we identify the inconsistent instances. According to sample dataset table 3.1, we read both classes MjC and MiC instances.

As we see in sample dataset table 3.1 instance No. **1** and **8** have same information with two different labels, now we removes instance No. **8** from MjC. According to 3.1, when we identify a inconsistent instance then we removes this inconsistent instance from MjC. In 1^{st} phase of SUS, we removes only inconsistent instance.

According to the figure 3.1, when we removes inconsistent instance from MjC, then checked the IR. If SUS not achieved the required IR, so SUS again read the MjC and MiC instances and find an other inconsistent instances. As we can see in sample dataset table 3.2, one instance is removed from sample dataset table 3.1, now SUS again read the MjC and MiC instance and identify the an other inconsistent instance from sample dataset table 3.2.

According to sample dataset table 3.2 instance No. **2** and **9** have same feature value with two different labels. Now we removes instance No. **2** from MjC and

again checked IR as we shows in figure 3.1.

According to sample dataset table 3.3, two inconsistent instance are removes from MjC. When we removes all inconsistent instance from MjC, then we move on 2^{nd} Phase as shows in figure 3.1.

TABLE 3.2: After First Iteration of 1st phase

Instance NO.	**Features A**	**Features B**	**Features C**	**Label**
1	3	2	1	YES
2	3	3	2	NO
3	3	2	3	NO
4	2	3	4	NO
5	2	1	1	NO
6	2	1	1	NO
7	3	2	3	NO
8	3	3	3	YES
9	3	3	2	YES
10	5	3	4	NO

TABLE 3.3: After Second Iteration of 1st phase

Instance NO.	**Features A**	**Features B**	**Features C**	**Label**
1	3	2	1	YES
2	3	2	3	NO
3	2	3	4	NO
4	2	1	1	NO
5	2	1	1	NO
6	3	2	3	NO
7	3	3	3	YES
8	3	3	2	YES
9	5	3	4	NO

2^{nd} Phase

When first phase is completed then SUS goes into the second phase, in second phase, we identify the duplicate instances. We split Sample dataset into two parts MjC and MiC before identify the duplicate instances as we shows in figure 3.2. As we seen in sample datasets table 3.4, samples datasets oder is change its mean we split samples dataset into two parts.

TABLE 3.4: Splited Sample Dataset After 1st phase

Instance NO.	**Features A**	**Features B**	**Features C**	**Label**
1	3	2	1	YES
2	3	3	2	YES
3	3	3	3	YES
-	-	-	-	-
1	3	2	3	NO
2	2	3	4	NO
3	2	1	1	NO
4	2	1	1	NO
5	3	2	3	NO
6	5	3	4	NO

In second phase we identify all duplicate instances from MjC. With the help of EDF we identify duplicate between MjC instances, according to the figure 3.2 when EDF value is "Zero" it's mean two instances have same information. Now we find similarity between MjC instances with help of EDF.

$$d(1,2) = \sqrt{(3-2)^2 + (2-3)^2 + (3-4)^2} = 1.732050 \tag{3.4}$$

$$d(1,3) = \sqrt{(3-2)^2 + (2-1)^2 + (3-1)^2} = 2.449489 \tag{3.5}$$

$$d(1,4) = \sqrt{(3-2)^2 + (2-1)^2 + (3-1)^2} = 2.449489 \tag{3.6}$$

$$d(1,5) = \sqrt{(3-2)^2 + (2-2)^2 + (3-3)^2} = 0 \tag{3.7}$$

According to the Eq. 3.4, 3.5, 3.6 and 3.7, all these pairs shows the similarity value between MjC instance. According to the Eq. 3.7, pair d(1,5) output is zero.

TABLE 3.5: After First Iteration of 2nd phase

Instance NO.	**Features A**	**Features B**	**Features C**	**Label**
1	3	2	1	YES
2	3	3	2	YES
3	3	3	3	YES
-	-	-	-	-
1	2	3	4	NO
2	2	1	1	NO
3	2	1	1	NO
4	3	2	3	NO
5	5	3	4	NO

When duplicate instances is identify in MjC, then we removed this duplicate instance from MjC and checked IR as we shows in figure 3.2. If we not achieved the required IR then again find the an other duplicate instance.

According to sample dataset table 3.5, one duplicate instance is removed from MjC, now SUS Checked IR and SUS again apply the EDF of MjC instance and find an other duplicate instance from MjC as shown in figure 3.2.

$$d(1,2) = \sqrt{(2-2)^2 + (3-1)^2 + (4-1)^2} = 3.605551 \tag{3.8}$$

$$d(1,3) = \sqrt{(2-2)^2 + (3-1)^2 + (4-1)^2} = 3.605551 \tag{3.9}$$

$$d(1,4) = \sqrt{(2-3)^2 + (3-2)^2 + (4-3)^2} = 1.732050 \tag{3.10}$$

$$d(1,5) = \sqrt{(2-5)^2 + (3-3)^2 + (4-4)^2} = 3 \tag{3.11}$$

$$d(2,3) = \sqrt{(2-2)^2 + (1-1)^2 + (1-1)^2} = 0 \tag{3.12}$$

TABLE 3.6: Sample Dataset After 2nd phase

Instance NO.	**Features A**	**Features B**	**Features C**	**Label**
1	3	2	1	YES
2	3	3	2	YES
3	3	3	3	YES
-	-	-	-	-
1	2	3	4	NO
2	2	1	1	NO
3	3	2	3	NO
4	5	3	4	NO

According to the Eq. 3.8, 3.9, 3.10, 3.11 and 3.12, we calculate similarity between MjC instance. According to the Eq. 3.12, pair d(2,3) output is zero. According to the Eq. 3.12 MjC have a duplicate instance, so we removed this duplicate instance from MjC and checked IR as we shows in figure 3.2. If we not achieved the required IR then again find the an other duplicate instance as we shows in figure 3.2. According to the sample dataset table 3.6 two duplicate instances are removed from MjC and SUS not achieved the required IR, then we move on 3^{rd} because no more duplicate instances in dataset as we shows in figure 3.2.

3^{rd} Phase

In third phase, we removes the lest diverse instance from sample dataset table 3.6. With help of EDF, we compute the similarity distance between all MjC instances and selects a pair of two instances with least similarity value. When we selects a

pair with lest similarity, then we decided from selected pair which insensate is lest diver. According to sample dataset table 3.6 MjC class have four instances, now we computes similarity of MjC instances with the help of EDF. According to the Eq. 3.13, 3.14, 3.15, 3.16, 3.17 and 3.18, all these pairs shows the similarity value between the MjC instances.

According to Eq. 3.14, pair d(1,3) have least similarity value as compared to other pairs as we seen in Eq.3.13, 3.15, 3.16, 3.17 and 3.18. Now we computes the diversity of selected pair d(1,3) with remaining two instances 2 and 4.

$$d(1,2) = \sqrt{(2-2)^2 + (3-1)^2 + (4-1)^2} = 3.605551 \tag{3.13}$$

$$d(1,3) = \sqrt{(2-3)^2 + (3-2)^2 + (4-3)^2} = 1.732050 \tag{3.14}$$

$$d(1,4) = \sqrt{(2-5)^2 + (3-3)^2 + (4-4)^2} = 3 \tag{3.15}$$

$$d(2,3) = \sqrt{(2-3)^2 + (1-2)^2 + (1-3)^2} = 2.449489 \tag{3.16}$$

$$d(2,4) = \sqrt{(2-5)^2 + (1-3)^2 + (1-4)^2} = 4.690415 \tag{3.17}$$

$$d(3,4) = \sqrt{(3-5)^2 + (2-3)^2 + (3-4)^2} = 2.449489 \tag{3.18}$$

According to the figure 3.3, when we selects a pair with least similarity value. According to the figure 3.3, we checked the least diver instance from selected pair and we removes the least diver instance from selected pair.

As we seen in Eq. 3.19 and 3.20 we computes the similarity of instance 1 with other two instances 2 and 4. When we computes the similarity value of instance 1

with other two instances then add these similarity values as we shows in Eq. 3.21.

Computes Diversity for Selected Pair d(1,3)

For Instance No. 1

$$d(1,2) = W = \sqrt{(2-2)^2 + (3-1)^2 + (4-1)^2} = 3.605551 \tag{3.19}$$

$$d(1,4) = X = \sqrt{(2-5)^2 + (3-3)^2 + (4-4)^2} = 3 \tag{3.20}$$

$$A = W + X = 3.605551 + 3 = 6.605551 \tag{3.21}$$

According to the Eq. 3.22 and Eq. 3.23 we computes the similarity of instance 3 with other two instances 2 and 4. According to the Eq.3.24 we add the similarity of Eq. 3.22 and 3.23.

For Instance No. 3

$$d(3,2) = Y = \sqrt{(3-2)^2 + (2-1)^2 + (3-1)^2} = 2.449489 \tag{3.22}$$

$$d(3,4) = Z = \sqrt{(3-5)^2 + (2-3)^2 + (3-4)^2} = 2.449489 \tag{3.23}$$

$$B = Y + Z = 2.449489 + 2.449489 = 4.898978 \tag{3.24}$$

According to the Eq. 3.21 and 3.24 we have two different diversity value of instance 1 and 3. On the based of Eq. 3.21 and 3.24 values, we decided which one instance is removes from selected pair. According to the Eq. 3.21 and 3.24 B $<$ A so we

removed instance No. 3 from sample dataset table 3.6. We removes the instance No. 3 because it is less diver, less diver instance is less informative that why we removes the instance No 3.

According to the figure 3.3 when we repeat the third phase process until we reached at required IR, When we achieved desired IR then stop SUS process. As we seen in sample dataset tabel 3.7, MjC and MiC instances have equal number of instance, so we reached at required IR now we end SUS example.

TABLE 3.7: Balanced Sample Dataset

Instance NO.	**Features A**	**Features B**	**Features C**	**Label**
1	3	2	1	YES
2	3	3	2	YES
3	3	3	3	YES
-	-	-	-	-
1	2	3	4	NO
2	2	1	1	NO
3	5	3	4	NO

In this chapter, we explain our proposed under-sampling approach and discussed the detailed explication of the proposed SUS approach. With the help of an example, we explain the process of under-sampling and show the instances information in different tables. With the help of different equations, we explain how we compute the distance between the MiC instances and how we compute the diversity between the selected pair. With the help of the proposed SUS approach, we achieve effective under-sampling with minimum loss of information as shown in the our proposed SUS example.

Chapter 4

Experimental Design

In this chapter, we have discussed experimental design for our proposed approach. We also discuss experimental design phases that are used in the experiment such as data preprocessing, ML model C4.5, and performance measures that are used to evaluate and compare our proposed approach with other existing approaches. First, we discuss under-sampling techniques used in our experiment, ML model used in our experiment, and performance measures used in our experiment to evaluate the performance of the ML model.

Before constructing software defect predictors, we first need to build a learning model with some datasets and evaluate it based on real-world software defect datasets. According to that evaluation, the learning model with better performance would be taken to build a defect predictor for new datasets. The software defect datasets used at the first stage are divided into two subsets, a training dataset for constructing learners and a test dataset for evaluating them. In the second stage, the software defect prediction model is built using all of the datasets, which could improve the general performance of the model. According to figure 4.1, our experiment design is divided into four phases. Following are the phases of experiment design:

1. Data Preprocessing
2. Cross validation

3. Evaluation

4. Comparison of Results

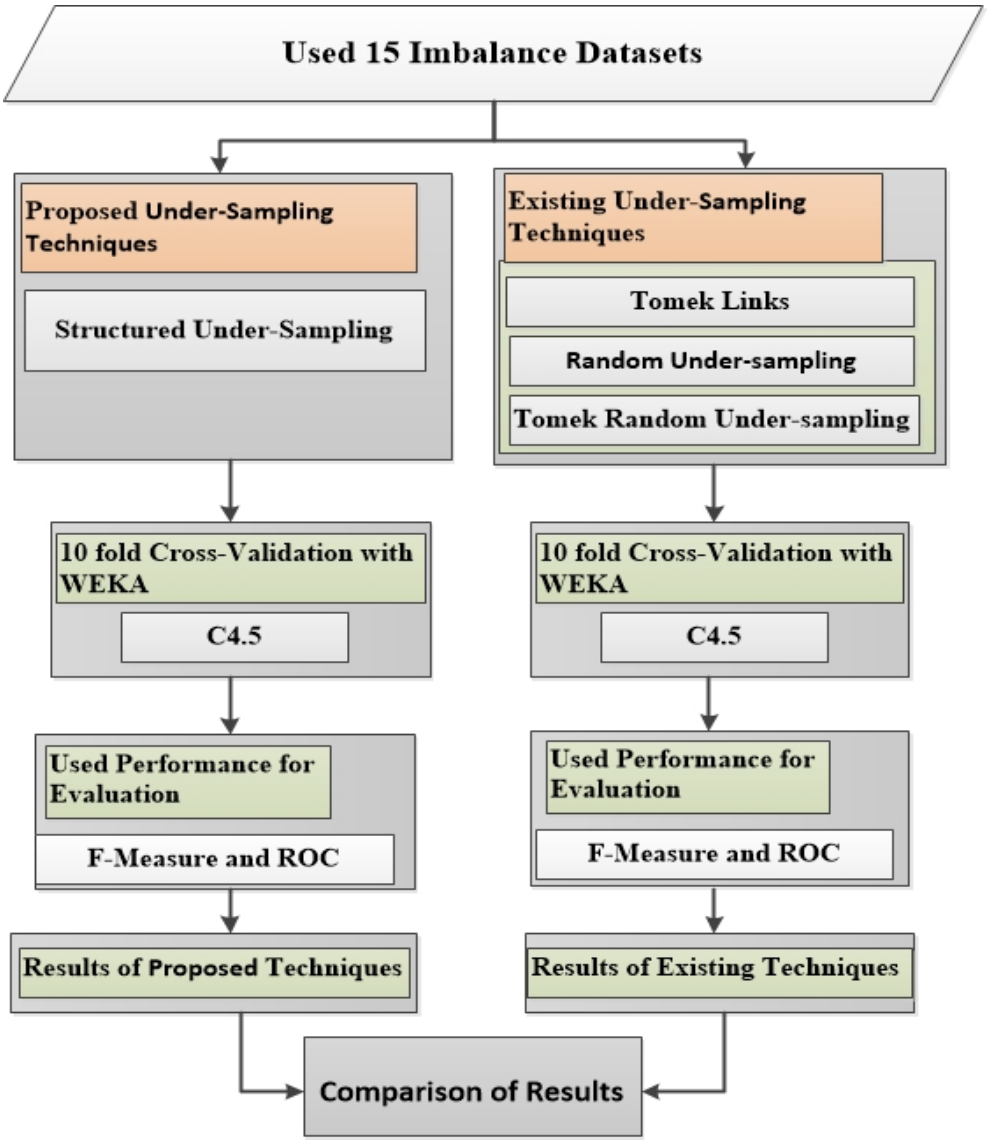

FIGURE 4.1: Experimental Design

4.1 Data Preprocessing

In CIP, class distribution between MjC and MiC instances is the main problem. For this purpose, many techniques are proposed in literature but these methods

try to address the CIP but these techniques have some issues as we discussed in the second chapter.

In our experiment, we have used RUS [47], Tomek links [47], and the combination of RUS and Tomek links and this know as Tomek Random Under-sampling [47]. We compared these three techniques with our proposed approach SUS.

Under-sampling a method is used for data balancing. Under-sampling is using the MjC instances and removes the MjC samples for data balancing between MjC and MiC instances. According to the literature RUS randomly eliminates samples from the MjC until the dataset gets balanced.

For our experiment, we run RUS 50 times across each dataset for the best results. For performance evaluation, we calculate the average percentage of 50 results of each dataset. RUS is the most common preprocessing technique used for data preprocessing.
However, the main drawback of under-sampling is that potentially useful information may be lost. In the literature, there are ways attempts to improve upon the performance of under-samplings, such as Tomek links [47], Condensed Nearest Neighbor Rule [31], and One-sided selection [32], etc.

Tomek link can be defined as follows: Two data points A and B are Tomek joined if, A is the nearest neighbor of B, B is the nearest neighbor of A, and A and B belong to different classes.
Consider the two examples a and b which belong to different classes, and d (a,b) is the distance between a and b. A(a,b) pair is called a Tomek Link if there is not an example c, such that d(a,c)¡d(a,b) or d(b,c)¡d(a,b). If two samples form a Tomek link, then either one of these samples is noise or both samples are border-line and then we can discard the samples. According to Lemaitre et al. [50] Tomek Link is used as data cleaning approach in literature.
Tomek Random Under-sampling [47], TL-RUS use both these RUS and Tomek link techniques for under-sampling. In this technique, the dataset is cleaned with Tomek link and noise is removed from dataset. TL-RUS perform under-sampling but information may be lost due to RUS.

4.2 Cross Validation

In this thesis, we use 10 fold cross-validation test for ML model classification on the PROMISE dataset using the WEKA tool [3]. WEKA tool is a collection of different machine learning algorithms and it also calculates the ML algorithm performance using different performance measures such as F-measure, ROC, FP rate, and TP rate.

First, datasets were download from the PROMISE repository. According to the literature, these downloaded datasets are used for SDP experiments [3]. At each round of the cross-validation, the original dataset is partitioned into 10 subsets in which 9 subsets are used for training learners and the remaining subset is treated as test data. By doing so, the test data set will not be used in building the predictors. The independence of the test set from constructing predictors is crucial for correctly assessing the performances of the predictors.

Software defects datasets are assessed by running them through 10 fold cross-validation over the decision tree algorithm C4.5 using with WEKA ML tool [53]. Experimental results are reported with two performance measures F-measure and ROC. WEKA is a free online available tool and according to literature WEKA is used in many experiments [3]. In our experiment, C4.5 ML algorithm is used for learning on the SDP dataset. The output of C4.5 with four different undersampling approaches are compared based on F-measure and ROC.

4.3 C4.5

Quinlan developed the method of decision tree induction in the 1970s and early 1980s [54]. He continued to improve his method, and his work has resulted in the C4.5 decision tree. In WEKA C4.5 is implemented with the name J48 [53].
The C4.5 algorithm uses a "divide and conquer" strategy to create a tree from a set of data instances. The algorithm attempts to divide the data into increasingly smaller partitions. If the dataset suffers from a severe class imbalance, however,

there may not be enough instances for such an approach to be effective. The learner simply does not have the information it needs to find the appropriate class boundaries. The problem is worse if we have data fragmentation where the MiC tends to cluster in different areas of the instance space. The C4.5 algorithm starts building decision tree with the top node and works downward. At each level of the tree, it greedily selects the attribute which maximizes the information gain and splits the dataset on that attribute. It continues recursively, working each time with smaller partitions of the dataset until it reaches a data partition that contains mostly instances of a single class at which time it creates a leaf node for that class. After the algorithm builds the tree, it then attempts to prune it from the bottom up, either by raising subtrees to higher levels in the decision tree or by replacing subtrees with leaf nodes. Pruning the decision tree generally reduces overfitting to the training data.

4.4 Evaluation

Performance measures are very important for classification evaluation. WEKA has a builtin function to calculate the output of some performance measure such as F-measure, ROC, TP rate, and FP rate as shown in figure 4.2. In our experiment, we report the performance of C4.5 with two performance measures F-measure and ROC. In our experiment, C4.5 ML algorithms are used for learning on the SDP dataset. The output of C4.5 with four different under-sampling approaches are compared based on F-measure and ROC.

4.5 Comparison of Results

In our experiments, we report the performance difference between the proposed under-sampling method SUS with other under-sampling methods as shown in Eq. 4.1 and Eq. 4.2. We calculate the performance difference of F-measure and ROC

as shown in Eq. 4.1 and Eq. 4.2, Diff(F-measures) represents the difference in F-measure and diff(ROC) represent the difference in ROC.

$$Diff(F-measures) = SUS(F-measure_Value) - RUS(F-measure_Value) \tag{4.1}$$

$$Diff(ROC) = SUS(ROC_Value) - RUS(ROC_Value) \tag{4.2}$$

4.6 WEKA

WEKA is a free online available tool and according to literature WEKA is used in many experiments [3]. In WEKA, different machine learning models are categorized into 9 different groups [53]. The name of these 9 groups of ML models are Bayes, Functions, Lazy, Pyscript, Meta, Misc, Rules, Sklearn, and trees as shown in figure 4.2. In the tree group, we highlight the J48, in WEKA C4.5 is implemented with the name J48 [53] model and in the bottom of the result box, we can see the results of different performance measure such ROC and F-measure.

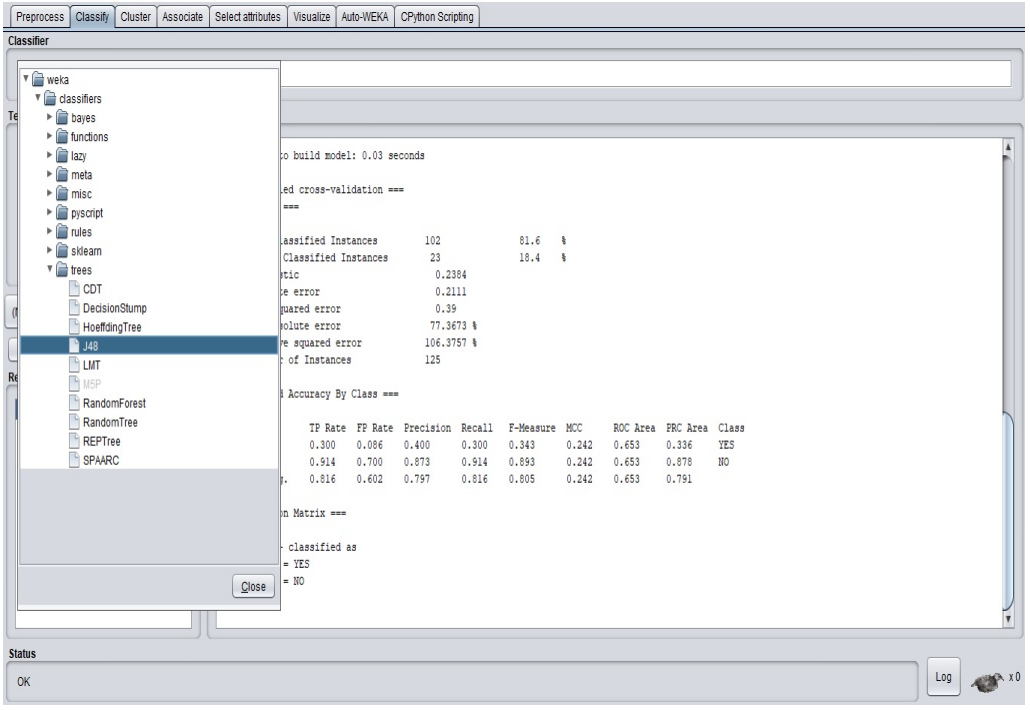

FIGURE 4.2: WEKA UI

4.7 Dataset

TABLE 4.1: Dataset Detail

No.	**Dataset Name**	**MiC**	**MjC**	**MiC + MjC**	**Features**	**Classes**	**IR**
1	velocity-1.6-CK	78	151	229	8	2	1.93
2	synapse-1.2-CK	86	170	256	8	2	1.97
3	jedit-3.2-CK	89	182	271	8	2	2
4	synapse-1-CK	60	162	222	8	2	2.7
5	jedit-4.1-CK	79	233	312	8	2	2.94
6	jedit-4.0-CK	75	231	306	8	2	3.08
7	Eclipse-JDT-Core-CK	206	791	997	8	2	3.83
8	xerces-1.2-CK	71	369	440	8	2	5.19
9	xerces-1.3-CK	69	384	453	8	2	5.56
10	Camel-1.6-CK	129	777	906	8	2	6.02
11	Eclipse-PDE-UI-CK	209	1288	1497	8	2	6.16
12	Camel-1.4-CK	115	727	842	8	2	6.32
13	Camel-1.0-CK	50	323	373	8	2	6.46
14	Mylyn-CK	245	1617	1862	8	2	6.6
15	Lucene-CK	64	627	691	8	2	9.79

In our experiments, we selected 15 datasets out of 27 from PROMISE repository, and these datasets were used in previous literature [3]. For our experiments, we select 15 datasets and in all 15 datasets, the number of MiC instances is > 50. We do not select those datasets which have the MiC < 50. Table 4.1 shows the detail of 15 datasets, this table also shows the number of MiC and MjC instances and the total number of instances, number of features, number of classes, and IR.
We have divided selected 15 datasets into two groups based on IR value. First 7 datasets are in 1^{st} group and 8 to 15 datasets are in 2^{nd} group as shown in table 4.1. In 1^{st} group all datasets have IR < 5 and in 2^{nd} group all datasets have IR $>$ 5, IR in 2^{nd} group datasets is between 5.19 and 9.79. All 15 datasets have same 7 features, in table 4.2 we have listed the number of features and also shown all feature abbreviations.

TABLE 4.2: Shows the Features of Dataset

Name	**Description**
WMC	Weighted methods per class
DIT	Depth of inheritance tree
RFC	Response for a class
NOC	Number of children
CBO	Coupling between object classes
LCOM	Lack of cohesion of methods
LOC	Line of code

In this chapter, we explain the experimental setup, and we also discuss how we evaluate the proposed approach with other approaches. In this chapter, we discuss the process of cross-validation and also the GUI of WEKA. In this chapter, we show the datasets in detail and also discuss the process of comparison of results.

Chapter 5

Results and Discussion

In this chapter, we discuss the experiment results and evaluate performance of the SUS. We compare the proposed method SUS with Tomek Links, Random under-sampling, and Tomke Random under-sampling.

These previous techniques were implemented in Python and the toolbox is publicly available at GitHub [50]. We compare the proposed method SUS with under-sampling techniques via using IR 1:1.

For better evaluation, we run RUS 50 times across each dataset and compare the average of 50 runs with SUS. We have evaluated each dataset by using F-measure and ROC. As for the classification, we used C4.5, each of the experiments is done with 10 fold cross-validation. The RUS experiments were repeated 50 times to produce statistically reliable results.

The performance of the classifiers is compared by using the ROC and F-measure. In the literature, ROC and F-measure are commonly used as performance measures in the SDP field, and C4.5 is also commonly used in the SDP field.

In SDP studies different ML models are used for classification, according to the Malhotra study, C4.5 is frequently used for SDP [4]. C4.5 gives good results on imbalanced datasets in SDP field [13] [4]. Performance measures F-measure and ROC generally used for SDP [4] [2].

5.1 Experiment

In this section, we discuss the performance of under-sampling techniques with imbalance datasets and compare the results with our proposed technique called Structured Under-Sampling (SUS). We also try to find out how IR affects the final result, which is discussed in more detail in the following subsections.
In our experiments, we report the performance difference between the proposed under-sampling method SUS and other under-ampling methods. The performance measure diff(F-measures) represents the difference in F-measure and diff(ROC) represent the difference in ROC.
For better evaluation of 15 datasets, we split them into two groups according to IR. All datasets have different IR as shown in table 4.1, so we divide these datasets into two groups. In the first group, all datasets have IR<5 and in the second group, all datasets have IR>5. These two groups help to understand the performance of SUS with other existing under-sampling methods. We also see the impact of IR on the sampling technique and get the best analysis.

5.1.1 SUS vs. Tomek link

In this experiment, we compare the performance of SUS and Tomek link on 15 datasets. The results of this experiment are analyzed according to the IR value.

Table 5.1 represents diff(F-measure) and diff(ROC) on 7 datasets with imbalance ratio is less than 5 (IR $<$ 5).
Table 5.2 represents diff(F-measure) and diff(ROC) on 8 datasets with imbalance ratio is greater than 5 (IR $>$ 5). Table 5.1 shows negative results in bold face. These results show SUS performance is not improved, in table 5.1 the first group SUS overall performance is better than Tomek link, while all the positive values of diff(F-measure) and diff(ROC) show better performance from Tomek link. If we analyze diff(F-measure), it shows better performance from SUS on 4 datasets out of 7 as compared to T-links which performs better on 3 datasets out of 7.

TABLE 5.1: Performance difference of SUS and Tomek link when IR < 5

Dataset Name	SUS vs. Tomek Link 1^{st} **Group (IR <5)**					
	C4.5					
	SUS-(F-measure)	**T-Links-(F-measure)**	**Diff-(F-measure)**	**SUS-(ROC)**	**T-Links-(ROC)**	**Diff-(ROC)**
velocity-1.6-CK	0.55	No Output	0.55	0.56	0.481	0.079
synapse-1.2-CK	0.546	0.554	**-0.008**	0.54	0.504	0.04
jedit-3.2-CK	0.683	0.508	0.175	0.69	0.506	0.186
synapse-1.1-CK	0.542	0.587	**-0.045**	0.56	0.489	0.069
jedit-4.1-CK	0.624	No Output	0.624	0.62	0.494	0.127
jedit-4.0-CK	0.595	No Output	0.595	0.65	0.478	0.168
Eclipse-JDT-Core-CK	0.648	0.685	**-0.037**	0.69	0.497	0.19

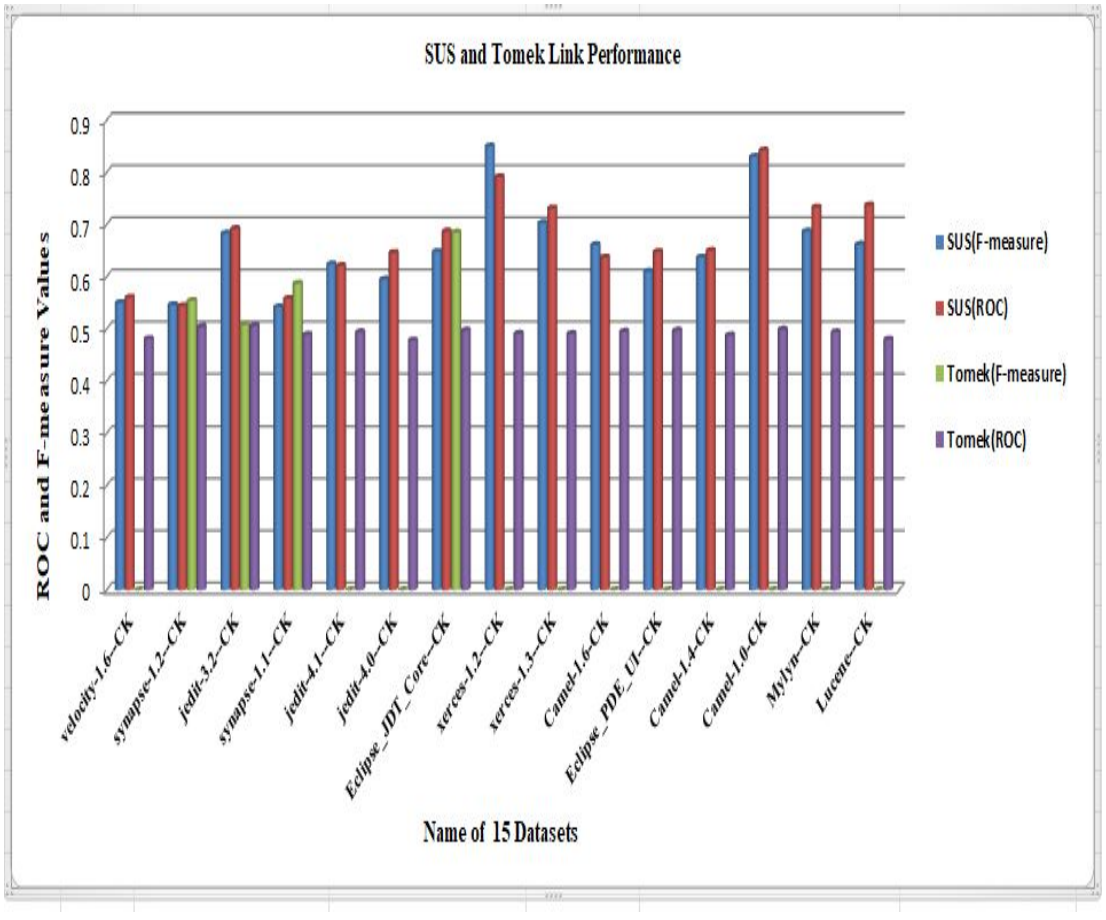

FIGURE 5.1: SUS and Tomek link Performance

TABLE 5.2: Performance difference of SUS and Tomek Link when IR > 5

Dataset Name	**SUS vs. Tomek Link** 2^{nd} **Group (IR > 5)**					
	C4.5					
	SUS-(F-measure)	**T-Links-(F-measure)**	**Diff-(F-measure)**	**SUS-(ROC)**	**T-Links-(ROC)**	**Diff-(ROC)**
xerces-1.2-CK	0.85	No Output	0.85	0.791	0.491	0.3
xerces-1.3-CK	0.703	No Output	0.703	0.731	0.491	0.24
Camel-1.6-CK	0.661	No Output	0.661	0.637	0.495	0.142
Eclipse-PDE-UI-CK	0.61	No Output	0.61	0.648	0.497	0.151
Camel-1.4-CK	0.637	No Output	0.637	0.65	0.488	0.162
Camel-1.0-CK	0.83	No Output	0.83	0.842	0.499	0.343
Mylyn-CK	0.687	No Output	0.687	0.733	0.494	0.239
Lucene-CK	0.662	No Output	0.662	0.737	0.48	0.257

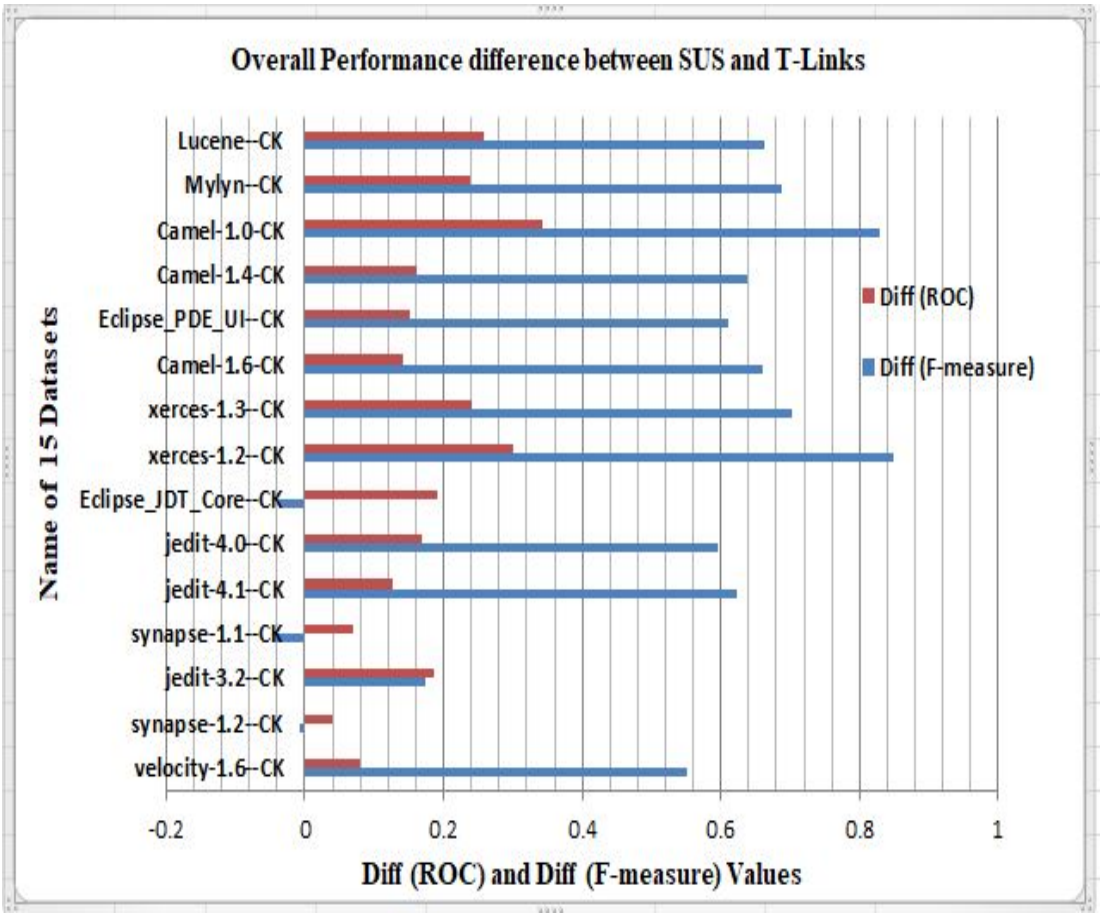

FIGURE 5.2: Overall Performance difference between SUS and Tomek link

So we can conclude from diff(F-measure) SUS has performed better than Tomek link. On the other hand, when we talk about performance measure ROC, SUS with diff(ROC) outperforms Tomek link on all 7 datasets.
In Table 5.2 SUS has again outperformed Tomek link with all positive values of diff(ROC) and diff(F-measure). In table 5.2, SUS performance further improves for SDP when IR is greater than 5.

Figure 5.2 shows the performance difference between SUS and T-links, According to figure 5.2 SUS has outperformed Tomek link on all 15 datasets with diff(ROC) and on 12 out of 15 datasets with diff(F-measure). According to figure 5.1 and 5.2, we conclude that SUS is a good under-sampling technique in the scenario when IR is greater than 5.

5.1.2 SUS vs. RUS

In the 2nd experiment, we compare the performance of SUS and RUS. According to table 5.3, 5.4 and figure 5.3 and 5.4, we can see the three different behaviors of SUS. Table 5.3 and 5.4, show that the performance of SUS changes when IR changes. In table 5.3, the number of negative values is greater than the number of positive values with both performance measures diff(ROC) and diff(F-measure), which means that the performance of RUS is better than SUS when IR < 5. RUS performs better on 5 datasets out of 7, only one dataset shows equal performance for SUS and RUS which is jedit-3.2. In Table 5.4, we can see that both performance measures diff(ROC) and diff(F-measure) have shown SUS outperforms RUS on all datasets. Positive values represent better performance of SUS as compare to RUS on all datasets of the 2nd group. Only one dataset has not shown any improvement in the performance of SUS which is Camel-1.4-CK.
Figure 5.3 shows the overall performance of SUS and RUS. Figure 5.4 shows the performance difference of SUS and RUS. According to figure 5.4, SUS has outperformed RUS on all 15 datasets with diff(ROC) and on 9 out of 15 datasets with diff(F-measure). According to these findings, we conclude that SUS is a good under-sampling technique in the scenario when IR > 5.

TABLE 5.3: Performance difference of SUS and RUS when IR < 5

Dataset Name	**SUS vs. RUS**					
	1st Group (IR <5)					
	C4.5					
	SUS-(F-measure)	**RUS-(F-measure)**	**Diff-(F-measure)**	**SUS-(ROC)**	**RUS-(ROC)**	**Diff-(ROC)**
velocity-1.6-CK	0.55	0.54	0.01	0.56	0.55	0.01
synapse-1.2-CK	0.546	0.695	**-0.149**	0.544	0.691	**-0.147**
jedit-3.2-CK	0.683	0.681	0.002	0.692	0.688	0.004
synapse-1.1-CK	0.542	0.675	**-0.113**	0.558	0.671	**-0.113**
jedit-4.1-CK	0.624	0.701	**-0.077**	0.621	0.708	**-0.087**
jedit-4.0-CK	0.595	0.706	**-0.111**	0.646	0.71	**-0.064**
Eclipse-JDT-Core-CK	0.648	0.693	**-0.045**	0.687	0.701	**-0.014**

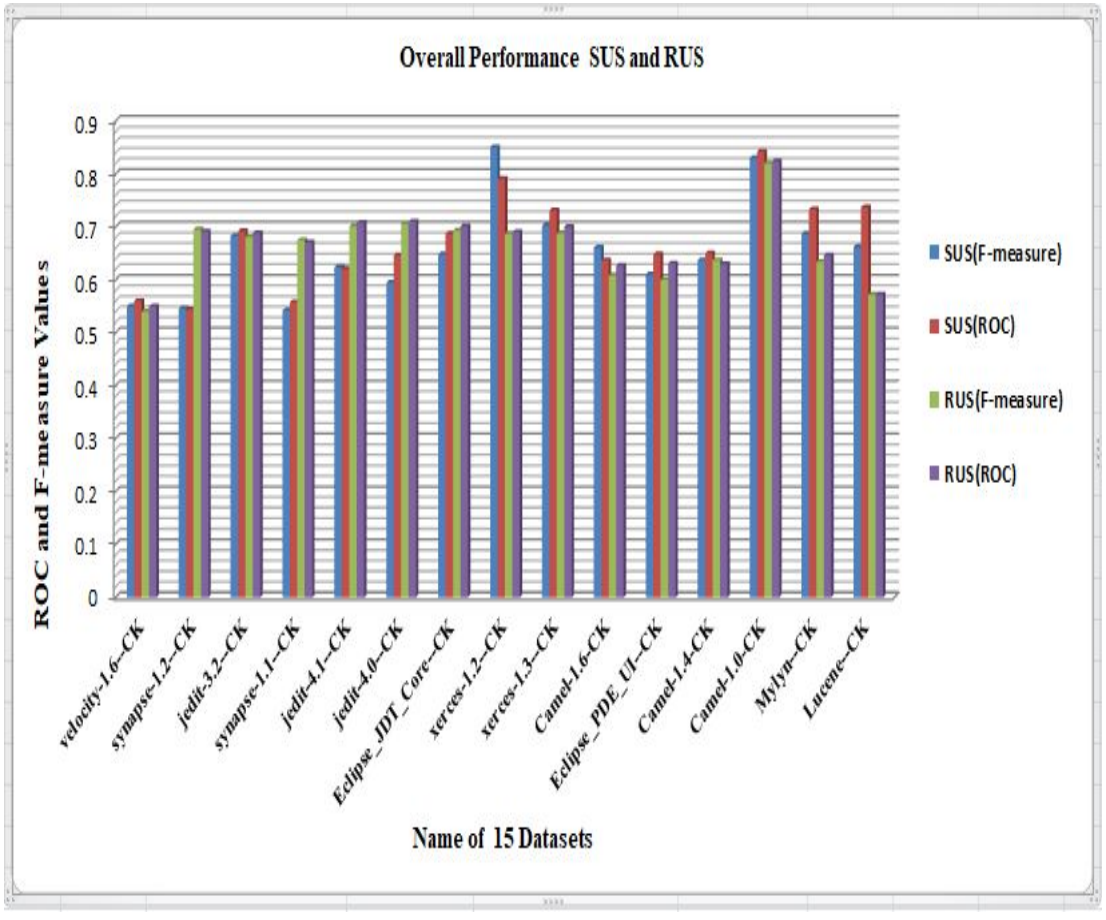

FIGURE 5.3: Overall Performance difference between SUS and RUS

TABLE 5.4: Performance difference of SUS and RUS when IR > 5

Dataset Name	SUS vs. RUS 2nd Group (IR >5) C4.5					
	SUS (F-measure)	**RUS (F-measure)**	**Diff (F-measure)**	**SUS (ROC)**	**RUS (ROC)**	**Diff (ROC)**
xerces-1.2-CK	0.85	0.687	0.163	0.791	0.69	0.101
xerces-1.3-CK	0.703	0.688	0.015	0.731	0.7	0.031
Camel-1.6-CK	0.661	0.608	0.053	0.637	0.627	0.01
Eclipse-PDE-UI-CK	0.61	0.6	0.01	0.648	0.631	0.017
Camel-1.4-CK	0.637	0.637	0	0.65	0.63	0.02
Camel-1.0-CK	0.83	0.82	0.01	0.842	0.824	0.018
Mylyn-CK	0.687	0.634	0.053	0.733	0.646	0.087
Lucene-CK	0.662	0.572	0.09	0.737	0.572	0.165

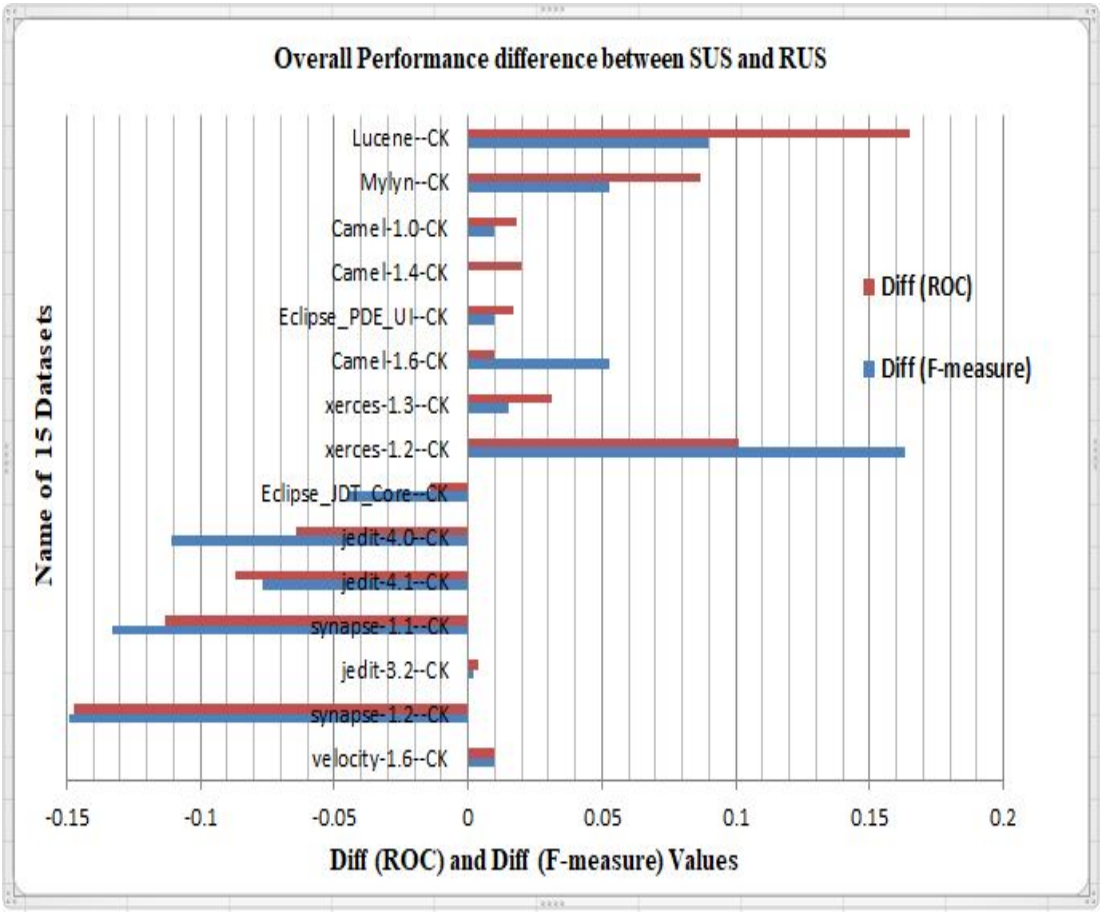

FIGURE 5.4: Overall Performance difference between SUS and RUS

5.1.3 SUS vs. TL-RUS

In this third experiment, we compare the performance of SUS and TL-RUS. TL-RUS is a combination RUS and Tomek links. Tomek links is used as a data cleaning method and RUS is used for the under-sampling of the MjC. Tomek links is a data cleaning method, cleaning under-sampling does not allow to reach a specific IR [50]. Elhassan results show performance improved when applying Tomek links used as a data cleaning method before the different sampling techniques [47].

According to the results of table 5.5, 5.6 and figure 5.5, 5.6, SUS shows outstanding performance and SUS outperforming TL-RUS in both groups.
Figure 5.6 shows the performance difference between SUS and TL-RUS. According to figure 5.6, SUS has outperformed on all 15 datasets with diff(ROC) and on 15 out of 15 datasets with diff(F-measure).

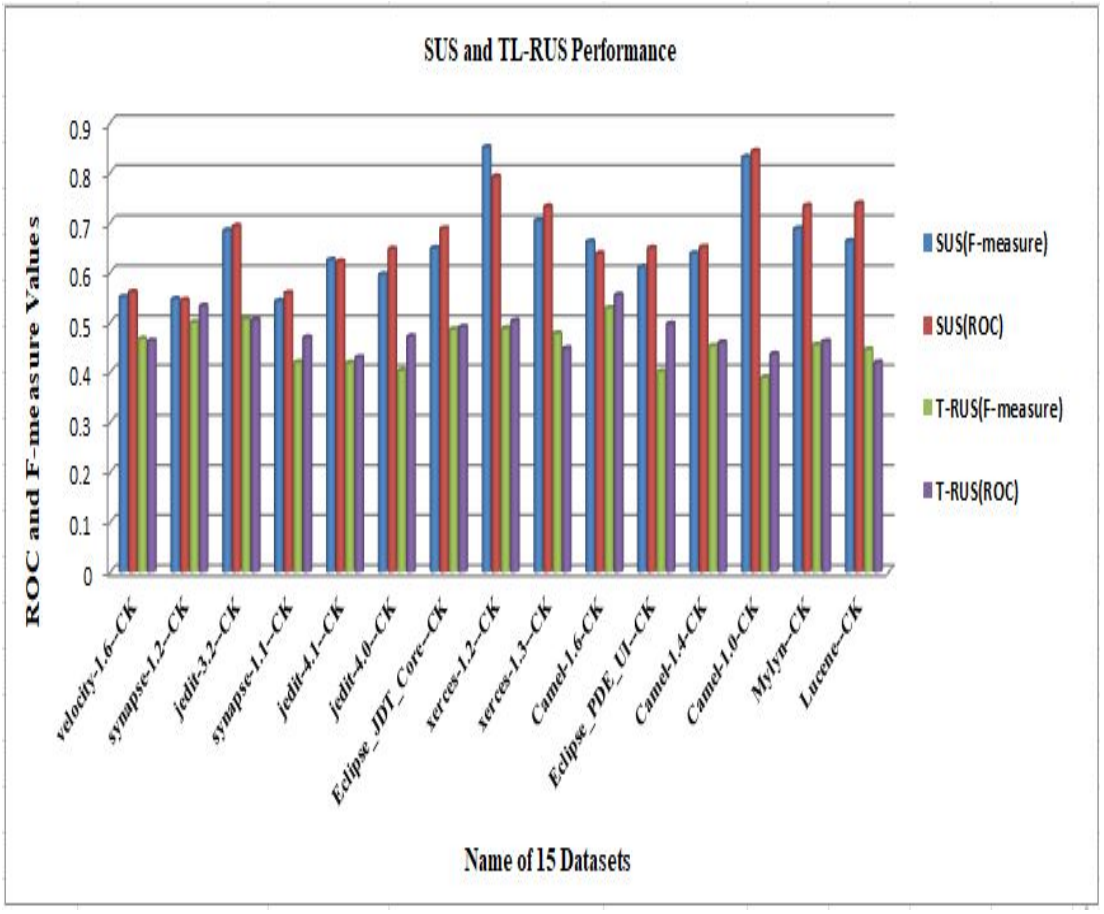

FIGURE 5.5: Overall Performance SUS and TL-RUS

TABLE 5.5: performance difference between SUS and TL-RUS when IR < 5

Dataset Name	**SUS vs. TL-RUS 1st Group (IR <5)**					
	C4.5					
	SUS-(F-measure)	**TL-RUS-(F-measure)**	**Diff-(F-measure)**	**SUS-(ROC)**	**TL-RUS-(ROC)**	**Diff-(ROC)**
velocity-1.6-CK	0.55	0.466	0.084	0.56	0.462	0.098
synapse-1.2-CK	0.546	0.499	0.047	0.544	0.532	0.012
jedit-3.2-CK	0.683	0.506	0.177	0.692	0.504	0.188
synapse-1.1-CK	0.542	0.419	0.123	0.558	0.469	0.089
jedit-4.1-CK	0.624	0.418	0.206	0.621	0.429	0.192
jedit-4.0-CK	0.595	0.403	0.192	0.646	0.471	0.175
Eclipse-JDT-Core-CK	0.648	0.485	0.163	0.687	0.49	0.197

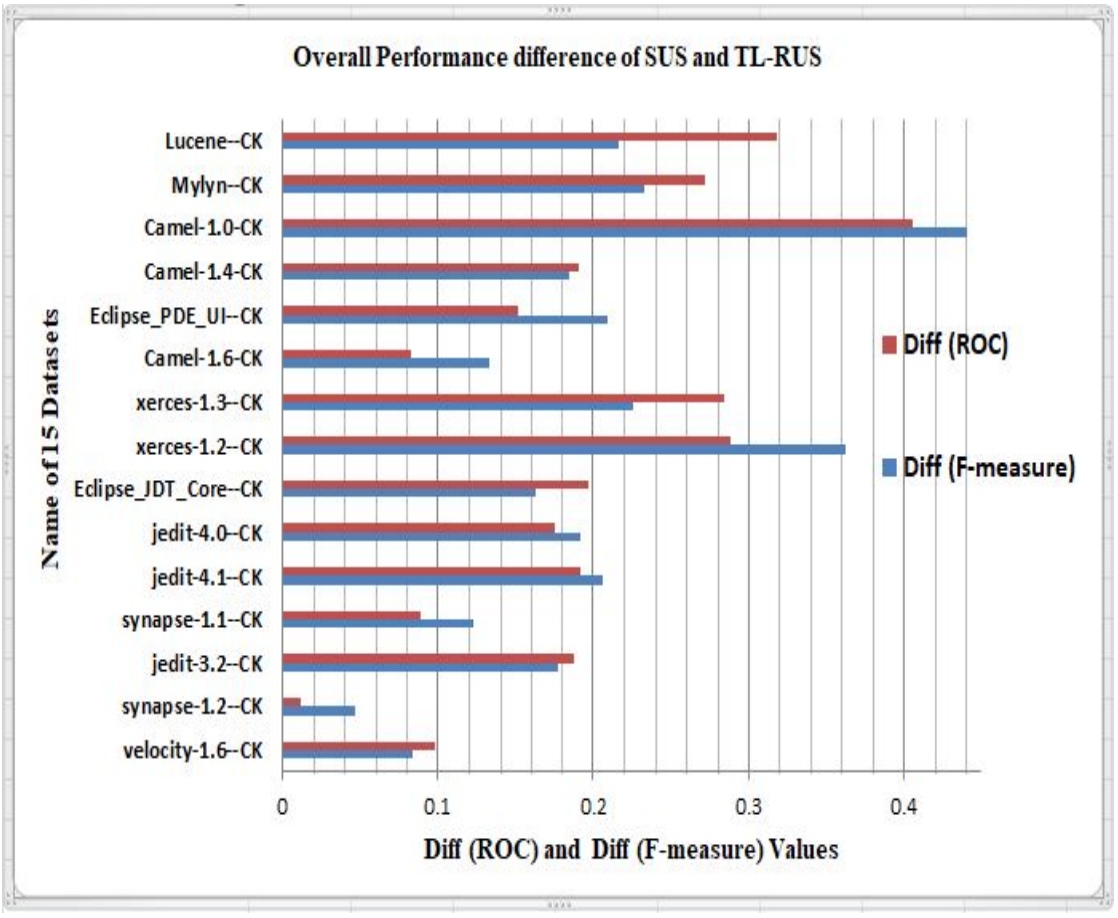

FIGURE 5.6: Overall Performance difference between SUS and TL-RUS

TABLE 5.6: Performance Difference between SUS and TL-RUS when IR > 5

Dataset Name	**SUS vs. TL-RUS 2nd Group (IR >5)**					
	C4.5					
	SUS-(F-measure)	**TL-RUS-(F-measure)**	**Diff-(F-measure)**	**SUS-(ROC)**	**TL-RUS-(ROC)**	**Diff-(ROC)**
xerces-1.2-CK	0.85	0.487	0.363	0.791	0.502	0.289
xerces-1.3-CK	0.703	0.477	0.226	0.731	0.447	0.284
Camel-1.6-CK	0.661	0.528	0.133	0.637	0.554	0.083
Eclipse-PDE-UI-CK	0.61	0.401	0.209	0.648	0.496	0.152
Camel-1.4-CK	0.637	0.452	0.185	0.65	0.459	0.191
Camel-1.0-CK	0.83	0.389	0.441	0.842	0.436	0.406
Mylyn-CK	0.687	0.454	0.233	0.733	0.461	0.272
Lucene-CK	0.662	0.445	0.217	0.737	0.419	0.318

5.2 Discussion

Imbalance data has a significant impact on the performance of standard classification algorithms. Applying the standard classification algorithms without any adjustment results in a classification biased towards the majority class.

For this Issue, we recommend the use of SUS as a under-sampling method. Our experiments results shows better performance of SUS as compare to other existing under-sampling methods. SUS removes three different types of instances from MjC with help of EDF. The proposed method, SUS overcome the problem of information lost as compared to RUS to reach the specific IR loses minimum information which therefore improves the performance of the classification algorithm.

The results of this investigation show a superior performance of SUS as compared to previous sampling techniques. Software defect datasets show the superiority of method SUS with ML algorithms C4.5. Some datasets showed a comparable performance using sampling methods. Figure 5.7 and 5.8 show experiment results on 15 datasets and show the performance difference using the performance measures diff(ROC) and diff(F-measure) with three under-sampling methods.

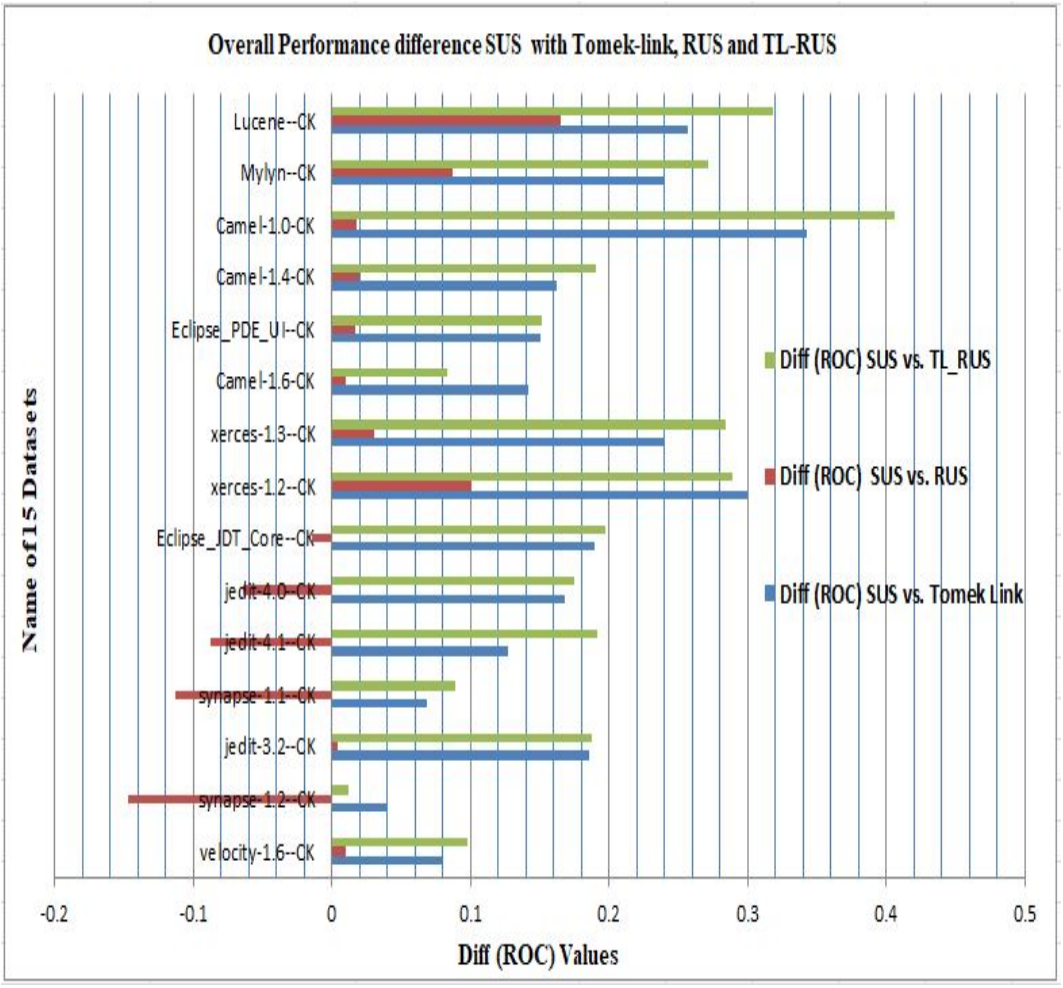

FIGURE 5.7: Overall Performance difference of SUS between Tomek-Link, RUS and TL-RUS

Figure 5.7 and 5.8 show experiment results on 15 datasets and show the performance difference using the performance measures diff(ROC) and diff(F-measure) with three under-sampling methods.

In figure 5.7 and 5.8 blue line shows the performance difference of SUS and Tomek link with performance measures diff(ROC) and diff(F-measure) respectively. According to 5.7 the blue line shows the positive results in all 15 datasets. In figure 5.7 the blue line shows SUS outperformed as compared to the Tomek links using performance measure ROC. But in the figure 5.8 SUS shows a little different results as compared to figure 5.7. Figure 5.8 shows the performance difference between SUS and Tomek link with the blue line and this blue line is below zero in three datasets. According to the figure 5.8, we can see that performance of SUS

is not improved in 3 datasets out of 15 datasets. But when we analyze the overall performance difference between SUS and Tomek link we see the overall SUS is outperforms with 15 datasets.

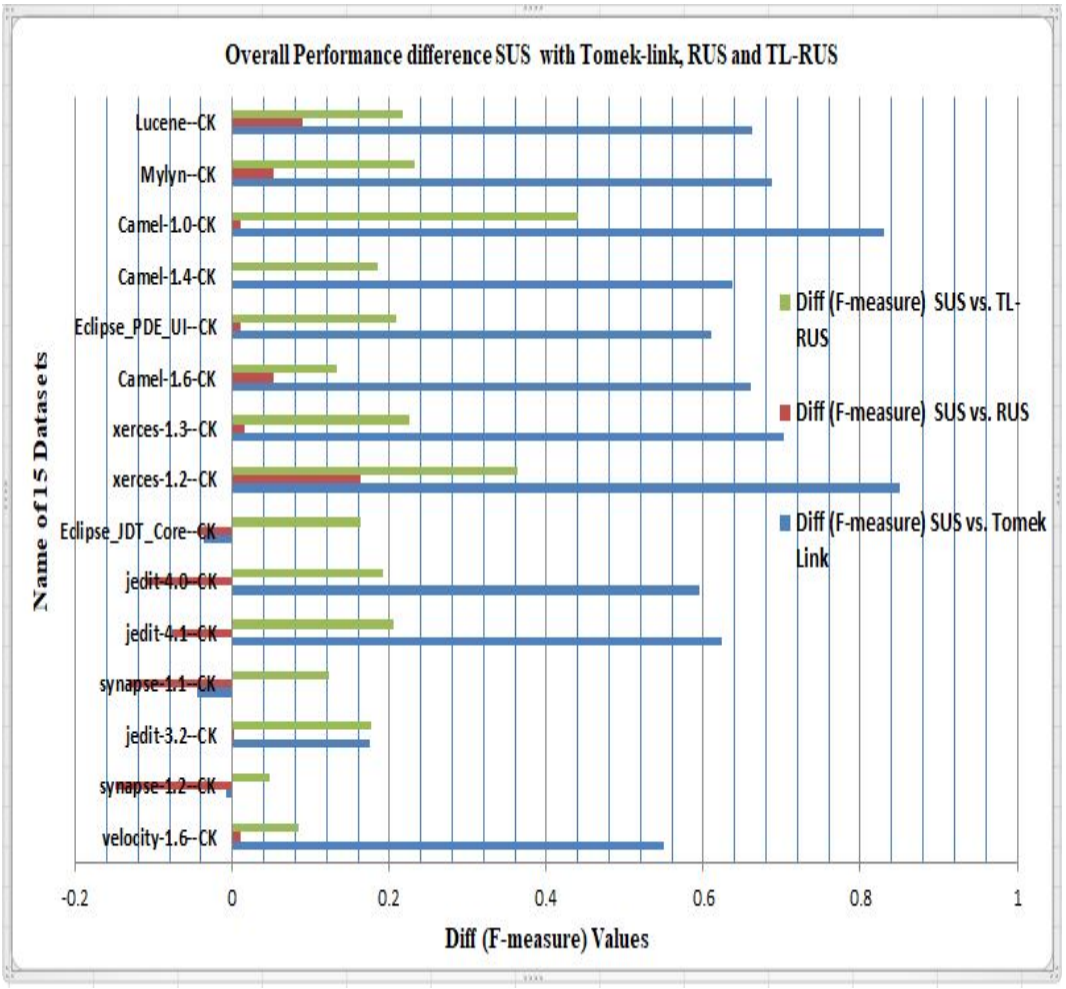

FIGURE 5.8: Performance difference of SUS between Tomek-Link, RUS and TL-RUS

In figure 5.7 and 5.8, red line shows the performance difference of SUS and RUS with performance measures diff(ROC) and diff(F-measure) respectively. According to 5.7 the red line shows positive results in 11 datasets and shows negative results on 5 datasets. According to the figure 5.7 and 5.8 we see that performance of SUS is not improved when IR is < 5. So, the performance difference in figure 5.7 and 5.8, shows overall SUS outperformed with both performance measures diff(ROC) and diff(F-measure) on 11 datasets out of 15 datasets.

In figure 5.7 and 5.8, green line shows the performance difference of SUS and

TL-RUS with performance measures diff(ROC) and diff(F-mesaure) respectively. According to figure 5.7 and 5.8 the green line shows positive results in all 15. According to these figure 5.7 and 5.8 SUS outperformed on 15 datasets with both performance measures diff(ROC) and diff(F-measure).

Figure 5.7 and 5.8, show the overall performance difference diff(ROC) and diff(F-measure) respectively. In both figures 5.7 and 5.8 the performance of SUS is not improved when IR is less than 5 (IR < 5) but on the other hand when IR > 5 the performance of SUS is improved and SUS is outperformed on all datasets which have IR > 5.

In this chapter, we explain the results of our proposed approach SUS with existing under-sampling approaches. With help of different tables, we show the output of different results and with help of different figures, we show the performance difference of the proposed approach with other existing approaches. The overall performance of the proposed approach SUS is better as compared to other existing approaches.

Chapter 6

Conclusion and Future Work

In this chapter, we make some final comments about our new under-sampling technique. In chapter 5, we discussed comparison of the proposed SUS results with other previous under-sampling techniques results. In this thesis, we have introduced a new approach that involves under-sampling of MjC in a systematic way. We compared our SUS approach with RUS, Tomek Links, and the combination of RUS and Tomek Links. The comparison results described in chapter 5 reveal that our under-sampling approach is generally better and outperforms the other under-sampling techniques. While RUS and Tomek Links techniques do perform well on some datasets when IR < 5, SUS shows far better results when we speak about the overall result.

RQ1: How can we achieve an effective under-sampling with minimum loss of information as compared to the other existing under-sampling approaches?

In the chapter 2.2, we discuss many under-sampling techniques and all these techniques used the EDF to remove the noisy instances for MiC and in literature these techniques are also known as data reduction methods. In some methods, when removing an instance its must be present in both classes then we remove this instance from the majority class. Some methods use nearest neighbors for removing

instances but the decision of which data point is selected for data reduction is based on both classes MiC and MjC.

In the proposed technique, the structured under-sampling (SUS) we systematically remove instances. In chapter 3, we discuss SUS has three phases to remove the samples from MjC and reduce the size of MjC in a structured way.

In the first phase, we identify inconsistent samples, and remove them from MjC. In the second phase, we identify and remove the same sample or duplicate instances with the same label. The third phase, removes the most similar instances in the MjC class. This phase has to achieve the desired IR.

In section 5.2, experiments show that SUS overall outperformed other sampling techniques. In some datasets when IR < 5 the performance of SUS is affected but when IR > 5 performance improves.

RQ2: Does the proposed under-sampling approach improve the performance over existing under-sampling approaches?

We compare the proposed method SUS with Tomek Links, Random under-sampling, and Tomke Random under-sampling. These previous techniques were implemented in Python and the toolbox is publicly available at GitHub [50]. We also compare the proposed method SUS with under-sampling techniques using IR 1:1. We have validated each dataset by using F-measure and ROC. As for the classification, we used C4.5, each experiment is done 10 cross folds validation.

Figure 5.2 shows the performance difference between SUS and T-links, According to figure 5.2 SUS has outperformed on all 15 datasets with diff(ROC) and on 12 out of 15 datasets with diff(F-measure). According to these findings, we conclude that SUS is a good sampling technique in the scenario when IR greater than 5.

Figure 5.4 shows the performance difference between SUS and RUS, According to figure 5.4 SUS has outperformed on all 15 datasets with diff(ROC) and on 9 out of 15 datasets with diff(F-measure). According to these findings, we conclude that SUS is a good resampling technique in the scenario when IR > 5.

In figure 5.6 shows the performance difference between SUS and TL-RUS, According to figure 5.6 SUS has outperformed on all 15 datasets with diff(ROC) and on

15 out of 15 datasets with diff(F-measure).
The results of this investigation show a superior performance using SUS to the prior resampling techniques. Software defect datasets showed the superiority of method SUS after using ML algorithms C4.5. Some datasets showed a comparable performance using all sampling methods

RQ3: How does imbalance ratio affect the performance of proposed under-sampling approaches?

IR is defined as the fraction of the number of non-defective (MjC) samples in the number of defective (MiC) samples as shown in Eq.1.1. The effect of IR is clearly seen in table 5.1, 5.2, 5.3, 5.4, 5.5 and 5.6, the performance of SUS changes when IR changes. In tables 5.2, 5.6 and 5.4, when IR is greater than 5 (IR $>$ 5) SUS outperforms in all experiments. We can say that, SUS performs well when IR is greater than 5.

Figure 5.7 and 5.8 show the overall all performance difference diff(ROC) and diff(F-measure) respectively. In both figures 5.7 and 5.8 the performance of SUS is not improving when IR is less than 5 (IR $<$ 5) but on the other hand when IR $>$ 5 the performance of SUS is improved 5.

For future work, we can combine SUS with other sampling approaches in the domain of SDP. Many authors combine under-sampling with over-sampling techniques. The over-sampling approach increases the size of MiC with random selection or with the adaptive approach. In the literature, different over-sampling techniques have been proposed such as SMOTE, MSMOTE, Borderline-SMOTE, and ADASYN. For future work, we combine our proposed technique SUS with other over-sampling and investigate the behavior of SUS. With the help of SUS, we reduced the size of MjC with specific IR. When we have done under-sampling with our proposed technique then use other over-sampling techniques and increases the size of MiC. With the help of under-oversampling we achieve desired IR.
In the conclusion of future work, we find the best combination of SUS with other over-sampling techniques. For future experiments, we use 27 datasets from

PROMISE repository. We also compare the proposed method SUS with over-sampling techniques using IR 1:1. For better evaluation, we run Random over-sampling 50 times across each dataset and compare the average of 50 runs with SUS. As for the classification, we used C4.5, the performance of the classifier is compared by using the ROC and F-measure.

Bibliography

[1] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[2] M. Rizwan, A. Nadeem, and M. A. Sindhu, "Analyses of classifier's performance measures used in software fault prediction studies," *IEEE Access*, vol. 7, pp. 82 764–82 775, 2019.

[3] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 12, pp. 1253–1269, 2018.

[4] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[5] A. Smola and S. Vishwanathan, "Introduction to machine learning," *Cambridge University, UK*, vol. 32, no. 34, p. 2008, 2008.

[6] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 15–25.

[7] Q. Kang, X. Chen, S. Li, and M. Zhou, "A noise-filtered under-sampling scheme for imbalanced classification," *IEEE transactions on cybernetics*, vol. 47, no. 12, pp. 4263–4274, 2016.

[8] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering*, vol. 33, no. 1, pp. 2–13, 2006.

[9] J. Nam, "Survey on software defect prediction," *Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep*, 2014.

[10] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4, pp. 531–577, 2012.

[11] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, "High-impact defects: a study of breakage and surprise defects," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 300–310.

[12] T. Lee, J. Nam, D. Han, S. Kim, and H. P. In, "Micro interaction metrics for defect prediction," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 311–321.

[13] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 181–190.

[14] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 284–292.

[15] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 4, pp. 86–96, 2004.

[16] V. Y. Shen, S. D. Conte, and H. E. Dunsmore, "Software science revisited: A critical analysis of the theory and its empirical support," *IEEE Transactions on Software Engineering*, no. 2, pp. 155–165, 1983.

[17] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE transactions on software engineering*, vol. 37, no. 6, pp. 772–787, 2010.

[18] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 9–9.

[19] A. Tosun, B. Turhan, and A. Bener, "Validation of network measures as indicators of defective modules in software systems," in *Proceedings of the 5th international conference on predictor models in software engineering*, 2009, pp. 1–9.

[20] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[21] W. W. Cohen, "Fast effective rule induction," in *Machine learning proceedings 1995*. Elsevier, 1995, pp. 115–123.

[22] R. Premraj and K. Herzig, "Network versus code metrics to predict defects: A replication study," in *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2011, pp. 215–224.

[23] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[24] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the" imprecision" of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th*

International Symposium on the Foundations of Software Engineering, 2012, pp. 1–11.

[25] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[26] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to" comments on'data mining static code attributes to learn defect predictors'"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 637–640, 2007.

[27] X. Xuan, D. Lo, X. Xia, and Y. Tian, "Evaluating defect prediction approaches using a massive set of metrics: An empirical study," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 1644–1647.

[28] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 481–490.

[29] C.-S. Lin, G.-H. Tzeng, and Y.-C. Chin, "Combined rough set theory and flow network graph to predict customer churn in credit card accounts," *Expert Systems with Applications*, vol. 38, no. 1, pp. 8–15, 2011.

[30] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[31] P. Hart, "The condensed nearest neighbor rule (corresp.)," *IEEE transactions on information theory*, vol. 14, no. 3, pp. 515–516, 1968.

[32] M. Kubat, S. Matwin *et al.*, "Addressing the curse of imbalanced training sets: one-sided selection," in *Icml*, vol. 97. Citeseer, 1997, pp. 179–186.

[33] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[34] I. Mani and I. Zhang, "knn approach to unbalanced data distributions: a case study involving information extraction," in *Proceedings of workshop on learning from imbalanced datasets*, vol. 126, 2003.

[35] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[36] S. Hu, Y. Liang, L. Ma, and Y. He, "Msmote: Improving classification performance when training data is imbalanced," in *2009 second international workshop on computer science and engineering*, vol. 2. IEEE, 2009, pp. 13–17.

[37] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *International conference on intelligent computing*. Springer, 2005, pp. 878–887.

[38] J. Laurikkala, "Improving identification of difficult small classes by balancing class distribution," in *Conference on Artificial Intelligence in Medicine in Europe*. Springer, 2001, pp. 63–66.

[39] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2008.

[40] S.-J. Yen and Y.-S. Lee, "Cluster-based under-sampling approaches for imbalanced data distributions," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5718–5727, 2009.

[41] M. A. Tahir, J. Kittler, and F. Yan, "Inverse random under sampling for class imbalance problem and its application to multi-label classification," *Pattern Recognition*, vol. 45, no. 10, pp. 3738–3750, 2012.

[42] W. W. Ng, J. Hu, D. S. Yeung, S. Yin, and F. Roli, "Diversified sensitivity-based undersampling for imbalance classification problems," *IEEE transactions on cybernetics*, vol. 45, no. 11, pp. 2402–2412, 2014.

[43] M. Beckmann, N. F. Ebecken, B. S. P. de Lima *et al.*, "A knn undersampling approach for data balancing," *Journal of Intelligent Learning Systems and Applications*, vol. 7, no. 04, p. 104, 2015.

[44] W. Liu, S. Liu, Q. Gu, J. Chen, X. Chen, and D. Chen, "Empirical studies of a two-stage data preprocessing approach for software fault prediction," *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 38–53, 2015.

[45] C. Yang, Y. Gao, J. Xiang, and L. Liang, "Software defect prediction based on conditional random field in imbalance distribution," in *2015 2nd International Symposium on Dependable Computing and Internet of Things (DCIT)*. IEEE, 2015, pp. 67–71.

[46] X. Chen, Q. Kang, M. Zhou, and Z. Wei, "A novel under-sampling algorithm based on iterative-partitioning filters for imbalanced classification," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2016, pp. 490–494.

[47] T. Elhassan and M. Aljurf, "Classification of imbalance data using tomek link (t-link) combined with random under-sampling (rus) as a data reduction method," 2016.

[48] R. A. Sowah, M. A. Agebure, G. A. Mills, K. M. Koumadi, and S. Y. Fiawoo, "New cluster undersampling technique for class imbalance learning," *International Journal of Machine Learning and Computing*, vol. 6, no. 3, p. 205, 2016.

[49] I. Tomek *et al.*, "An experiment with the edited nearest-nieghbor rule." 1976.

[50] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 559–563, 2017.

[51] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and image processing*, vol. 14, no. 3, pp. 227–248, 1980.

[52] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[53] R. R. Bouckaert, E. Frank, M. Hall, R. Kirkby, P. Reutemann, A. Seewald, and D. Scuse, "Weka manual for version 3-9-1," *University of Waikato, Hamilton, New Zealand*, 2016.

[54] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.