2020-08

DESIGNING SOFTWARE DEFECT PREDICTION MODEL USING FILTER FEATURE SELECTION AND SUPPORT VECTOR MACHINE ALGORITHMS

ZELALEM, FISSIHA

BAHIR DAR UNIVERSITY

BAHIR DAR INSTITUTE OF TECHNOLOGY

SCHOOL OF RESEARCH AND POSTGRADUATE STUDIES

FACULTY OF COMPUTING

DESIGNING SOFTWARE DEFECT PREDICTION MODEL USING FILTER FEATURE SELECTION AND SUPPORT VECTOR MACHINE ALGORITHMS

BY

ZELALEM FISSIHA

BAHIR DAR, ETHIOPIA

August, 2020

DESIGNING SOFTWARE DEFECT PREDICTION MODEL USING FILTER FEATURE SELECTION AND SUPPORT VECTOR MACHINE ALGORITHMS

BY

ZELALEM FISSIHA

A Thesis Submitted to the School of Research and Graduate Studies of Bahir Dar Institute of Technology, BDU in Partial Fulfillment for the Degree of Master of Science in Software Engineering in the Faculty of Computing

ADVISOR: DR. MEKUANINT AGEGNEHU (PHD)

BAHIRDAR, ETHIOPIA

August, 2020

DECLARATION

I, the undersigned, declare that the thesis comprises my own work. In compliance with internationally accepted practices, I have acknowledged and refereed all materials used in this work. I understand that non-adherence to the principles of academic honesty and integrity, misrepresentation/ fabrication of any idea/data/fact/source will constitute sufficient ground for disciplinary action by the University and can also evoke penal action from the sources which have not been properly cited or acknowledged.

Name of the student ZELALEM FISSIHA Signature [illegible]

Date of submission: 08/17/2020

Place: Bahir Dar

This thesis has been submitted for examination with my approval as a university advisor.

Advisor Name: Mekuanint A. (PhD)

Advisor's Signature: [illegible]

BAHIR DAR UNIVERSITY
BAHIR DAR INSTITUTE OF TECHNOLOGY
SCHOOL OF RESEARCH AND GRADUATE STUDIES
FACULTY OF COMPUTING
Approval of thesis for defense result

I hereby confirm that the changes required by the examiners have been carried out and incorporated in the final thesis.

Name of Student Zelalem Fissiha Signature [illegible] Date 17/ Aug/20

As members of the board of examiners, we examined this thesis entitled "Designing software defect prediction model using FFS and SVM Algorithms" by Zelalem Fissiha. We hereby certify that the thesis is accepted for fulfilling the requirements for the award of the degree of Masters of Science in Software engineering.

Board of Examiners

Name of Advisor	Signature	Date
Mekuanint (PhD)	[illegible]	17/Aug/20
Name of External examiner	Signature	Date
Ayalew Belay(PhD)	[illegible]	8/17/2020
Name of Internal Examiner	Signature	Date
Esubalew Alemneh (phD)	[illegible]	17 Aug 2020
Name of Chairperson	Signature	Date
Belete Biazen	[illegible]	17/Aug/20
Name of Chair Holder	Signature	Date
Gebeyel B [illegible]	[illegible]	17/Aug/20
Name of Faculty Dean	Signature	Date
Belete Biazen	[illegible]	17/Aug/20

ባሕር ዳር ዩኒቨርሲቲ ባሕር ዳር ቴክኖሎጂ ኢንስቲትዩት የኮምፒዩቲንግ ፋኩልቲ
Bahir Dar University Bahir Dar Institute of Technology Faculty of Computing

Faculty Stamp

ACKNOWLEDGEMENT

At the outsetand foremost, I would like to praise my almighty GOD, who favors me from beginning to end of this study, Without GOD nothing happen forever.

I would like to express my deepest gratitude to my advisor, Dr. Mekuanint Agegnehu, for his continuous support and guidance throughout the various stages of this thesis. He provided critical and useful feedback and suggestions on how to address the research problems systematically and tactfully. His punctuality, encouragement and on time contact has always inspired me and help to overcome good result and the preceding of work in time.As my supervisors, his observations, guidance and comments help me to find the right direction of the research and to move forward, and to complete the thesis.

In addition, Thanks too to Mehari Demlew and AtlawMulat for their practical assistance in the investigations carried out for this thesis. Finally, I would like to thank everybody who was important to the successful realization of a thesis.

ABSTRACT

With an increase in the size and complexity of the software, software testing is a vital activity in software engineering to measure software quality. Finding and fixing defects in software modules have a significant impact on the cost of development and maintenance of the software product. Software defect prediction (SDP) is the process of finding defective components in software prior to deliver the software product to the customer. So that the quality assurance team can effectively allocate minimum resources for testing the product by setting more effort to the defective source code. In this regard, a wide range of Machine Learning (ML) models has been developed to predict defects in software. However, those SDP models have inadequate performances due to challenges like the presence of redundant, irrelevant features, and class imbalance problem. Class imbalanced occurs with data sample from two groups, the minority group contains considerably smaller samples than the majority group. The class imbalance nature of the defect data increases the learning difficulty of the classification algorithm to train the model.The use of imbalanced data leads to off-target predictions of the minority class, but which is considered to be more important than the majority class.Thesechallenges depreciate the performance of the defect prediction model depending on the predictor's ability to tackle data frauds. In this study, we proposed a software defect prediction model that addresses class imbalance problem using Filter-Based Feature Selection (FBFS), Synthetic Minority Oversampling Techniques (SMOTE)and Support Vecctor Machine (SVM) algorithms. FBFS is used for selecting the relevant software features, SMOTE is used to produce balanced data.SVM is used for classification in which the use of Radial Basis Function (RBF) kernel function that enables the SVM classifier to maximize the optimal marginbetween the minority and the majority class.The main contribution of this study is application of FBFS and SMOTE sampling together that enables proposed model can effectively solve the binary classification faults from the minority and the majority class equally. To assess the performance of the proposed approach, we did experiments on six highly imbalanced datasets from a public NASA repository. The experimental resultsindicated that the proposed model makes an impressive improvement in SDP performancewhen compared with MAKAHAL, DNN Hybrid and EUS Adatptive related state-of art models, in which 99%, 99%, 100%, 99%, 99%, 99% accuracy is attained for KC1, MC1, JM1, PC1, PC3, PC5 datasets respectively. Thus, we conclude that the proposed model improve the performance of SDP effectively, and it provides a brand new way of dealing with the imbalanced data problem.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANN	**Artificial Neural network**
ANSI	**American National Standards Institution**
CISDP	**Class-Imbalance Software Defect Prediction**
FBFS	**Filter Based Feature Selection**
MDP	**Modular toolkit for Data Processing**
ML	**Machine Learning**
NASA	**National Aeronautics and Space Administration**
SDLC	**Software development life cycle**
SDP	**Software Defect prediction**
SMOTE	**Synthetic Minority Oversampling Technique**
SVM	**Support Vector Machine**

CHAPTER ONE

INTRODUCTION

1.1 Background

Nowadays, the software engineering community is moving on to the use of most complex artificial intelligence techniques in software development, where critical fields like a medical, airline, bankingrequire very high-quality reliable software as a failure in these systems cost people’s lives along with huge financial losses(Misha & Sarika, 2016). However, with the increase in size and complexity of the software, testing cost, and duration of traditional software testing is increasing exponentially. Due to this reason, testing these complex software system products after release is the main challenging task for software testers(Rana & Tarhan, 2018). Therefore, software engineers are shifting their intention to early detection of the defects modules in the software during software development. Earlier identification of defects in the software can help the improper allocation of resources for testing and maintenance, help the removal of software defects, yield a cost-effective and good quality of software product(Rana & Tarhan, 2018),(Menzies & Greenwald, 2007).

Software testing is aiming to detect as many defects as possible before the software product is released, plays an important role in ensuring software quality.Most of the time, large software systems be likely tohave ahigh number ofdefective software modules . Therefore, software testing is still a challengingtaskin software development practice for software developers.How to improve testing efficiency with limited testingresources to assure software quality is a great challenge to software engineer researchers. Software Defect Prediction (SDP) technique was proposed to help to allocate testing resources reasonably, determine the testing priority of different software modules. SDP is one of the most assisting activities of the software testing phase, and it increases software reliability by early identification and removal of defects.

SDP is the process of determining parts of a software system modules or files that may contain defects or not. By using the result of SDP, software experts can efficiently judge that which software modules are more likely to be defective, the possible number of defects in a module, or other information related to software defects before software testing(Shuib Basri, et al, 2019).In the area of software engineering, SDP can precisely predict the most defect-prone software components and help software engineers to develop reliable system and allocate limited resources to the systems that are most likely to contain defects in testing phases(Garcia, et al, 2012).

A software defect is an imperfection or deficiency in a work of software product where that work product does not meet its requirements or specifications and needs to be either fixed or changed. In other words, a defect is a difference (variance) between expected and actual results in the context of testing that causes a deviation of the customer requirements. Most probably, defect is an error found after the application goes into production. Software defects are also programming errors that may occur because of faults in softwaresource code, requirements, or proposal(Agarwal, Tomar, 2014). This defect affects software quality &reliability, increase maintenance costs, and efforts to resolve them. Software development teams can detect bugs by analyzing software testing results, but it is costly and time-consuming by testing entire software modules(Xuan, et al, 2019).

In fact that, developingtotallya defect-free software system is very difficult and most of the timethere are some criticalunidentified bugs or unpredictederrorsare occurred in software products whereall the strategies and methods of the software development life cycles were applied carefully. Because ofthese defective software modules, the cost of the maintenance phase of a software productmightbecome certainlyhigh for the consumers and expensive for the enterprises.According to the study(Haonan, et al, 2018), a small number of defects are caused by compilers
that produce incorrect code, many branches from errors and mistakes made by programmers in the design and coding process. These defects reduce the quality of software and increase the cost of testing. Certainly, many software development companies including Microsoft have spent a vast amount of money and effort on testing their software products before releasing them to customers (CHEN, et al, 2016).

In a real program, exhaustive testing is very expensive and not feasible for software engineerswith the given thefinite budget and employee’s resources, and many final software products still contain defects. As a result, it is essentialto identify source code instances (software metrics) that are likelyto contain defects. By focusing on fault-prone instances, identifying software components containing defects. Hence, the ability toaccurately predict the defectiveness of software components are highly desirable for theimprovement of software quality.Therefore, it is very important and critical to predicting the defectiveness of softwaremodules using defect prediction techniques, and it becomes a hot research topic in software engineering fields, which has attracted lots of attention from both academic and industrial communities of the software product.

1.2Motivation

Although a bulk of defect prediction machine learning methods have been presented,itis detected that performance of these methods severely limited with respect to lack of common feature representation and selection of a good feature selectionalgorithm in order to deal with theimbalanced nature of the software prediction dataset(Zhang Tang, et al, 2019). Specifically, the sampling methods usually need to remove or attach lots of samples to achieve the class-balanced state, which might lead to the loss of sensitive information or the addition of synthetic information. For cost-sensitive learning-based methods, how to set the cost value is a crucial problem not yet being effectively solved(Xuan, et al, 2019). In ensemble learning-based methods, how to effectively guarantee and utilize the diversity of individual classifiers have not been addressed efficiently. Due to the skewed nature of the dataset, the majority of defect prediction models based on imbalanced datasets may have led to faulty findings. On the other hand, using feature selection techniques can reduce the time and spacecomplexity for defect prediction without effecting prediction accuracy.Althougha number of various research work has been done using featureselection and class imbalance problemsindividually. Nevertheless, there is a very limited study that can be found in investigating them together, particularly in the software engineering field. Therefore, this motivates us to explore and conduct this research.

1.3 Statement of the problem

Testing the software is an important part of the software development process to ensure the correctness and reliability of the software product. But testing the whole product is not feasible, time-consuming, and requires a high amount of resources for software tester. So, earlier identification of defects in the software using defect prediction can help the improper allocation of resources for testing and maintenance. It also helps the removal of software defects, yield a cost-effective and good quality of software product. Therefore, defect prediction is a critical activity to ensure the correct functionality and reliability of the software system.In reviewing literature, it is found that various machine learning approaches have been used for building SDP models. However, it is observed that the performance of these methods is severely different and limited because SDP models are greatly shaped by the class distribution of the training data. Unfortunately, SDP is facing two major challenges class imbalance and high dimensional features of the software products(Haixiang, et al, 2017). Since software defect prediction is a task of binary classification that the possible predicted class has only two classes. Class imbalance occurs with data examples from two groups, the minority group, contains considerably smaller samples than the majority group. Majority class contains a higher number of data samples than the minority class. When a class imbalance exists within training data, learners will typically over classify the majority group.

Finally, this problem greatly affects the prediction accuracy of the model by classifying the minority class into the majority class. On the other hand, SDP models were built on various high dimensional software metrics (features), this affects the performance of the model due to the lack of feature selection techniques because imbalanced data contain non-uniform class distributions, redundant and irrelevant features.

In addition, datasets extracted from archives of Halsted & McCabe metrics usually contain more correlated information, paired wit random error and noised data.Generally, we can organize it into three main problems imposed by data with unequal class distribution, high dimensional features, and noised redundant information of the datasets, listed as follows:

- ✓ **The machine problem**: ML algorithms are built to minimize errors. Since the probability of instances belonging to the majority class is significantly higher in the imbalanced dataset, the algorithms are much more likely to classify new observations to the majority class because there is a high number of instances in the training dataset.
- ✓ **Feature selection problem**: The data features that we used to train the machine learning models to have a huge influence on the performance you can achieve. Both irrelevant and partially relevant features can negatively influence model performance.
- ✓ **The intrinsic problem:** In real life, the cost of False Positive is usually much larger than False Negative, yet ML algorithms punish both at a similar weight(Hualong, Zhao, 2013).

1.3.1 Research questions

In this work, we aim to find and report on experimental evidence to answer the following research questions, which constitute a typical set of research questions found in the related literature.

RQ1: How to develop software defect prediction model for class imbalance datasets?

RQ2: Which software features (attributes) are critical for class imbalance defect prediction?

RQ3: To what extenttheproposed model is feasible and effective for defect prediction?

1.4 Objectives of the study

1.4.1 General objective

The general objective of this research is to develop a class imbalance software defect prediction (CISDP) model using the Filter feature selection and support vector machine algorithms.

1.4.2 Specific Objectives

In order to achieve the general objective specified above, the following specific objectives are identified:

- ❖ To identify the most significant and critical software features for defect prediction in the class imbalance dataset.
- ❖ To study the learning impact of the data preprocessing approaches on imbalanced datasets usingthe sampling method.
- ❖ To investigate the effect of attribute selection for software defect prediction using class imbalance defect datasets.
- ❖ To develop a model for software defect prediction for software products.
- ❖ To evaluate the performance of the proposed defect predictionmodel.

1.5 Methodology of the study

1.5.1 Data Collection

For this thesis, we used datasets obtained from NASA MDP promise repository. The NASA MDPpromise datasets repository stores a collection of datasets that are commonly used by the software engineering research community to construct software defect predictive models. Moreover, NASA MDP datasets are collected from different projects, which are written in different programming languages like Java, C++, and C.

1.5.2 Research Design

Designing research is a conceptual building within which investigation is conducted, itcreates the blueprint for the gathering, measuring, analysis of data, and make interpretation of data. There are different types of research designs. However, in this thesis, we followed an experimental research design approach to achieve the objective of the study. The purpose of the study is, to identify orpredict whether the software module is defective or not using the software metrics defect dataset.

1.5.3 Data Preprocessing

1.5.3.1 Data Sampling

In order to address the problem of class imbalance, a number of different techniques have been studied in the literature. For this study, we used SMOTE sampling techniques as a data sampling. SMOTE is a simple, effective, data sampling technique that achieves balanceddataset by creating extra training sample data forminority groups, in which the minority class is over-sampled by creating synthetic examples rather than replicating(Cholmyong Pak, et al, 2017). SMOTE sampling decreases the learning difficulty of the prediction model by creating a new synthetic sample data of the minority class.

1.5.3.2 Feature Selection

To identify and select the relevant features of the software metrics, different kinds of literature are reviewed thoroughly in the software engineering area. In this study, we used a Filter-based feature selection technique. First, data cleaning is performed to remove the duplicate, irrelevant features, missing values, and noised defect data. Second, using theChi-square ($\chi 2$) method, calculate each individual score of the features. The high score features are selected, which is very important and have a great effect on the prediction defects in the software product, and it is the main part of data preprocessing weprocess steps.

1.5.4 Model Designing

In this thesis, to design a defect prediction model, we followed the machine learning approach. There aredifferent algorithm in machine learning approach such as aSVM, Logistic Regression (LR), artificial neural network (ANN),K-Nearest Neighbor (KNN),From those, we usedFBFS techniquefor feature selection, SMOTE data sampling technique is used for solving the class imbalance issue and it achieves balanced dataset by creatting extra training sample data for the minority group. Finally, SVM used to predict defects in the software module, which is the most popular and efficient algorithm for binary class classification problem as the class of learning algorithms using the idea of the kernel function(Phuoc Huhyn, et al, 2019).

1.5.5Development Procedure

Predicting defectsinsoftware productsis to make thepractice of machine learning techniques that provide computer systems the ability to learn from data samples without being explicitly programmed. In this thesis, Defect prediction hasthree-phase. In the first step, data cleaning and feature selection process is performed using the filter feature selection method. In the second step, the datasets are balanced using SMOTE sampling techniques. At the third step, the balanced data is given to the SVM algorithm to learn the relevant featuresis required more training times and adjusting the value of kernel function to get the maximum marginalhyperplane. Finally, SVM performs the classification task. After getting the good performance of the model, we evaluated by testing data set. Finally, the model performance measured based on the number of correctly detected software modules using the confusion matrix.

1.5.6 Evaluation

The developed system is evaluated to measure how well it supports a solution to the problem. To evaluate the system in a rational method, testing datasets were fed into the developed model. Subsequently, the model was evaluated by comparing its output against the observed data using precision, recall, and f1-score values for evaluating diagnostic accuracy. In addition, we conducted

a comparison withMAKAHAL, DNN Hybrid and EUS Adatptive related state-of art models.

1.5.7 Tools and Implementation

To develop the proposed model of software defect prediction, we used theAnacondatool, which is more user-friendly to machine learning algorithms. Itis familiar, freely available and, contains the necessary library for data processing like Imbalanced-learn and tensor flow to implement SMOTE algorithm. PyCharm environment is an exposed source delivery of pythonprogramming language for systematic computing. It is talented of consecutively on the upperof Tensor Flow.

Tensor Flow is a representative math public library and used for machinelearning applications.The popular python programming language is used to develop the system using software defect datasets, which obtained from NASA MDP Promise repository.

1.6 Scope and Limitation

The aim of this research is concerned with designing, modeling and development of a model for predicting defects or faults in software modules in class imbalance datasets.The efficiency of the modelincreasing the fitness of the defect prediction model by selecting the appropriate feature for the proposed model. The SMOTE sampling approach is used to solve the class imbalanceproblem of the defect software dataset. In this study, we used only six publically available datasets from NASA Modular toolkit for Data Processing (MDP) Promise repository, such as MC1, JM1, KC1,PC1, PC3, PC5,no other dataset. Finally, performance measures show the statistical view of an experiment's results, Analysis, together with an explanation of the produced result and measure the accuracy of the proposed model. We use a free Anaconda tool with different libraries and Jupiter notebook editor for implementation of the proposed model.

1.7 Significance of the Study

This studyhas the following key significances.

- ☞ Enabling software developers to design, implement, and deliver defect-free software product with low maintenance costs.
- ☞ Improving the reliability and quality of software products by predicting the defective software module at an early stage of the software development life cycle.
- ☞ Ensuring the quality of the software product delivered to the customers helps in gaining their confidence.

Technically, this study also has the following significances.

- ☞ We proposed a new defect prediction approache that learns from valuable software features of the software defect data.
- ☞ We investigated attribute selection usingthe Filter-Based feature selection technique to select the relevant software features according to their individual predictive capability.
- ☞ We leverage SMOTE sampling techniques which efficientlysolves binary class classification problem of software defect prediction.

- ☞ The proposed model helps to identify the main characteristics of prediction models concerning software metrics, datasets, and performance evaluation approaches.

1.8 Organization of the Thesis

This section presents an overview of the contents of the remaining chapters. The rest of this thesis

is prepared as follows. The whole thesis is organized into five chapters. The first chapter describes the introductory part of the study.

In Chapter Two, the literature reviewed the concept of prediction of defects in software products, and the approaches used for predicting defects are presented. Brief description of software defects, causes of defects, and techniques used to prevent software faults. Secondly, software metrics and machine learning classification algorithms used in the literature are analyzed in depth. In addition, we have given a detailed description of performance evaluation metrics used for imbalanced data. Finally, the gaps in the reviewed related works and the approaches to how we fill in the gaps are described.

In Chapter Three, we described the specific research methodology used in this thesis. The detailed description of the proposed system and components that compose the system: data preprocessing that includes data cleaning, feature selection, and data sampling and classification using SVM, and the purpose of each component are described in detail.

In Chapter Four, an experimental evaluation of the proposed model for the prediction of defects in software products is described in aspect. The dataset used for the study, results, and discussion of results and the implementation of the proposed model is described thoroughly. Finally, the experimental results are compared with the state-of-the-art of defect prediction models.

In Chapter Five, we summarize the major findings and contributions to practical issues in this research work. In addition, the conclusion, as well as provides possible direction for future research are discussed.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

As described in the introductory part of the thesis, the main objective of this study is to design software defect prediction model using software defect data. In this chapter, a systematic review of the literature and analysis of related works are presented. The literature on the concept of software defects and the approaches used for predicting defects in software modules are discussed. In addition, we review literature about sampling techniques, software metrics, and, model design approaches for software defect prediction and related works from the existing literature.

2.2 Software Defects

In the software engineering community, most of the time many people used the term error, mistakes, faults, bugs, failures, and defects interchangeably, but they have different meanings and represent different aspects of the software products. Defects can be introduced at different phases of the software development life cycle, and software testing is the phase where its focus is on discovering and eliminating these defects. According to IEEE standard 1044, classification for software anomalies provides common vocabulary useful meaning for these terms in this context, according to the standard(IEEE, 2010):

Error: it is the first term of software anomalies, which is a mistake, fallacy, or misinterpretation on the part of a software developer. In the group of software developers, they are software engineers, programmers, and software testers. For example, a software developer may misunderstand a design notation, or a programmer might type a variable name incorrectly which leads to an Error. Most of the time, the error is the one that is made because of the wrong login, loop, or syntax. It usually arises in software products, it leads to alteration of the functionality of the software system.

Defect: A deficiency or shortage in a work product where that software product does not encounter its requirements or conditions and wants to be repaired or replaced. In other words, the defect is the difference (variance) between expected and actual results in the context of testing that causes a deviation of the customer requirements. Most probably, it is an error found after the application goes into production. Furthermore, it refers to several troubles with the software products, with its external behavior or with its internal features (Misha Kakkar, 2016).

Figure 1 shows the interrelation between these software anomalies: Error,Bug,Fault, Defect, and Failure. Error or mistake is founded by developer or programmer whereas defect or Fault/ Bug is found by a software tester. And failure is found by customers(Aditya Krishna, et al, 2020).

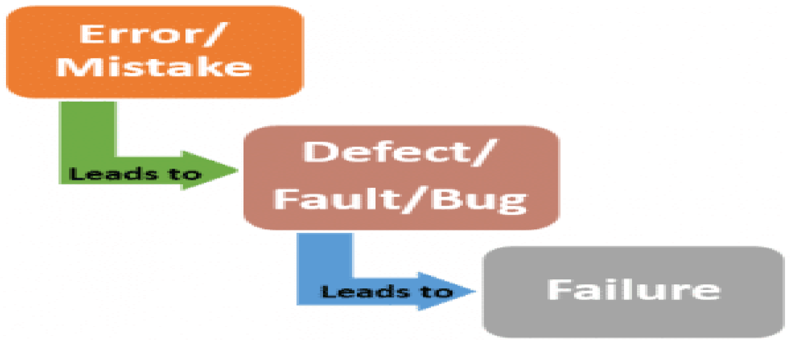


Figure 2.1 the interrelation between software anomalies

Bug:it is commonly known as a coding error. It is found usually an error found in the software development setting before the software product is distributed to the client. A bug is also an encoding error that causes the software to work unwell, yield inappropriate results, or crash, and these errors in software lead the system to fail.

Fault: An improper process or data description in a computer program which causes the program to complete in an unintentional or unexpected way. A fault is introduced into the software as the result of an error or simply it is the manifestation of an error in software products. It is an abnormality in the software product that causes it to work inaccurately, and not allowing to its requirement. It is the result of the error.

Failure: It is a situation that occurred when a software system can not performs or handles its functionality or its inability to perform the required operations within the specified period of time. And more, it is a result in which a software system or software component does not complete a required function within definite time limits. Failure is a consequence of a defect and it is the observable incorrect behavior of the system. Failure occurs when the software fails to perform in the real environment or uncertainty experienced by one or more persons, resulting from an unsatisfactory system in use or a negative situation to overcome. However, all the software industry can still not agree on the definitions for all the above.

2.3 Causes of Software Defects

Developing completely defect-free software applications & use it without testing is not practicable since the complexity of the software application increased exponentially, and then it considers that the defects can be found in the source code (Munir Ahmad, et al, 2019). Therefore, there are factors that are the initial causes of the outline of the defects in the source code of software applications.

Miscommunication of requirements: requirement mismatch is the most common problem in the software development process which causes a starter of defects in the code(Bergmane, et al, 2017). It means inaccurate & lack of communication from requirement gathering to development of the software product. When software requirements are not clear enough then the development process of the software leads to a condition where software developers facing in developing an application based on incomplete requirements and this causes the testing of an incomplete application(Bergmane, et al, 2017).

Unrealistic time schedule: develop the application in unrealistic project deadlines make an impact on the quality of the project & cause to introduce the defects in the application. Most of the time software developers cannot get enough time to design and develop software applications due to unrealistic time schedules, and they give time prior for testing to complete software applications.

Lack of designing experience: how good our design is decided on the overall software application development. In the current age of complex software development market, either implementation is complex or to implement the project more research required. Designing vague software systems is difficult to implement, and unable to complete the system in a specific time, so doing design, development & testing in a specific period time may cause errors.

Lack of coding experience: bad coding leads to errors in code which mean unhandled exceptions, errors, improper validations of inputs. Some software programmers are coding with old development tools in which they have no good faulty debuggers, compilers, validators, etc.

Last-minute changes in the requirement: requirement changes in the last minute can be dangerous which results in instability of software application. The last time changes in the requirements are very tedious to implement because changes alter the whole functionality of the application, and it will certainly bring faults, and also existing components will stop working.

Poor Software testing skills: obviously insufficient knowledge of testing skills leads to defects in the system. Moreover, in the era of agile software development, poor unit tests may result in poor coding and hence escalate the risk of errors.

2.4 Prevention of Software Defects

Defect prevention is an approach applied to the software development life cycle that identifies the root causes of defects and prevents them from repeating. This involves analyzing defects that we faced in the past and specifying checkpoints and taking specific action to prevent the occurrence of those types of similar software defects in the future (Vaseem, et al, 2020). Defect prevention is the essence of software quality management that is a serious activity in any software development process as showninfigure 2.

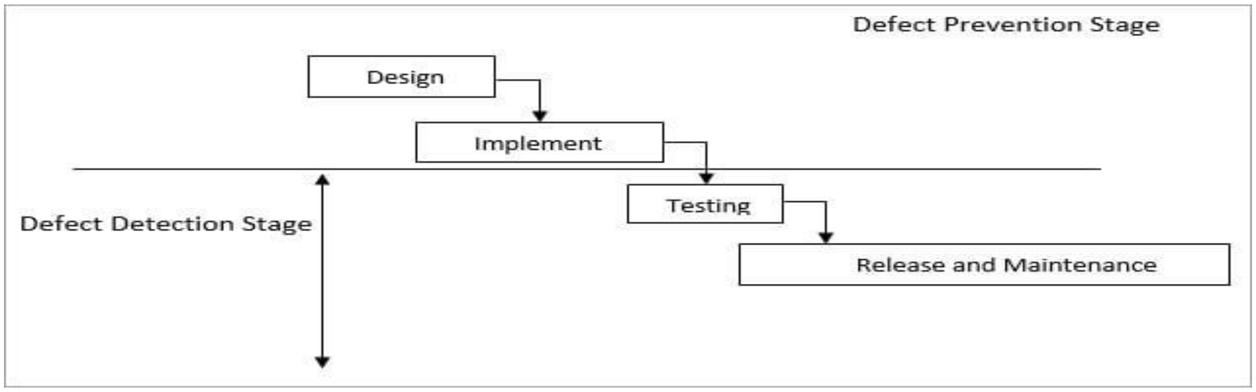


Figure 2.2Defect prevention stages

Some of the traditional approaches that have been used for software defect prevention are listed:

Review and Inspection: This method includes the review by an individual team member peer reviews and inspection of all work products.

Walkthrough: like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness or the look-and-feel of the system.

Requirement Specification Review:it is a type of a review approache that clearly prepare and review the customer requirements in a meaningful way and review it within a teamfollowed by another level of external review to make sure that all the perspectives requirements are in included.

Design Review: a stage of sorts and going through it will ensure that the QA team understands the pros and cons of each strategy, andit can be considered a feasibility study for the strategy.

Code Review:software developers performs the code inspection walkthroughs and reviews before unit and integration test the application(Jointha, 2019). Even if those traditional defect prevention approaches are applied in SDLC, it is difficult to produce a completely defect free software product. Therefore, an automated SDPtechnique is proposed in which it helps to manage the quality of the software product in a cheaper manner at the earlier stage of the SDLC.

2.5 Software Defect Prediction

The process of identifying defect-prone software components (modules) using different techniques is calledsoftware defect prediction. One of the most commonly used processes for predicting software defects is to make use of machine learning techniques that provide computer systems the ability to learn from data without being explicitly programmed using defect software datasets(CHEN, et al, 2016). Those datasets are obtained from NASA software repositories including defect tracking systems, source code changes, mail archives, data extraction, and version control systems. And more,those datasets consist of instances, which can be software components, files, classes, functions, and modules. Based on particular metrics like static code attributes extracted from the software repositories, an instance is labeled as defective or defect-free. On the collected datasets, data preprocessing methods such as noise detection and reduction, data normalization, attribute selection. Finally, thedefect prediction model is build using preprocessed NASA datasets in which the model used for predictingwhether new data or instances contain defects or not, which is a binary classification(David, et al, 2018).

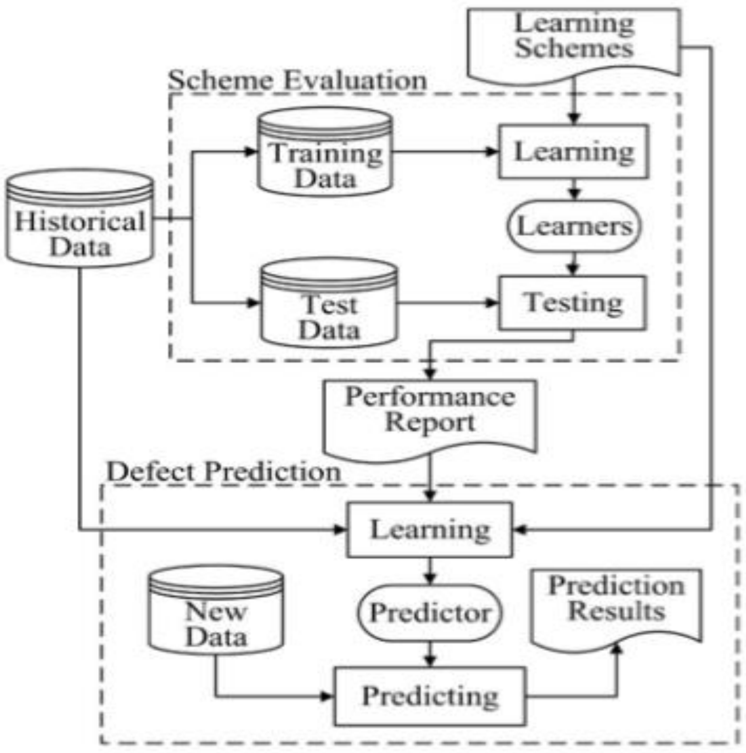


Figure 2.3 Software defect prediction process

2.6 Software Defect Prediction Techniques

SDP techniques are the most important activities of the testing phase of the software development life cycle that identifies defective and non-defective software modules. Even if defect prediction is very important in testing software products, it is not always easy to predict defects in software. SDP has the task of binary classification problems (David, et al, 2018). These binary classification problems are widened when the target prediction class has a class imbalance problem. There are different ML techniques that have been studied to address imbalanced problems extensively over the last few decades(Xuan, et al, 2019). As such, there are three types of methods for handling this problem:data-level,algorithm-level,and combine approaches (Haonan, et al, 2018).

2.6.1 Data-level Method

Data-level methods are resampling techniques used for manipulating training data to rectify the slanted class distributions, such as random over-sampling, random undersampling, and SMOTE oversampling techniques. It is a way of handling class imbalance data problems using the sampling method, and it can improve the performance of classification algorithms by changing training data before providing the data as input to the machine learning classification algorithm.

2.6.2 algorithm-level methods

It isalso called a cost-sensitive learning approach to dealing with data imbalance. It considers different misclassification costs for different classes in such a way that the data samples of minority class get importance. In(Linchang, et al, 2019) proposed a software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks (SPFCNN) method. Their results showed that the SPFCNN method contributes to higher performance compared with benchmarked prediction approaches. However, finding the cost metrics for cost-sensitive defect classifiers is another big challenge, and still, there is no systematic way of setting cost metrics.

2.6.3 Ensemble Method

In machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the individual constituent learning algorithms (Haonan Tonga, et al, 2018). Ensemble methods combine many models to get better prediction result. Three ensemble methods are widely used in SDP includes: bagging, boosting, and stacking. Applying these ensemble methods could achieve better performance than using a single classifier and it can identify fault-prone software modules using software products.

2.6.4 Defect Classification

One of the primary methods for performing software defect detection is to build a classification model of the software system on historical defect data. In the machine learning community, these classification models are known asclassifiers, usually consist of numerous attribute variables and a single class variable(Logan Perreault, et al, 2018). Practically, these models work by learning patterns between attributes of the software and a corresponding binary label that indicates whetheror not a defect exists. And thesemodels can help to better recognize the essential software features and how software features affect the defect rate of the class imbalance data(Noreen Kausar, et al, 2016). Software defect classifiers have been applied to the task of predicting defects in software projects, but the performance of the model is quite different due to the type of classifier they used. Here, we discussed the standard commonly used machine learning algorithms that are used for defect classification.

2.6.4.1 Naïve Bayes

Naive Bayes (NB) is the most straightforward and fast classification algorithm, which is suitable for a large amount of data. NB classifier is a classifier constructed based on the Bayesian network principle, which works by determining the posterior probability of every class of variables inputs(Kumar Pandey, et al, 2018). Consider the classification problem where sample x belongs to one of two classes, denoted as C1 and C2. Let priori probabilities P (C1), and P (C2) are known.The density function, P(Ci|x), is obtained by:

P(Ci|x) = $P(x|Ci)P(Ci)\ P(x)$ (2.1)

NB classifier uses Bayes theorem of probability for prediction of unknown class. According to Bayes theory, the probability of the classification error can be minimized by the following rule:

x is classified to C1, if P(C1|x) > P(C2|x)

x is classified to C2, if P(C2|x) > P(C1|x) (2.2)

Naïve Bayes assumes that the attribute values are conditionally independent to one another. It ignores the possible dependencies among the inputs. It is also successfully used in various applications such as spam filtering, text classification, sentiment analysisand software defect prediction(Rashid Ibrahim, et al, 2017) .

2.6.4.2 Decision Tree

Decision tree (DT) is one of the popular classification algorithms to understand and interpret for the detection of patterns.DT algorithm is a flowchart-like tree structure where an inner node denotes features, the branch represents a decision rule, and each leaf node represents the outcome of the prediction. It learns to classify the data based on the attribute value. It partitions the tree in a recursive manner call recursive partitioning(Cholmyong Pak, et al, 2017). The DT classifier uses comparisons to divide different instances of a set into appropriate classes. The system classifies each instance of the set, associates each class with the attributes of each instance and learns to what class each instance belongs, it is able to classify instances of a previously unseen dataset(Haonan Tonga, et al, 2018). It is also a non-parametric method, which does not depend upon probability distribution assumptions, which can handle high dimensional data.It is easy to build and implementDT classifier using the new Python Scikit-learn package.

2.6.4.3 Random Forests

Random forests (RF) is a supervised learning algorithm that can be used for both classification and regression problems. RF creates decision trees on randomly selected data samples, gets a prediction from each tree and selects the best solution by means of voting. It also delivers anattractive good pointer of the feature significance. This makes, RF is the most flexible and easy to use machine learning algorithm. A study in (Ibrahim Rashid, et al, 2017), proposed an approach for the SDP purpose, it employs two existed algorithms to have a high performance, whichis the Bat-based search Algorithm (BA) for the feature selection process, and Random Forest algorithm for the defect prediction purpose. And proved that the efficiency of theSDP model is higher with Random Forest classifier unlike other. As shown in the diagram below, RF is working with the following four steps:

- ✓ **First Step:**start with the selection of random samples from a given dataset.
- ✓ **Second Step:** Creating a decision tree for every sample of the dataset. Then it getting the defect prediction result from every sample of the decision tree.
- ✓ **Third step:** In this step, voting is performed for every predicted result of the decision.
- ✓ **Fourth Step:** Finally, selecting the most voted prediction result among the samples as the final defect prediction result.

The following diagram illustrate how random forest algorithms are working.

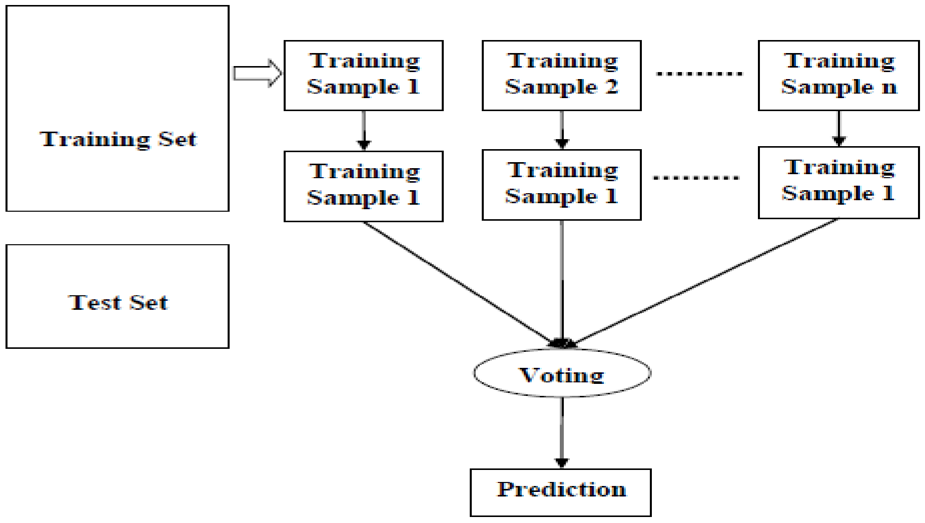


Figure 2.4 Random Forest Algorithm

2.6.4.4 Logistic Regression

Logistic Regression (LR): is one of the most popular machine learning algorithms which is used for binary (two-class) classification problems. LR is a predictive analysis algorithm and based on the concept of probability, which uses a more complex cost function. It describes and estimates the relationship between one dependent binary variable and independent variables(Rana & Tarhan, 2018).LR classification technique works in which one dependent variable can assume only one of two possible cases.

Let Y be a binary outcome and X be a predictor, then the modeling p***(x)=(Y=1|X=x),*** the probability of a success for predictor value of X=x. Then the logistic regression model can be defined as $\boldsymbol{P(X) = Log\left(\frac{PX}{1-PX}\right) = \beta + \beta 1X.}$ ***(2.3)***

LR works with odds rather than proportions where odds are simply the ratio of the proportions for the two possible outcomes. If $\boldsymbol{p}$ is the proportion for one outcome, then $\boldsymbol{1-p}$is the proportion for the second outcome(Kalaivani, 2018). Let $\boldsymbol{p}$ be the probability of success. Recall that

$Where\left(\frac{\mathrm{PX}}{1-\mathrm{PX}}\right)$ is called theOdds of Sucess.

Log $\left(\frac{\mathrm{PX}}{1-\mathrm{PX}}\right)$ is called the log Odds of Sucess

The probability for the occurrence of each case is defined by the logistical regression equation: the logistic function to model p(X) that gives outputs between 0 and 1 for all values of X:The logistic (sigmoid) function gives an **S**-shaped curve that can take any real-valued number and map it into a value between 0 and 1(Hualong, Zhao, 2013). When the curve goes to positive infinity, y predicted will become 1, and if the curve goes to negative infinity, y predicted will become 0.

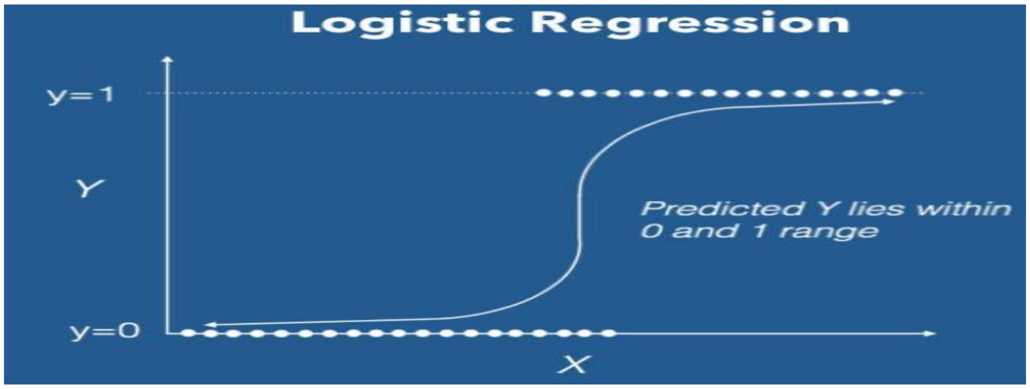


Figure 2.5 Logistic Regression

2.6.4.5 Support Vector Machine

Support Vector Machine **(**SVM)is a type of supervised machine learning algorithm that provides an analysis of data for classification and regression analysis. Moreover, SVMs are the most popular algorithm used for binary classification problems in different areas like face detection, handwriting recognition, bioinformatics, and software testing(Munir Ahmad, et al, 2019). SVM is looking for a hyperplane in a high-dimensional space as a separating plane for two aspects in order to ensure a minimum error rate. The value of each feature is also the value of the specified coordinate. Then, it finds the ideal hyperplane that differentiates between the two classes. These support vectors are the coordinate representations of individual observation.

Building Block of SVM

There are basic components that are used to build SVM classifiers(Haonan, et al, 2018) as shown in figure 6.

- ❖ **Support Vectors**: features or data point that are closest to the hyperplane is called support vectors. Separating lines will be defined with the help of these data points (separating the two class's data points).
- ❖ **Hyper plane**: As we can see in the diagram, it is a decision plane or space which is divided between a set of objects having different classes.

❖ **Margin:** It is defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors.

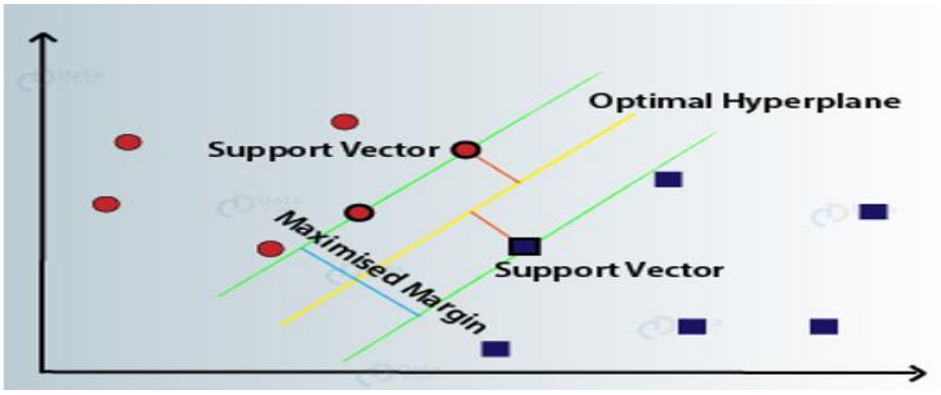


Figure 2.6 Components of SVM Algorithm

When working with SVM, there are two common steps. In the first step, we have to find the data points that lie closest to both the classes. These points are known as support vectors. In the next step, we find the proximity between our dividing plane and the support vectors. The distance between the points and the dividing line is known as margin. The SVM model tries to enlarge the distance between the two classes by creating a decision boundary(Cholmyong Pak, et al, 2017).

The basic idea of SVM is to identify a similarity distance between two classes by considering a distance metric between themSVM separates a set of input feature vectors into two classes with anoptimal separating hyperplanewhich has high prediction capability(CHEN, et al, 2016). SVMproduces the pattern classifier by applying a variety of kernel functions such as linear, polynomialfunctions as the possible sets of approximating functions.Generally, there are three different types of SVM classifiers: linear maximal margin, linear soft margin,and nonlinear classifier. These classifiers are recognizing objectsdepending upon the type of input features use.

SVM classifier useskernel functions, such as polynomial functions are used to transform the input space to a feature space of higher dimensionality when the input vectors cannot be linearly separated in the input space.SVM with different kernel functions can transform a nonlinear separable problem into a linearly separable problem by projecting data into the feature space and then finding the optimal separate hyperplane(Xiao-Xiao Niu, Ching Suen, 2012).When classes are not linearly separable, map them to high dimensional space to linearly separate.

2.7 Software Metrics

Software metrics can be considered as a quantitative measurement that assigns symbols or numbers to features of the predicted instance(David, et al, 2018). In fact, they are features, attributes that describe many properties such as reliability, effort, complexity, and quality of software products. Moreover, software metrics can be defined as a measure of some property of a piece of software that can be used for defect prediction to ensure the quality of software. These metrics play a key role in building an effective software defect predictor (Liang & Yang, 2011). Therefore, in order to predict, evaluate defectiveness of a software system, or to measure the wealth of a software system, we need to be able to measure software metrics. According to the (Daskalantonakis, 1992), software metrics can be categorized as product metrics, process metrics, and project metrics, which discussed so far these metrics in detail below.

2.7.1 Product Metrics

Product metrics are called static code attributes introduced by McCabe (1976) and Halstead (1977)(Hongyu & Zhang, 2007). It describes the characteristics of the software product such as size, complexity, design features, performance, and quality level. These metric values are directly extracted or calculated from the source code and give an idea about the complexity and the size of the source code of the software. A study in(Hongyu & Zhang, 2007), is classifying static code metrics as a line of code (LOC) metrics, McCabe metrics, Halstead metrics in general. It iswidely used metrics and easy to use metrics have been applied for creating defect predictors. McCabe attributes are cyclomatic metrics representing the complexity of a software product.

Line of Code Metrics

The Line of code metricare directly related to the number of source code lines. These metrics are:

Loc_total: Number of lines in the source code.

Loc_blank: Number of blank lines in the source code.

Loc_code and comment: Number of source code lines and comment lines.

Loc_comments: Number of lines of comment in the source code.

Loc_executable: Number of lines of executable source.

McCabe Metrics

McCabe metrics used to measure the complexity of the source code. McCabes's main argument is that loops and branches make the source code more complex. The attributes proposed based on the assumption that the complexity of pathways between module symbols is more insightful than just a count of the symbols. The following three complexity attributes introduced by McCabe (1976).

- ✓ *Cyclomatic complexity*: *v (G)* represents the number of linearlyindependent paths through the flow chart of the node.
- ✓ *Essential complexity*: *EV (G*), measures the degree to which a flowchart is able to reduce by decomposing all the sub flow charts that are proper one entry one-exit.
- ✓ *Design complexity*: *IV (G*), represents the cyclomatic complexity of a reduced flow chart of a class or module.

Halstead Metrics

Halstead complexity metrics are selected based on the reading complexity of source code. Halstead attributes were defined four key attributes and derives 6 drivenmetrics from the key attributes. Ten Halsted metrics that describe the complexity of the software from the source code which are all listedbelow. These attributes areLength(L), volume(V), Difficulty(D), Content(C), level(L), effort(E), Number of operator (Num_oper), Number of operands (Num_Opend), Number of unique operator (Num_Uni_Oper), Number of unique operands (Num_Uni_Opend)(Hongyu & Zhang, 2007).

2.7.2 Process metrics

It is the second type of software metrics used to measure the software development process, such as overall development time, type of methodology used, or the average level of experience of the programming staff(Hongyu & Zhang, 2007). Along with processmetrics are used together with source code metrics, the prediction performance couldbe improved significantly.

2.7.3 Project Metrics

Project metrics are directly related to project quality. These metrics are used to measure defects, cost, schedule, efficiency and assessment of various project funds and deliverables. In addition, project metrics describe the project characteristics and execution. Which include the number of software developers, the staffing pattern over SDLC, cost, schedule, and productivity.As we have seen in the literature of software metrics, static codemetrics are used more frequently than other software metrics (Munir Ahmad, et al, 2019).

2.8Preprocessing

Data preprocessing techniques are important and widely used in machine learning and data mining, which are the foundation of most software defect prediction studies(Misha & Sarika, 2016). There are many factors negatively affecting the performance of defect prediction models such as redundant, duplicate, and irrelevant information or noisy data. These problems can be solved by using data preprocessing techniques including data cleaning, attribute selection, and data sampling, even if differences in selecting models and software metrics among different studies are there.

2.8.1 Data cleaning

Since NASA dataset metric values are directly extracted or calculated from the source code of the software using McCabe and halted metrics, that contains duplicate, redundant and irrelevant features that negatively affect the performance of the classification model. Therefore, data cleaning alleviates thesechallenges when we apply before the defect prediction model is built.

Duplicate values:when two or more software features have similar value for all instances then those attributes are said to contain identical values. Therefore, only one of them can be preserved and the remaining can be removed as redundant data in which they depreciate the performance of the prediction model.

Missing Values: The attributes for which at least one instance value is not present are known as attributes with missing values. In (Misha & Sarika, 2016), it was mentioned that missing values in data sets can occur due to division by zero error. The possible solution is either to dropping (remove) all instances which contain missing values or replace the missing values by zero.

Constant Values: It refers to those attributes in which every instance has the same value. Since such attributes contribute no information to the data, they can be deleted.

2.8.2 Feature selection

Feature selection is the process of identifying and selecting the most relevant features (attributes) of the software metrics to build a robust prediction model. The feature selection method selects a subset of features that are used as independent variables in the prediction model. It was found that features selection methods produce the subset of features that can be used for creation of a model without affecting the classification quality of the prediction model(Shuib Basri, et al, 2019). Nowadays,there is an increase in size and complexity of software's, an accurate prediction of defect is a crucial issue based on several attributes.

Therefore, instead of considering all the features, it would be more useful to select the best features which are relevant and significant for defect prediction in any software module(Taghi, Kehan, 2009). Although feature selection has been widely applied in various application domains like image processing, disease detection for many years. However, the application of feature selection techniques in software testingfor software defect and reliability prediction domain is very limited.In software engineering area, there are two common feature selection methods: wrapper-based feature selection and filter-based feature selection methods.

The wrapper-based feature selectionapproach involves training learner algorithms during the feature selection process. This works by evaluating a subset of features machine learning algorithm that employs a searching strategy to look through the space of possible feature subsets, and evaluating each subset based on the quality of the performance of a given classification algorithms. wrapper method is also known as a greedy searching algorithm is because it aims to find the best possible combination of features that results in the best prediction model. For a given data set, a wrapper-based technique may produce different feature subsets when using different learners. these problems of a wrapper-based technique lie in its high computational cost and risk of overfitting to the model(Prasanth, et al, 2017).

Filter-Based Feature Selection (FBFS): this method uses the computational characteristics of datasets to independently assess and rank attributes in datasets which are found to be independent of the prediction model(Shuib Basri, et al, 2019).The filter-based approach practices the intrinsic features of the data based on a given metric for feature selection and does not depend on the training of the learner algorithm(Satria Wahono, et al, 2014). Features are selected on the basis of their individual score in the various statistical tests for their correlation with the outcome variable. Linear discriminant analysis is used to find a linear combination of features that characterizes or separates two or more classes of a categorical variable. In this regard, Chi-square ($\chi 2$) is the commonly used FBFS statistical test method. $\chi 2$ test is used to examine the distribution of the class as it relates to the values of the given feature(Agarwal, Tomar, 2014).Filtering methods have fast computational time, and it very good for eliminating irrelevant, redundant, constant, duplicated, and correlated features. Since most class imbalance learning methods require careful parameter settings to train the predictive model, and feature selection to control the strength of emphasizing the minority class before learning.

2.8.3Data Sampling

Data sampling is a data resampling techniques, manipulating training data to rectify the skewed class distributions of the datasets. These techniques modify the training distributions in order to decrease the level of imbalance or reduce noise like mislabeled samples or anomalies. Data resampling techniques are planned to add or remove samples from the training dataset in order to make equal class distribution among the two classes. After getting the balanced dataset, the standard ML classification algorithms can be fit successfully on the transformed datasets. There are three types of data sampling techniques used in machine learning. These are over-sampling, undersampling, and SMOTE sampling techniques.

a) Under-sampling

Under-sampling is a common type of data sampling used in the machine learning community. it can be defined as removing some observations of the majority class that results in having the same number of examples in each class, and it creates balanced data. The only used undersampling method is random under-sampling (RUS). In RUS method, the majority class instances are discarded at random until a more balanced distribution is reached. This could lead to poor generalization to the test set because most sensitive information will remove randomly.In (Haixiang, et al, 2017), proposed a new model that analyzes the effects of over and under sampling on fault-prone module detection. They conclude that even if RUS is effective to balance datawhen we are removing information randomly that may be valuable and sensitive. Moreover,repeated sampling often leads to severe underfittingand produce more noised data sample problems.

b) Oversampling

Over-samplingis another data sampling techniqueswidely used in machine learning for class imbalance data. Over-samplingmethods attempt to balance data either by replicating the minority class or by generating new synthetic samples of the minority class. The elementary version of over sampling is called random oversampling, which simply duplicates randomly selected features from the minority class.In(Cholmyong Pak, et al, 2017), propose a new software defect prediction model using oversampling techniques. They introduced the effectiveness of over-sampling on imbalanced data. They conclude that oversampling is effective, and it is operational in numerous classifiers.However, both random-oversampling and random-under sampling techniques are vulnerable tooverfitting and losing sensitive information problems respectively.

2.8.4 SMOTE Sampling

Synthetic Minority Over-sampling Technique (SMOTE) is a special and more advanced sampling method that aims to overcome the drawbackof random oversampling techniques where new instances are created by combining features of the target instance and its nearest neighbors(Cholmyong Pak, et al, 2017). This sampling method generates artificial minority class instances from existing ones, instead of duplicating existing instances, and it works in the feature space rather than the data space. SMOTE is an effective data sampling technique that achievesa balanceddataset by creating extra training sample data forminority group, in which the minority class is over-sampled by creating synthetic examples rather than replicating(Cholmyong Pak, et al, 2017).

Another study in(Bushra Hamid, et al, 2015) proposed an approache called using SMOTE for convalescing software defect prediction. The result showed that the probability of prediction would be affected a little when using balanced datasets. While in class imbalanced data, SMOTE effectively improves the performance of the defect prediction model than from the previous study. However, SMOTE alone cannot completely solve the class-imbalance problem in software defect prediction, which needs the data preprocessing, while performing sampling techniques irrelevant feature are also oversample which needs another feature selecton method.
Despite the success of oversampling, still, only a handful of techniques are available in open-source software. Now, the current version of the imbalanced-learn Python package implementsdifferent oversampling techniques, this is the first public,open source implementation available for the SMOTE algorithm.

SMOTE is an effective data sampling technique that achieves a balanced dataset since it works based on nearest neighbors judged by euclidean distance between instances in the feature space of the data samples of the minority class.To create the new artificial minority class instance, SMOTE randomly selects one existing minority class sample m Next, the algorithm should find its k-nearest neighbors and should select at random one of the k samples, called n. Subsequently, it is necessary to calculate the difference between samples m and n and then multiply this with a random number between 0 and 1.

After the resulting value is added to the feature vector of m, m and n form a line segment in the feature space (Haonan Tonga, et al, 2018). The pseudo-Code of the SMOTE algorithm is presented in the following section.

Algorithm *SMOTE*(T,N,K, (Tian, 2018))

Input: Number of minority class sample T; Amount of *SMOTE* N%*; Number of nearest neighbors Output(N/100)*T synthetic minority class samples*

```
1.  If (N <100%, randomize the minority class samples as only a random percent of them will be
    SMOTED )
2.  If  N < 100
3.   Then Randomize the T minority class samples
4.       T = (N/100)*T
5.        N = 100
6.  Endif
7.    N = (int)()N/100) (the amount of SMOTE is assumed to be integral multiples of 100)
8.  K = Number of nearest neighbors
9.  Numattrs = Number of attributes
10.  Sample [][]:array for original minority class samples
11. Newindex: keeps a count of  number of synthetic sample generated, initialized to 0.
12.  Synthetic [][]: array for synthetic samples.  (compute K nearest neighbors for each
    minority class samples only  )
13.  for i ← 1 to T
14. compute K nearest neighbors for i, and save the index in the nnarray
15.               Populate (N, i, nnarray)   // function to generate synthetic samples
16. endfor
17.  while N ≠ 0
18. Coose a random number between 1 and K. call it nn. This step chooses one of the K nearest
    neighbors of i
19.              for attr← 1 to numattrs
20. Compute:dif = sample [nnarray] [nn][attr]- sample[i]
```

```
21. Compute:ga p Random number between 0 and 1.
     Synthetic [newindex][attr]=sample[i][attr] + gap dif
22. dif
23. endfor
24. newendex ++
25.   N = N – 1
26. endwhile
27.    return
28. End of Pseudo-Code
```

2.9 Performance Evaluation Metrics

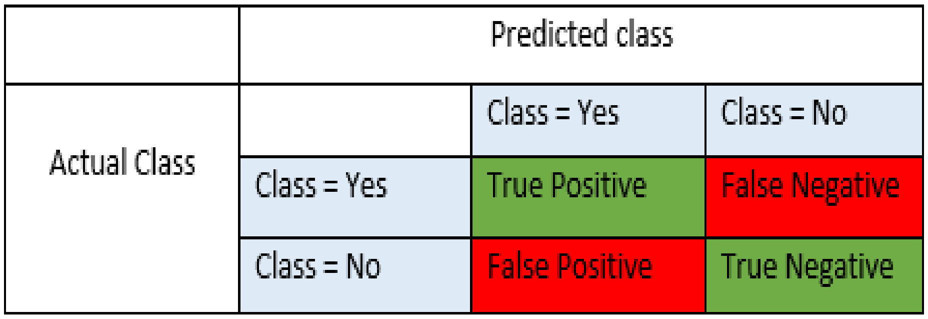
In order to measure the performance of the proposed model, we used the confusion matrix and Precision-Recallcurve, which is commonly used for a binary classification problem. It is a tabular format that is used to describe the performance of classification of the model or classifier on the givenset of test data for which the true values are well-known.Accuracy, Precision, Recall, and F1-score, these are widely used evolution metrics in software defect prediction for class imbalance, (Ahmed, Umair, 2019). And they accurately report how accurately learn the machine.

2.9.1 Confusion Matrix

A confusion matrix of binary classification is a two by two matrix formed by counting the number of four outcomes of a binary ML classifier as shown in table 8. Confusion Matrix is needed for finding Accuracy, Precision, Recall, and F1-score, which represents in the following section: from these four criterions, four evaluation metrics have been calculated. As showed in figure 8, the performance is analyzed and evaluated through various measures generated from confusion matrix. A confusion matrix consists of the following four parameters:

- ✓ **True Positive (TP)** – An instance that is positive and is classified correctly as positiveinstances. e.g., classified as defective data which is in fact defective module.
- ✓ **True Negative (TN)** – An instance that is negative and is classified correctly as negative instances.I.e. classified as defect free data which is in fact defective free module.
- ✓ **False Positive (FP)** – An instance that is negative but is classified wrongly as positive instances. I.e. classified as defective data which is in fact defect-free.

- ✓ **False Negative (FN)** – An instance that is positive but is classified incorrectly as negative instances. I.e. classifying as fact defect-free data which is in fact defective module.



		Predicted class	
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Figure 2.7 Confusion Matrix

Accuracy: is the most widely used spontaneous performance measurement, and it is simply a ratio of correctly predicted samples to the total samples.It delivers the best result if the cost of false positives and false negatives are similar. It measures the proportion of the files classified correctly, to the total number of files.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (2.6)$$

Precision: Precision measures the proportion of files that were correctly classified as faulty over the total number of files classified as either faulty or non-faulty. In other words, precision or Confidence denotes the proportion of Predicted cases that are indeed real faulty files(Menzies & Greenwald, 2007). This is a measure of how good a prediction model is at identifying actual faulty files. It talks about how accurate your model is out of those predicted positive, how many of them are actually positive.

$$Precision = \frac{TP}{TP+FP} \quad (2.7)$$

Recall: Recall measures the proportion of faulty files that are correctly identified as faulty over the total number of faulty files available. Recall or Sensitivity is the proportion of real faulty files that are correctly predicted as faulty files.

$$Recall = \frac{TP}{TP+FN} \quad (2.8)$$

F1-Score: F1Score is computed by taking the weighted harmonic average of precision and recall as shown in the equation. F1 is usually more useful than accuracy, especially for uneven class distribution. If the cost of false positives, and false negatives are very different, it's better to look at both Precision and Recall.

$$F1Score = \frac{2*Precision*Recall}{Precision+Recall} \quad (2.9)$$

Macro-average precision or recall: It is the average of the precision and recall (respectively) ofthe model on the majority and minority classes.

Macro-average precision = (P1 + P2 + … + P_N) / N (2.10)

Macro-average recall = (R1 + R2 + … + R_N) / N (2.11)

Weighted-average precision or recall is calculated by finding the average of sum up the individual true positives, falsepositives and false negatives for each class.

Micro-average precision = TP1 + TP2 + … + TPN (TP1 + TP2 + … + TPN) +(FP1 + FP2 + … + FPN) (2.12)

Micro-average recall = (TP1 + TP2 +⋯+TPN) (TP1 + TP2+⋯+TPN)+(TN1 + TN2 + ... + TNN) (2.13)

2.9.2 Precision-Recall Curve

Precision-Recallcurve (ROC) is another useful measure of success of prediction when the classes are very imbalanced(Tanujit , et al, 2019). ROC curve is a graphical plot used to show the diagnostic ability of binary classifiers. ROC shows the trade-off between precision and recall for the different thresholds between zero and one.A ROC curve is constructed by plotting the true positive rate against the false-positive rate.

The True Positive Rate (Sensitivity) is the proportion of observations that were correctly predicted to be positive out of all positive observations (TP/(TP + FN)). In software defect prediction, the true positive rate is the rate in which defective modules are correctly identified to test positive for the defective in question.

The False Positive Rate (FPR) is the proportion of observations that are incorrectly predicted to be positive out of all negative observations(Turabieh Hamza, et al, 2019). In other word, it can be defined as the total number of negative cases incorrectly identified as positive cases divided by the total number of negative cases (FP/(TN + FP)). Since ROC does not depend on the class distribution of the dataset, this makes it useful for evaluating defect classifiers predicting rare events class imbalance software defect prediction. In contrast, evaluating performance using accuracy would favor classifiers that always predict a negative outcome for rare events(Ahmed, Umair, 2019).

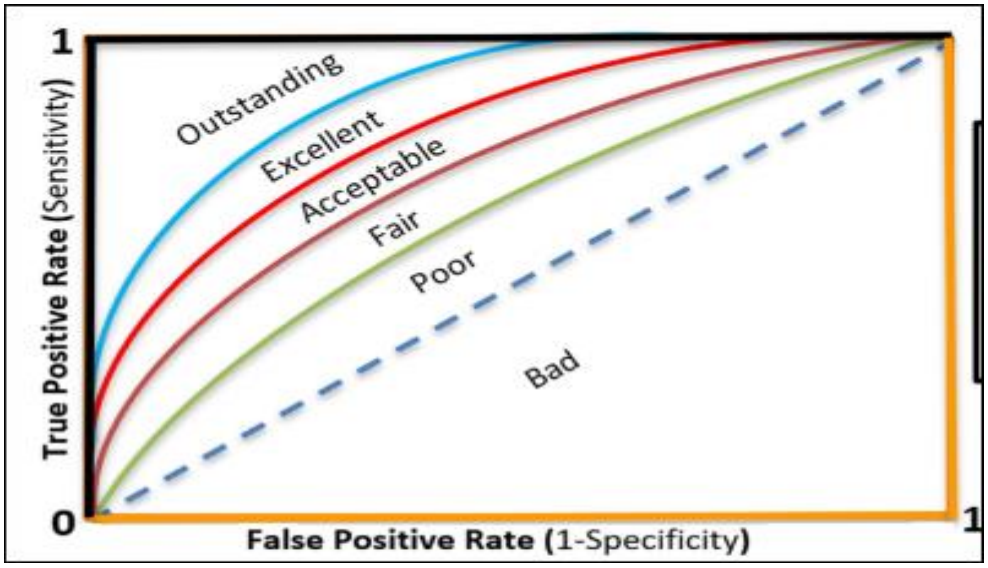


Figure 2.8 Precision-Recall curve

2.10 Related work

Software company has been trying to find the best methods for developing software products within an appropriate time and according to the requirements of the customers. A large number of defect prediction models have been proposed so far using machine learning techniques on software metric defect datasets. Only those works whose contributions are related to our work are discussed.

Though a variety of earlier studies have successfully used ML techniques for predicting and detecting defects in software products on the balanced dataset, these techniques produceinadequate results when applied on class imbalanced datasets. Therefore, in this section, we reviewed the researches that are most related with this study on imbalanced data. The use of imbalanced datasets leads to off-target predictions of the minority class, which is generally considered to be more important than the majority class. Thus, handling imbalanced data effectively is crucial for the successful development of a competent defect prediction model(Garcia, et al, 2012),(Tanujit , et al, 2019).

High-class imbalance dataset are usuallyappearein complex software, this makes the learner algorithm very difficult to identify the minority class, and this introduces a bias in favor of the majority class, (Francisco Navarro, 2011). Consequently, it becomes quite difficult for the learner to effectively discriminate between the minority and majority classes, which yields an incorrect result. Such a biased learning process could result in the classification of all instances as the majority (negative) class and produce a misleadingly high accuracy metric. In situations where the occurrence of false negatives is relatively costlier than false positives, a learner's prediction bias in favor of the majority class could have adverse consequences, (Hualong, Zhao, 2013).

Data imbalance can lead to unexpected mistakes and even serious consequences in data analysis, especially in classification tasks. This is because the skewed distribution of class instances forces the classification algorithms to be biased to the majority class. Therefore, the concepts of the minority class are not learned adequately. As a result, the standard classifiers tend to misclassify the minority samples into majority samples when the data is imbalanced, which results in quite a poor classification performance. Though data imbalance has been proved to be a serious problem, it is not addressed well in the standard classification algorithms
A study in (Khoshgoftaar, et al, 2010)present a defect prediction approaches using feature selection and random under data sampling together. To increase the learning ability of the minority class, they used RUS form the training data for building a software defect prediction model. The study was evaluated using data obtained from the NASA using WEKA tool. They achieve an average result of 84% and 88% accuracy using KNN and SVM classifiers respectively. Their empirical results show that feature selection based on sampled data performs better than feature selection based on original data. Unfortunately, random oversampling techniques duplicate the minority class which creates another overfitting problem of the prediction model.

According to(Misha & Sarika, 2016) declaration, a new framework is proposed using attribute selection for software defect prediction based on five classifiers IBk, KStar, LWL, Random Tree and Random Forest. To evaluate the performance of the proposed framework, they used a cleaned version of five NASA datasets namely MC1, JM1, KC1, KC3, and PC1 from the promise repository.They compared the performance of the five classifiers.The result shows that the proposed method improves the classification accuracy of defect prediction using selected fewer features as compared to the previous studies. However, only feature selection techniques is not solve the class imbalanced problem of software defect prediction.

In study(Rashid Ibrahim, et al, 2017)proposed a new approache for software defect prediction using theBat-based search and random forest algorithms. Bat-based search algorithm used for the feature selection process, and the random forest algorithm for the defect prediction purpose.And also differentmetaheuristic algorithms have been used to get the most effective featuresand deliver them to the RF classifiers in order to make the prediction. They evaluate the performance of the model using five types of base learners SVM, LR, DT and RF classifiers. The experiments were implemented in four NASA defect datasets.Their experimental results showed that, random forest classifier wereachieved the highest accuracy compared to other classifier. However, the performance of the proposed model was very inadequate by measuring it using accuracy because they do not consider the class imbalance problems, and the minority class is affected by prediction.

According to(Ruchika, Kamal, 2019) declaration, a new SPIDER3 method wasproposed using oversampling techniques. Furthermore, ML learners were also evaluated to ascertain their effectiveness in improving the results of the developed defect prediction models on imbalanced datasets. To evaluate the performance of the SPIDER3 model, their experiments were implemented in ten standard NASA software defect prediction datasets.In addition, they made a comparative analysis of machine leanrning learners and oversampling methods,and the proposed SPIDER3 performed better results.They conclude that oversampling improved classification and effectively handle imbalanced problem of the predictions. However, replicating data samples using oversampling leads to the creation of noised data and this needs another feature selection method to remove noised data.

In study(Changzhen, et al, 2018)introduced a defect prediction model with the help of a local tangent space alignment support vector machine (LTSA-SVM) algorithm. The model employes the SVM algorithm as the base classifier of the software defect prediction model. LTSA algorithm used to extract the intrinsic structure of the low-dimensional feature and performs dimension reduction. The SVM is trained by the reduced dimension data and verified the classification model.The experimental result shows thatthe proposed method can effectively extract sensitive features in the dataset which effectively solves data redundancy and improves the performance of the defect prediction model. However, the time cost of the model in parameter optimization is very high, and they put utilizing a more effective approach to create the a low dimension space as future work.

In study(Manjula and Florence, 2018), a new model wasdesigned Cognitive Deep Neural Networks (DNN) prediction method for software fault tendency module based on Bound Particle Swarm Optimization (BPSO). They used the DNN prediction algorithm for the software fault tendency module based on BPSO dimensionality reduction.The simulation experiments were implemented in four standard test sets: PC1, JM1, KC1, and KC3 to verify the performance of the algorithm. The study evaluated the proposed model by using the performance measure (accuracy). Finally, the study results showed that the average proposed model accuracy is 82.27%.

2.11 Summary

On reviewing literature in the related work, it is found that the latest approaches have been used for predicting defects in software products. Unfortunately, those approaches cannot be used effectively to build powerful models when previous data available is limited and imbalanced. Moreover, the classification accuracy of these studies has higher in the majority class than the minority class. Therefore, those studies can be used as baseline for our researches so that the results of the proposed model can be compared and verified with it. In addition, particularly in the software engineering field, while considerable work has been done for feature selection and class imbalance problem separately, limited research can be found on investigating them both together. Therefore, we proposed a combination of feature selection and data sampling techniques in the context of software defect prediction.

The following table show the major finding and the limitation of the most related study on defect prediction using NASA MDP datasets.

No	Author	Title	Methodology	Major finding and limitation
1	(Kakemono & Solomon, 2017)	Diversity based Oversampling Approach to Alleviate the Class Imbalance Issue in Defect Prediction	MAKAHAL ROS with five classification models (C4.5, NNET, KNN, RF, SVM using NASA datasets	MAHAKIL achieved good result to the other methods.But the repeated ROS often produce other noise data that leads sever over fitting problem.They didn't use feature selection mdthod to solve it.
2	Ibrahim Rashid, et al, 2017)	Software defect prediction using feature selection & random forest algorithm	Bat-based search Algorithm for feature selection, & random Forest algorithm for defect prediction on NASA dataset.	The result shows good classification accuracy for majority class. But the minority class was low because the instance resampling method cannot effectively solve imbalance problem due to data duplicate during sampling.

3	(Manjula and Florence, 2018)	Deep neural network based hybrid approach for software defect prediction using software metrics	Genetic algorithm for feature optimization and deep neural network for classificationon NASA datasets.	They achieved better performance than using a single classifier. But they did not sampling method for class imbalance problem and it affect the minority class issue.
4	(Zhang Tang, et al, 2019)	Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks	Cost sensitive learning for train the model Artificial neural network for prediction using NASA datasets.	Even if they achieve good accuracy finding the cost metrics for cost-sensitive defect classifiers is another big challenge, and still, there is no systematic way of setting cost metrics.
5	(Changzhen, et al, 2018)	Establishing a software defect prediction model via effective dimension reduction.	LTSA algorithm for dimension reduction. SVM algorithm for defect prediction using NASA dataset.	They effectively reduce dimension of features and Predict the defective software modules.But only dimension reduction cannot solve class imbalance issue effectively.
6	(Jinghui Chu, et al, 2017)	Learning framework for imbalanced data using Adaptive Ensemble Under sampling-Boost	Combining EUS with boosting and an adaptive boundary decision strategy using NASA dataset.	Effectively guarantee and utilize the diversity of individual classifiers have not been addressed efficiently. Removing information randomly that may be valuable and sensitive.

Table 2. 1 summary of related work

CHAPTER THREE

3. System Design

3.1 Introduction

In this chapter, a detailed description of the proposed software defect prediction system or model for prediction defects in softwareproductis discussed. It has a series of steps starting from data preprocessing, feature selection, data sampling, and learning to classification into predefined binary classes which are defective or non-defective software modules.In section 3.2, a general description of the proposed class imbalance software defect prediction system architecture is presented. In the following sections, each data preprocessing process: data cleaning, feature selection, data sampling is described thoroughly. Finally, a classification that prediction of defects in software module is discussed.

3.2 Proposed Class Imbalance Software Defect Prediction Model

The proposed system has four components. Data cleaning, feature selection, data sampling, and classification.In data cleaning, we remove the duplicate, missing, and noise data. In feature selection, we used filter feature ranking and selection techniques for selecting the most significant and appropriate software features used for defect prediction. In data sampling, we used SMOTE sampling techniques to resolve the skewed class scatterings of the dataset. This is done by altering the training data distributions of the minority class in order to decrease the level of class imbalance.For classification, we use asupport vector machinealgorithm.Classification encompasses two main constituents: training and testing phase. In the training phase, we use balanced data to learn (train) the model, and the original test data is used for testing the proposed CISDP model. Finally, a linear maximal marginal classifier is used for classifying into a specific predicted class (defective or non-defective module).

In order to learn the model adequately in the training phase, we use balanced data since class imbalance data has skewed class distribution,which leads to an off-target classification result.Unlike the training phase, the testing phase is performed using the un-sampled defect data this is because we want to test the proposed CISDP model on the original data. Lastly, the learning model is constructed from the training of data samples.

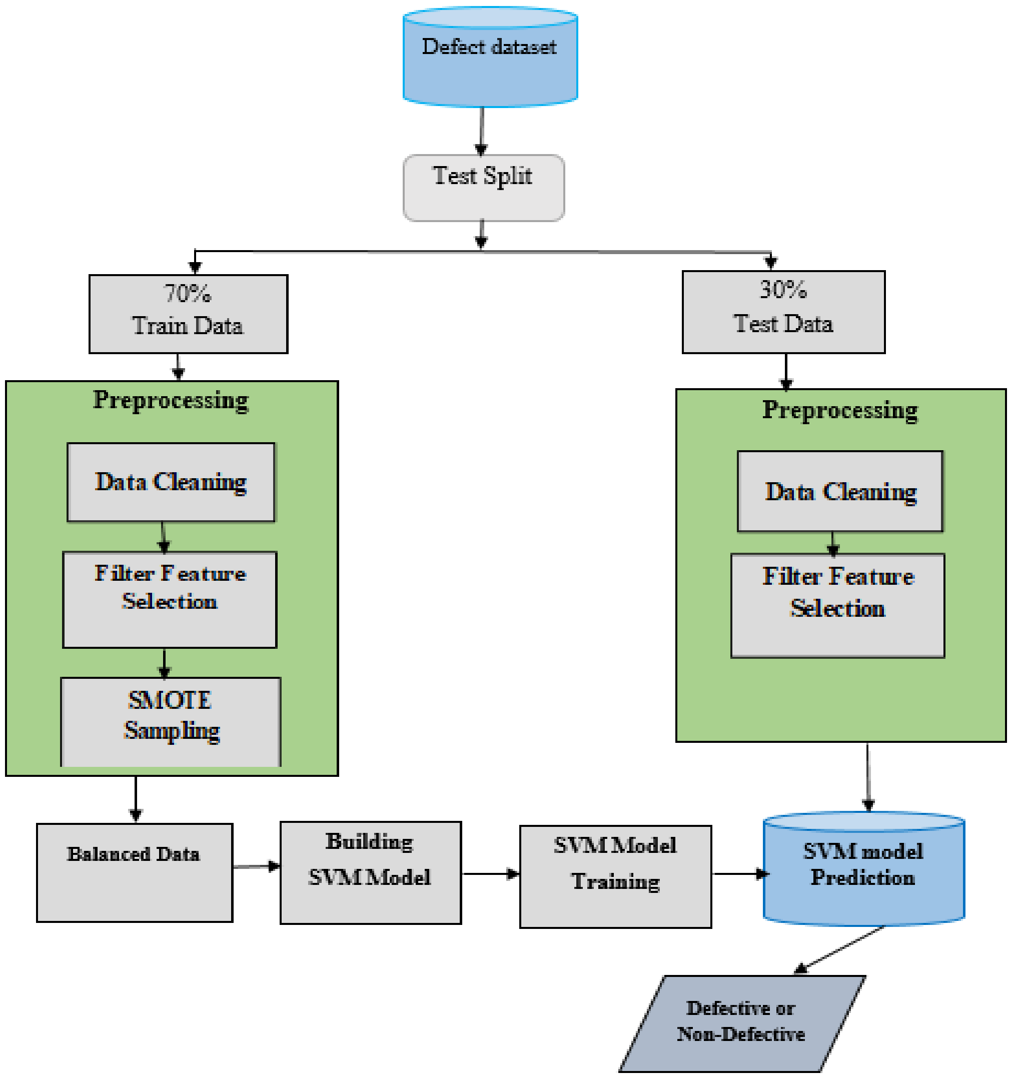


Figure 3. 1Architectureof the Proposed CISDP Model

As shown in the figure, the architectures of the software defect prediction process accept defect data, the first process of the model is data pre-processing. Software defect data are collecting from the NASA MDP repository. These datasets have numerical values since software metrics are frequently extracted from McCabe, Halstead, and other design metrics. Therefore,the proposed model accepts numerical value defect data as input. In the data preprocessing process, data cleaning, feature selection, and SMOTE samplingare performed sequentially as unnecessary features like missing values, duplicate instance and correlate attributes are removed in this process.

3.3 Preprocessing

Preprocessing phase involves an identification and removal of duplicate, irrelevant information or noisy data that negatively affecting the performance of the proposed defect prediction model, and selection of significant features to build a robust prediction model.

3.3.1 Data Cleaning

Data cleaning is the first step of data preprocessing, which is used to remove duplicate missing and irrelevance features values from defect datasets. It deletes attributes those in which every instance has the same value since such attributes contribute no information to the data. In data cleaning, we use the filtering method implemented sklearnpackagein python library.In the preprocessing step, first NumPy, pandas, and sklearnpageges are imported. Then numerical dataset is load using pandas. Next, we import the train test split and classify the dataset into training and testing data. Then calculate the constant filter threshold, and the constant features are removed by constant filter threshold.Similarly, the datasets which contain instances with missing values are discarded all instances which contain missing values.In addition, the datasets which contain null values, which means instances have no desired class are removed from the dataset using the drop null method.

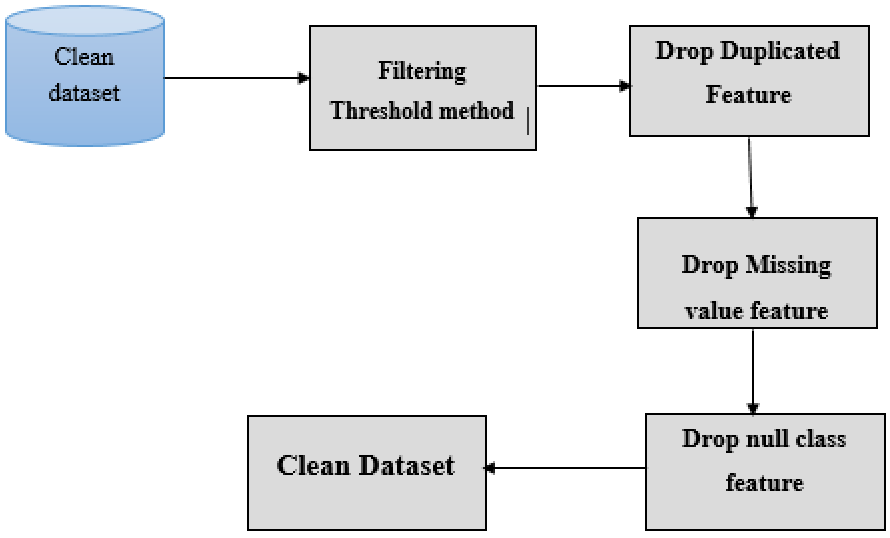
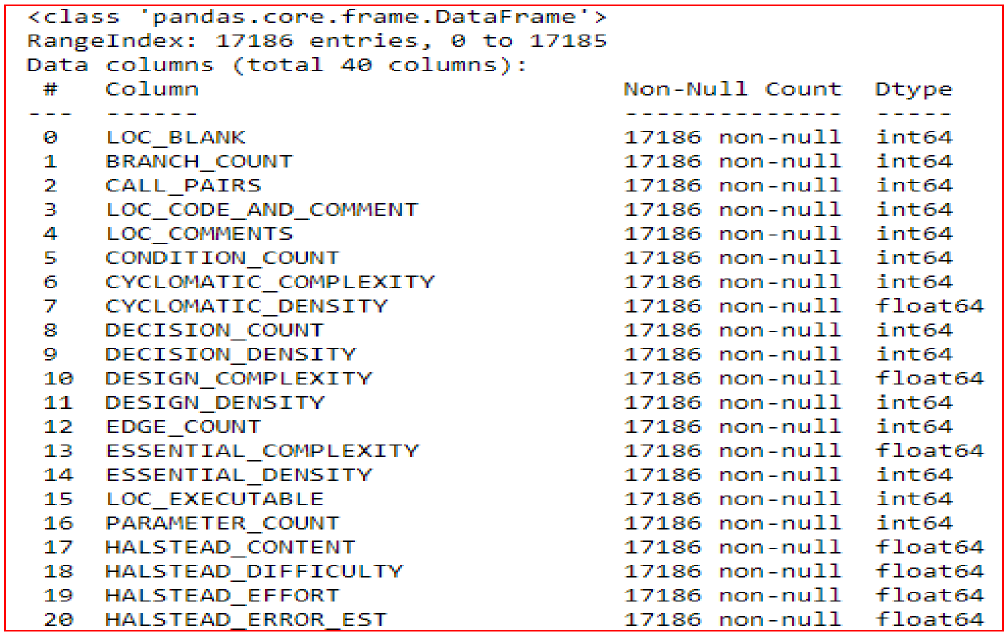
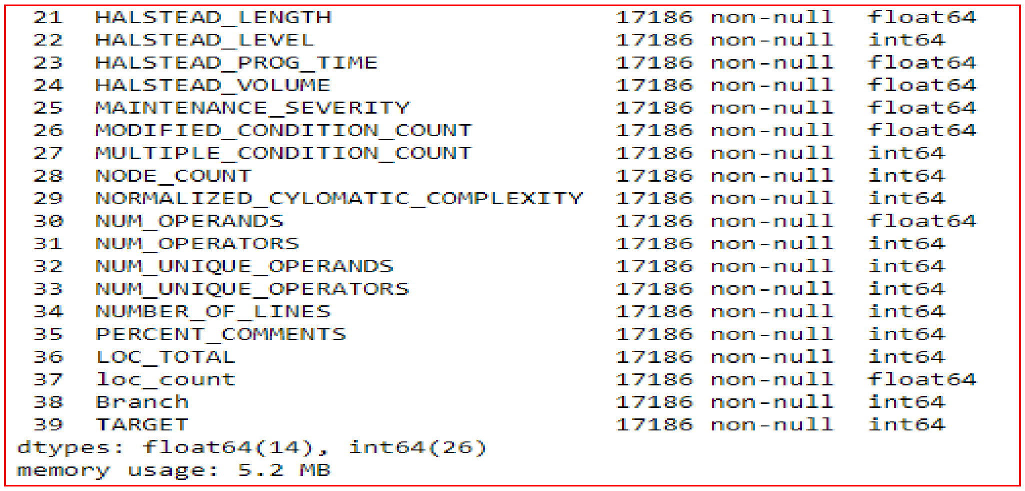


Figure 3. 2Data cleaning process

In data cleaning, we applying the filter method to sparkling the defect data since software defect data is often extracted from logfiles, version controls and bug reports in bug tracking databases automatically by usingtools, these data contains missing values, duplicate instance and correlate attributes (David, et al, 2018).The software defect dataset have 21 and 39 features. Therefore, we used sklearn feature selection in the python library. Thus, constant and irrelevant features are selected using the filter threshold method.Finally, we removed all these instances in all the six datasets. After data cleaning process of data preprocessing, we get all the software feaureswith no missing, duplicate value, and non-null 1109 instances in PC1 dataset as shown in the following figure 12.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17186 entries, 0 to 17185
Data columns (total 40 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   LOC_BLANK                            17186 non-null  int64
 1   BRANCH_COUNT                         17186 non-null  int64
 2   CALL_PAIRS                           17186 non-null  int64
 3   LOC_CODE_AND_COMMENT                 17186 non-null  int64
 4   LOC_COMMENTS                         17186 non-null  int64
 5   CONDITION_COUNT                      17186 non-null  int64
 6   CYCLOMATIC_COMPLEXITY                17186 non-null  int64
 7   CYCLOMATIC_DENSITY                   17186 non-null  float64
 8   DECISION_COUNT                       17186 non-null  int64
 9   DECISION_DENSITY                     17186 non-null  int64
 10  DESIGN_COMPLEXITY                    17186 non-null  float64
 11  DESIGN_DENSITY                       17186 non-null  int64
 12  EDGE_COUNT                           17186 non-null  int64
 13  ESSENTIAL_COMPLEXITY                 17186 non-null  float64
 14  ESSENTIAL_DENSITY                    17186 non-null  int64
 15  LOC_EXECUTABLE                       17186 non-null  int64
 16  PARAMETER_COUNT                      17186 non-null  int64
 17  HALSTEAD_CONTENT                     17186 non-null  float64
 18  HALSTEAD_DIFFICULTY                  17186 non-null  float64
 19  HALSTEAD_EFFORT                      17186 non-null  float64
 20  HALSTEAD_ERROR_EST                   17186 non-null  float64
```



```
 21  HALSTEAD_LENGTH                      17186 non-null  float64
 22  HALSTEAD_LEVEL                       17186 non-null  int64
 23  HALSTEAD_PROG_TIME                   17186 non-null  float64
 24  HALSTEAD_VOLUME                      17186 non-null  float64
 25  MAINTENANCE_SEVERITY                 17186 non-null  float64
 26  MODIFIED_CONDITION_COUNT             17186 non-null  float64
 27  MULTIPLE_CONDITION_COUNT             17186 non-null  int64
 28  NODE_COUNT                           17186 non-null  int64
 29  NORMALIZED_CYLOMATIC_COMPLEXITY      17186 non-null  int64
 30  NUM_OPERANDS                         17186 non-null  float64
 31  NUM_OPERATORS                        17186 non-null  int64
 32  NUM_UNIQUE_OPERANDS                  17186 non-null  int64
 33  NUM_UNIQUE_OPERATORS                 17186 non-null  int64
 34  NUMBER_OF_LINES                      17186 non-null  int64
 35  PERCENT_COMMENTS                     17186 non-null  int64
 36  LOC_TOTAL                            17186 non-null  int64
 37  loc_count                            17186 non-null  float64
 38  Branch                               17186 non-null  int64
 39  TARGET                               17186 non-null  int64
dtypes: float64(14), int64(26)
memory usage: 5.2 MB
```

Figure 3. 3PC1 cleaned dataset

3.2.2 Feature Selection

Software defect datasets contain significant features and provide abundant information to build a robust defect prediction model. But, all features are not equally important since some features are inappropriate, and leads to an off-target classification.Therefore, inthe defect prediction process, feature selection is used in order to increase efficiency and classification performance.Since feature selection techniques are used to find a subset of highly discriminant features, it selects features that are capable of discriminating features that belong to the predicted target class.

To do this, we used a filter-based feature ranking and selection techniques. The filtering method uses the computational characteristics of datasets to independently assess and rank features in datasets that are found to be independent of the prediction model (Ahmad, 2019). It selects the most significant features from a given high dimensionaldataset. Features are selected on the basis of their individual score in various statistical tests for their correlation with the outcome variable.

Then, we selected the top $\log2^{N}$ number offeatures from six NASA datasets according to a given performance metric based ranking and used them as the set of the selected attributes for model building, where N is the total number of software metricsfeatures. The reasons why we selected the top [$\log2^{N}$] features include: related literature(Shuib Basri, et al, 2019),(Taghi, Kehan, 2009), showed that it was appropriate to use top $\log2^{N}$ features for the binary classification problem, in general for particular imbalanced datasets. Although we used the SVM learner in this study, our result showed that $\log2^{N}$ is a good choice for this learners as it obtained good classification results.

Chi-Square Test for Feature Selection

The Chi-square ($\chi2$) is one of the commonly used FBFS statistical test methods. $\chi2$ test is used to examine the distribution of the class as it relates to the values of the given feature. It grades each feature based on different characteristics such as statistics, probability, instance, or classifier based indicators(Seliya, 2010).In this technique, the score is given to each feature is called **ranking**. When two software features are independent, the observed value is close to the expected value, thus we will have matching Chi-Square value. In simple words, higher the Chi-Square value the feature is more dependent on the response of the target predicted class and it can be selected for model training.

The practical meaning of the Chi-square test is that it is a hypothetical test that is used to test the relevance of the two features and evaluate whether two events are independent or relevant. When describing the degree of deviation between the observed and expected values, the larger the Chi-square value, the weaker the independence of the two variables, and the stronger the independence of the two variables on the contrary.

We have a target variable (class label) and feature variables that describe each sample of the data. Now, we can calculate the Chi-square between every feature variables and the target variables, and observe the existence of the relationship between the feature and the target variable. If the target variable (predicted class) is independent of the feature variable, discard the feature variable. And if the target and the feature variable are dependent, the feature variables are very significant and select for model building.While doing the feature selection process, there are two steps. The first step is calculating the score of each individual feature.Therefore, we set the expected value of **E** and the observed value of ***xi,*** thus, the degree of deviation to determine the Chi-square value $\chi 2$ can be expressed as:

$$\chi 2 = \sum \frac{(xi-E)2}{E} \quad (3.1)$$

In the secondstep, selecting the best features which have the highest score corresponding to the output of the target class.Scikit-learn python packages provideSelectKBest class that can be used as a suit of selecting the top K features based on their score.

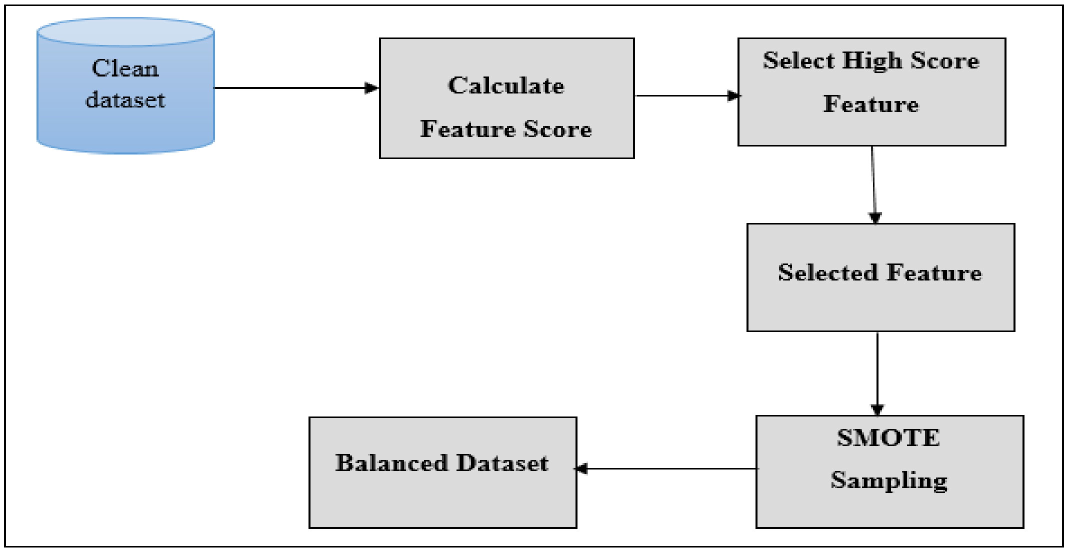
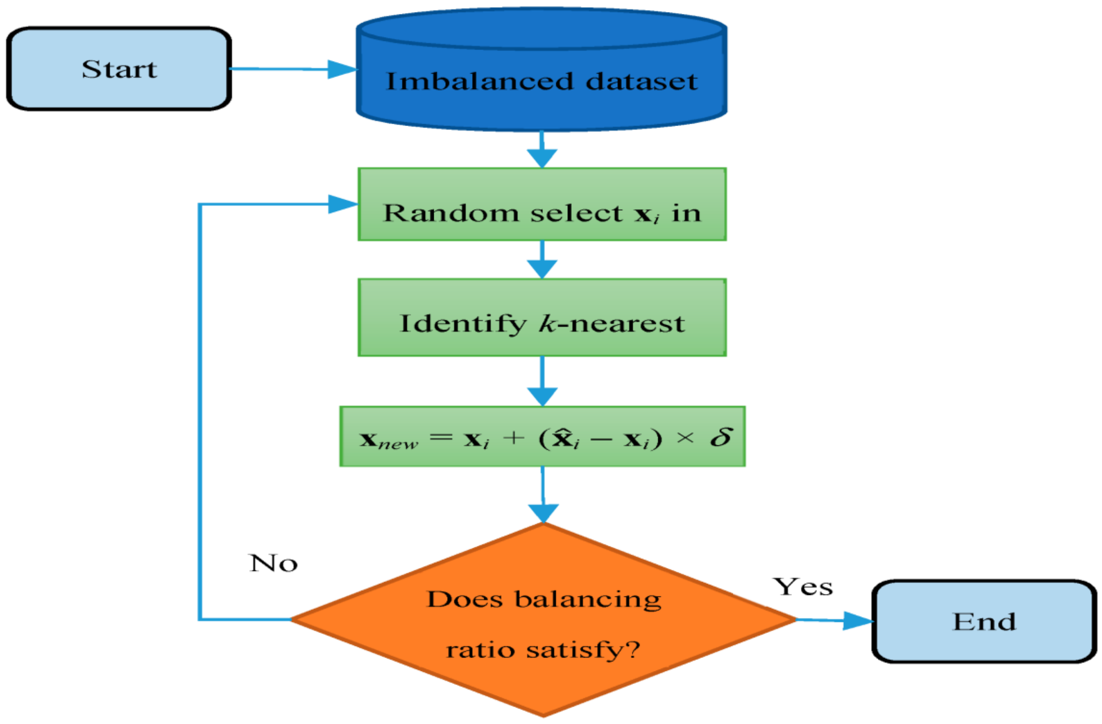


Figure 3. 4Architecture of feature selection and data sampling

3.2.3 Data Sampling

The Software defect dataset is extensively affected by the class imbalance problem in which the majority feature of software productsare defect-free. Training ML model with this imbalance dataset often causes the model to develop a certain bias in the direction of the majority class. This leads to an off-target prediction of a defect in which the efficiency of software defect prediction is greatly formed by the class distribution of the training data.

Therefore, in the data sampling phase, we used SMOTE sampling techniques on the selected features to elucidate the highly class imbalance rate.It is an effective sampling method that achieves balanced dataset by creating extra training sample data for minority groups since it works based on nearest neighbors judged by euclidean distance between instances in feature space unlike undersampling techniques. As we have discussed in section 2.8.4, SMOTE synthesis new samples by inserting samples between minority class samples that lie together. Thus, the overfitting problem is escaped and the decision space for the minority class increase. This enables the advantage of learning models to train adequately for the minority class on the selected relevant features. **Figure 13** shows how the SMOTE algorithm attains balanced training data by generating new synthetic data.



Start
Imbalanced dataset
Random select $\mathbf{x}_i$ in
Identify k-nearest
$\mathbf{x}_{new} = \mathbf{x}_i + (\hat{\mathbf{x}}_i - \mathbf{x}_i) \times \delta$
No
Does balancing ratio satisfy?
Yes
End

Figure 3. 5flow chart of SMOTE Algorithm

3.4.4 Classification

Software defect prediction has a task binary classification problem in which all the elements of the given dataset is classified into two groups based on the classification rule. And, building a robust classification model is the primary goal of defect prediction on historical defect dataset. So that, suspicious selection of classification algorithms plays the main role to build a strong predictive model since the imbalanced binary defect data has a great influence to train it. Because training ML classifier with the low ability to handle imbalance issue yields incorrect predicting results.

Therefore, in the classification phase, we used a support vector machine classifier for predicting whether the software modules are defective or non-defective. SVM is the most popular algorithm used for binary classification problems in different areas like face detection, handwriting recognition, bioinformatics, and software testing(Munir Ahmad, et al, 2019). The main idea of SVMis looking for a hyperplanein a high-dimensional space as a parting plane for two aspects in order to ensure a minimumerror rate. Then, we find the ideal hyperplane that differentiates between the two classes using a kernel function.SVM classifier which used for linearlyseparable data,where the training data can be separated by a hyperplane,

$w0 * x + b = 0.$ (3.2)

If the sample sets are linearly separable, (x1; y1)…….., (xm; ym); $x \in Rn$; $yi \in \{1; -1\}$ are assumed. **x** is the number of samples and **y** stands for category, n stands for entered dimension.

Linear soft margin classifier: is a type of SVM classifier which used for handling linearly non-separable input features that are overlapping or linearly non-separable classes. This classifier with different kernel functions can transform a linear non-separable problem into a linearly separable problem by projecting data into the feature space. That means it separate the training data of the two classes with a minimal number of errors. Therefore, theabovelinearly separable equation can be re-written as f(x):

$$\mathbf{w .xi + b >= 1 - ei, \text{ if } yi = 1}$$

$$\mathbf{w .xi + b <= -1 - e, \text{ if } yi = -1} \qquad \mathbf{(3.3)}$$

Where b is scalar and w is p-dimensional Vector and where ei, is non-negative slack variables. If $f(x) \geq 0$, the category label equals to 1, otherwise, -1 (defective and non-defective).

When working with SVM, there are two common steps. In the first step, we have to find the data points that lie closest to both the classes. These points are known as support vectors. In the next step, we find the proximity between our dividing plane and the support vectors. The distance between the points and the dividing line is known as margin. When the margin reaches its maximum, the hyperplane becomes the optimal one.

In defect prediction, a predictive model is a representation of twoclasses (defective and non-defective) in a hyperplane in bi-dimensional space by which it divides the datasets into classes to find a maximum marginal hyperplane, and it tries to enlarge the distance between the two classes by creating a well-defined decision boundary. In practice, the SVM algorithm is implemented with a kernel function that transforms an input data space into the required form.

Radial basis function kernel (RBF)

In order to increase the performance of the classifier, we used the RBF kernel function. The secret behind the success of SVM for binary classification is that SVM uses a technique called kernel function. RBFis a powerful kernel functionthat is used in support vector machine classification(Noreen Kausar, et al, 2016). Applying kernel function means just to the replace dot product of two vectors by the kernel function.

The RBF kernel on two samples **x** and **y**, represented as feature vectors in some input space, is defined as $K(x,y) = \exp\left(-\frac{||x-y||2\ )}{2\sigma 2}\right)$ (3.4)

Where $||\ x\text{-}y||^2$can be recognized as the squared Euclidean distance between the two feature vectors of x and y, and $\boldsymbol{\sigma}$ is a free parameter.

RBF is the mostsignificant kernel function whose value depends on the distance from the origin or from some point(Thant, Nyein , 2015).In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible, and accurate.In this process, since code metric values are numerical data, which is used as training and testing input data for theclassifier. At this stage, we used the balanced training dataset in order to train(learn) the proposed model, unlike testing data. Therefore, the RBF kernel transforms the input test datasets into two separable classes by creating high optimal hyperplane.

CHAPTER FOUR

ExperimentandResultDiscussion

4.1 Introduction

In this chapter, we described the experimental results and evaluation of experimental result of the proposed model for the prediction of defects in software modules. The experimental evaluation of the proposed model for the software defect prediction model is described in detail. Experimental results are evaluated using a confusion matrix, ROC-score curve, and statistical performance that approves the consciousness of the proposed class imbalance defect prediction model are described. The dataset used, their characteristic, and the implementation of the proposed defect prediction model are also described thoroughly. In addition, the effect of feature selection and sampling techniques are evaluated and compared before and after these techniques are applied. Finally, the experimental test results of our study are compared with state-of-the-art models.

4.2 Dataset

Datasets used for this studyare collected fromthe National Aeronautics and Space Administration (NASA) Modular toolkit for Data Processing (MDP) promises repository. NASA MDP promise repository stores a collection of the software defect data that are commonly used by software engineering research community to construct defect predictive models(Ahmed, Umair, 2019). This repository contains software defect data that are collected from the software systems that represent faults detected during software development. We have used six well-known defect datasets from this repository: MC1, JM1, KC1, PC1, PC3, and PC5. These datasets are software projects written using C, C++,and Java programming languages. In order to select those datasets, we have two main reasons.First, these datasets have high class-imbalance problems(unequal distribution of the majority and the minority class). Second, the study in(Shuib Basri, et al, 2019), (Haixiang, et al, 2017),(Ahmed, Umair, 2019), and(David, et al, 2018) are related work used these datasets and they recommend that class imbalance issue still needs more finding.Therefore, this can be used as a baseline for our researches so that the results of the proposed model can be compared and easily verified with those.

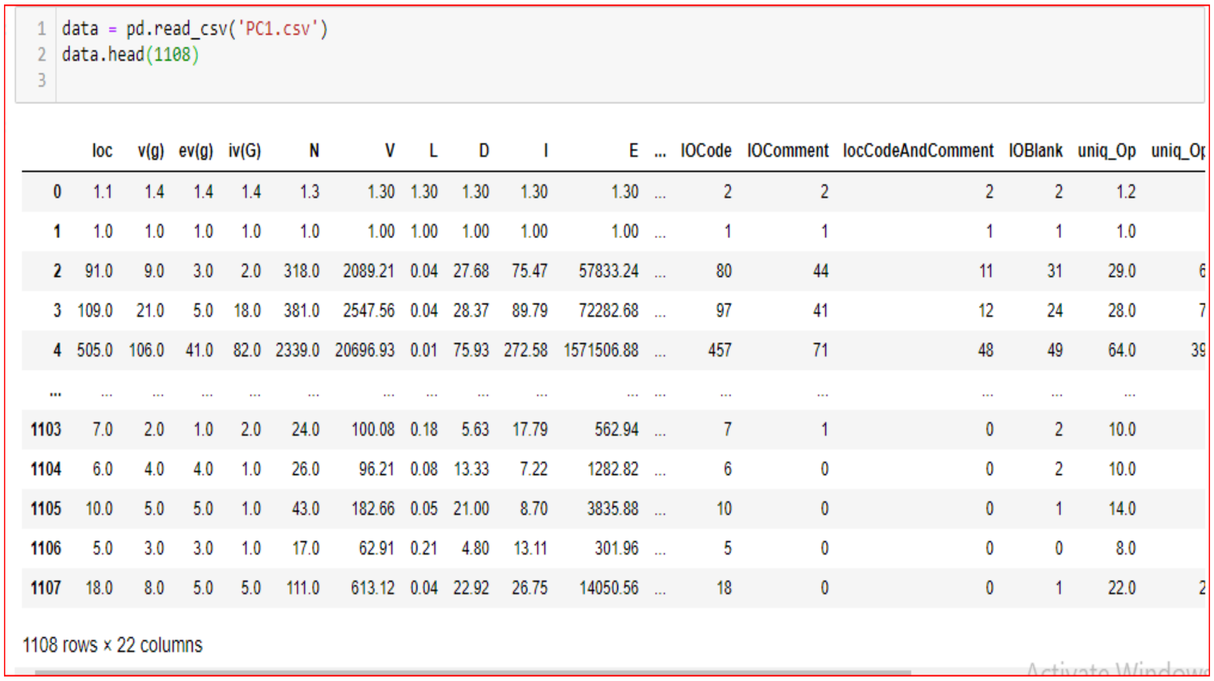
4.2.1 Dataset Details

Table 4.1 shows a detail summary of the datasets. It contains project name, class imbalance rate, software metrics, programming language and number of software features.Dataset PC1, JM1, KC1 projects have 21 fatures while the remaining PC3, PC5 and MC1 dataset has 39 features. Each of these dataset features have a numerical value with their intended target class.

Dataset	**Language**	**No. of attributes**	**No. of modules**	**Non-defective**	**Imbalanced rate**
PC1	C	21	1109	1032	6.94 %
JM1	C++	21	10,885	8779	19.3 %
KC1	C	21	2,109	1783	15.4 %
PC3	C++	39	1985	1897	7.9 %
PC5	C++	39	17,186	16,670	3.0 %
MC1	Java	39	2,670	2,322	13.4 %

Table 4.1Dataset description

Table 4.2 shows the values of sample PC1 dataset which have 22 software features.



```
data = pd.read_csv('PC1.csv')
data.head(1108)

```

```
        loc   v(g)  ev(g)  iv(G)     N         V     L      D       I           E  ...  lOCode  lOComment  locCodeAndComment  lOBlank  uniq_Op  uniq_Op
   0    1.1    1.4    1.4    1.4     1.3      1.30  1.30   1.30    1.30        1.30  ...       2          2                  2        2      1.2
   1    1.0    1.0    1.0    1.0     1.0      1.00  1.00   1.00    1.00        1.00  ...       1          1                  1        1      1.0
   2   91.0    9.0    3.0    2.0   318.0   2089.21  0.04  27.68   75.47    57833.24  ...      80         44                 11       31     29.0        6
   3  109.0   21.0    5.0   18.0   381.0   2547.56  0.04  28.37   89.79    72282.68  ...      97         41                 12       24     28.0        7
   4  505.0  106.0   41.0   82.0  2339.0  20696.93  0.01  75.93  272.58  1571506.88  ...     457         71                 48       49     64.0       39
 ...    ...    ...    ...    ...     ...       ...   ...    ...     ...         ...  ...     ...        ...                ...      ...      ...
1103    7.0    2.0    1.0    2.0    24.0    100.08  0.18   5.63   17.79      562.94  ...       7          1                  0        2     10.0
1104    6.0    4.0    4.0    1.0    26.0     96.21  0.08  13.33    7.22     1282.82  ...       6          0                  0        2     10.0
1105   10.0    5.0    5.0    1.0    43.0    182.66  0.05  21.00    8.70     3835.88  ...      10          0                  0        1     14.0
1106    5.0    3.0    3.0    1.0    17.0     62.91  0.21   4.80   13.11      301.96  ...       5          0                  0        0      8.0
1107   18.0    8.0    5.0    5.0   111.0    613.12  0.04  22.92   26.75    14050.56  ...      18          0                  0        1     22.0        2

1108 rows × 22 columns
```

Table 4. 2 PC1 dataset Values

4.2.2 Dataset Description

Table 4.3 shows the description of each of software metrics. There are 40 features:thirteen basic software metric, eight derived halstead metrics, eighteen design metrics,and one target class which are useful to identify either a software has any defect or not(Agarwal, Tomar, 2014).

Number	**Software metrics (Attributes)**	**Definition**	**Category**
1	Loc	Total Line of Code	McCabe Attributes
2	V(g)	Cyclonic Complexity	
3	Ev(g)	Essential complexity	
4	Iv(g)	Design complexity	
5	N	Total number of oper& operands	Basic Halsted Attributes
6	V	Volume	
7	L	Program length=(v/n)	
8	D	Difficulty=(1/L)	
9	I	Intelligence	
10	E	Effort to write the program	
11	B	Effort estimate	
12	T	Time estimator=E/18s	
13	lOCode	Count of statement lines	Derived Halstead Attributes
14	IoComment	Count of comment lines	
15	lOBlank	Count of blank lines	
16	locCodeAndComment	Count of code and comment line	
17	Uni_op	Total number of unique operators	
18	Uni_opnd	Total number of unique operands	
19	Total_op	Total number of operators	
20	Total_opnd	Total number of operands	

21	Branch_count	Total number of branch counts	
22	Parameter_Count	Total number of parameter	Misc.
23	Decision_Density	Decision Density of the code	
24	Essential_Density	Essential Density of program	
25	Loc_Executable	Number of executable line of code	
26	Halstead_Error_Est	Halstead error estimation	
27	Call_Pairs	Calling method pairs	
28	Decision_Count	Count the number of edges	
29	Edge_Count	Count the edge of code	
30	Halstead_Error_Est	Estimation of error in halstead	
31	Cyclomatic_Density	Cyclomatic density	
32	Halstead_Content	Content of the halstread	
33	Halstead_Prog_Time	Time to halstead the program	
34	Node_Count	Total number of node	
35	Maintenance_Severity	Severity to maintain the error	
36	Design_Density	Design density of the program	
37	Normalized_Cylomatic_Complexity	Cylomatic Complexity of the code	
38	Multiple_Condition_Count	Total number multiple condition	
39	Modified_Condition_Count	Total number of modified condition	
40	Target class	Defective or non-defective	

Table 4. 3 Description of features

4.3. Implementation

Experiments are done based on the model developed with Anaconda(Tensor Flow as a backend) on Intel(R) Core(TM) i3-5005 CPU@ 2.00GH, and 4 GB of RAM. Anaconda within thePyCharm environment is an exposed source delivery of pythonprogramming language for systematic computing. PyCharmis talented in consecutively on the upperof Tensor Flow. Tensor Flow is a representative math public library and used for machinelearning applications. Imbalanced-learn python library used for data sampling is used for theSMOTE algorithm.The data is partitioned into training andtesting dataset such that 70 percent of the data is assigned for training the model and 30 percent ofthe data is selected for testing.

4.4. Experiment

This section describes the experimental study carried out for software defect prediction performance analysis using NASA MDP promise datasets. The performance of these datasetsis studied using feature selection and synthetic minority over sampling techniques. Furthermore, the studies have two experimental scenarios and it is carried out using the proposed model and the experimental results are discussed.

Before the analysis and comparison of the test result of the proposed model with the state-of-the-art models, we have to clearly showthe intermediate data preprocessing results (data cleaning, feature selection data sampling).

4.4.1 Data Cleaning

These defect data are often extracted from logfiles, version controls, and bug reports in bug tracking databases automatically by usingtools. However, these data contain missing values, duplicate instance, and correlate attributes. In order to identify and remove these problems, we useda filter threshold method that identifies the correlation between the software features as shown in table 4below. This is done usingsklearn in python library. These correlated and redundant features are selected and removed since they affect the classification performance.

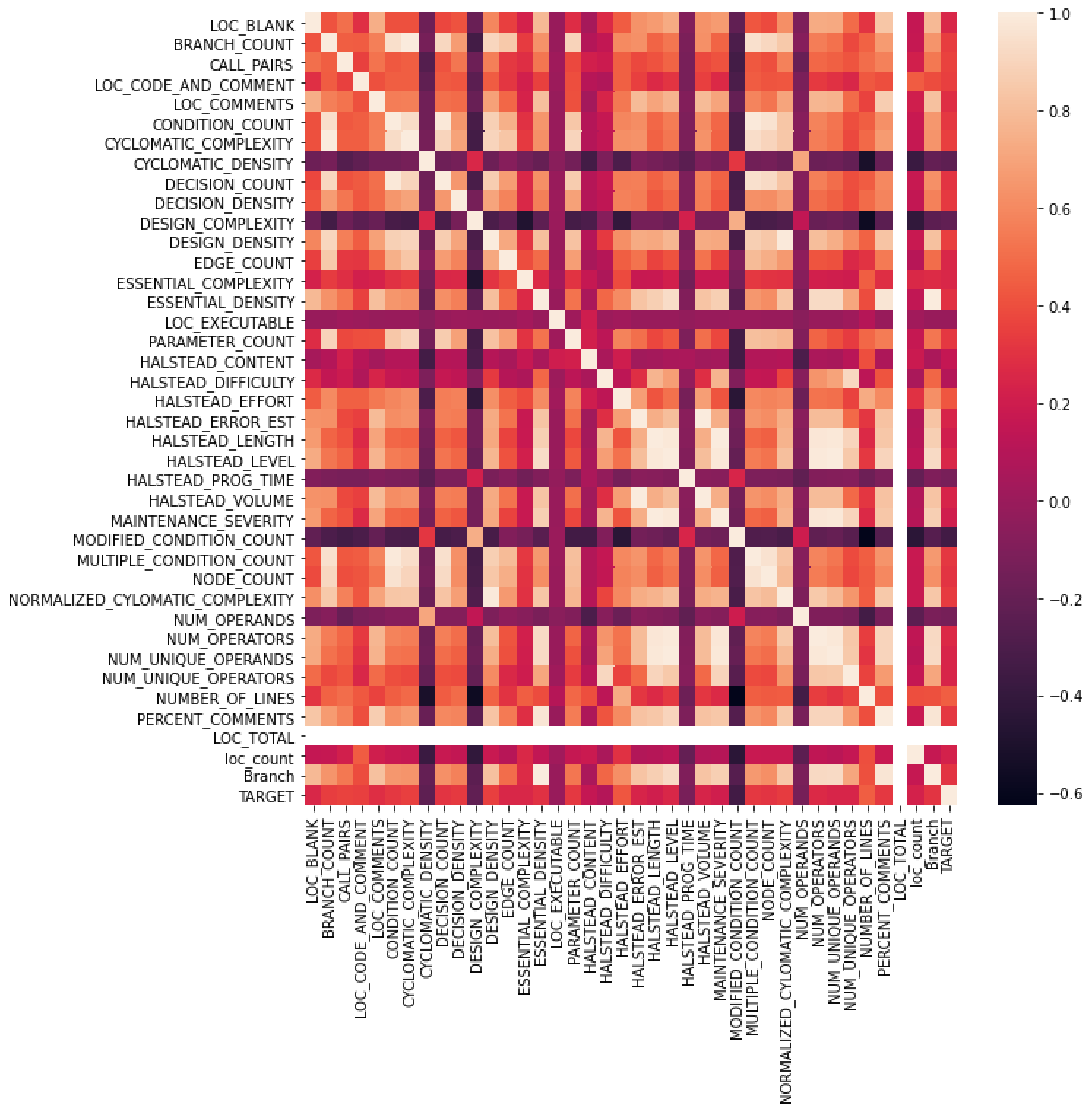
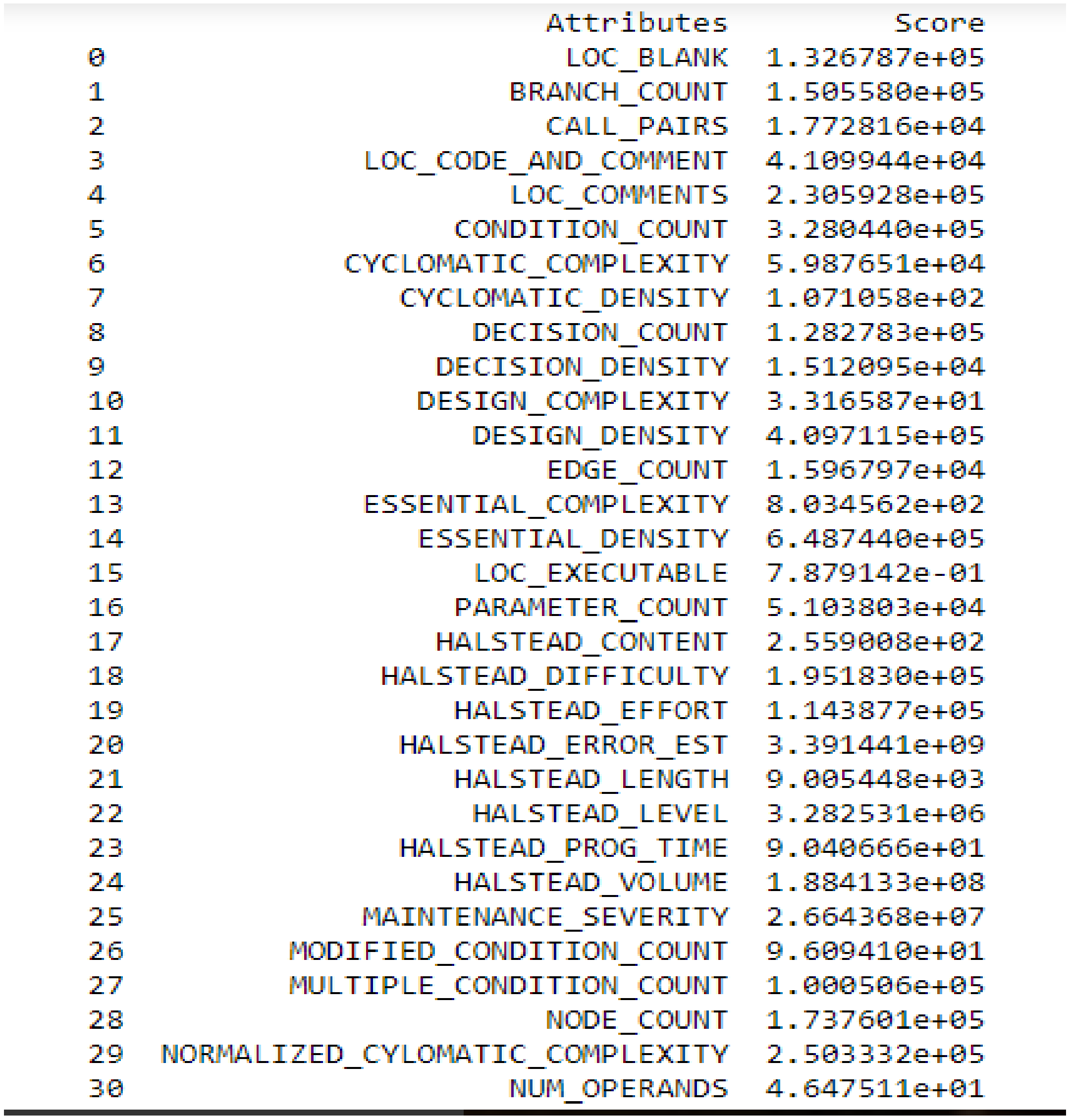


Table 4. 4 correlation of features

Correlated features meana software features that they bring the same information or have less effect for building a prediction model, so it is reasonable to remove or select the most important features among them. Therefore, we removed thesecorrelated and irrelevant features to make the learning algorithm faster by decrease the dimensionality of features in which fewer features usually mean high improvement in terms of speedand decrease harmful biasbetween features.

4.4.2FeatureSelection

In our experiments, we determine the influential or associational relationshipsamong the software metric attributes and defectiveness of the software modules by identifying the most effective attributes by giving those scores and considering their effect on defect proneness.While performing the feature selection process, there are two steps. In the first step, we calculate each individual score of the attribute with their values using theChi-Square method corresponding to the desired output. As shown in the following figure, the score of an individual attribute is listed.



```
                              Attributes         Score
0                              LOC_BLANK  1.326787e+05
1                           BRANCH_COUNT  1.505580e+05
2                             CALL_PAIRS  1.772816e+04
3                      LOC_CODE_AND_COMMENT  4.109944e+04
4                           LOC_COMMENTS  2.305928e+05
5                        CONDITION_COUNT  3.280440e+05
6                   CYCLOMATIC_COMPLEXITY  5.987651e+04
7                      CYCLOMATIC_DENSITY  1.071058e+02
8                         DECISION_COUNT  1.282783e+05
9                       DECISION_DENSITY  1.512095e+04
10                     DESIGN_COMPLEXITY  3.316587e+01
11                        DESIGN_DENSITY  4.097115e+05
12                            EDGE_COUNT  1.596797e+04
13                  ESSENTIAL_COMPLEXITY  8.034562e+02
14                     ESSENTIAL_DENSITY  6.487440e+05
15                        LOC_EXECUTABLE  7.879142e-01
16                       PARAMETER_COUNT  5.103803e+04
17                      HALSTEAD_CONTENT  2.559008e+02
18                   HALSTEAD_DIFFICULTY  1.951830e+05
19                       HALSTEAD_EFFORT  1.143877e+05
20                    HALSTEAD_ERROR_EST  3.391441e+09
21                       HALSTEAD_LENGTH  9.005448e+03
22                        HALSTEAD_LEVEL  3.282531e+06
23                    HALSTEAD_PROG_TIME  9.040666e+01
24                       HALSTEAD_VOLUME  1.884133e+08
25                  MAINTENANCE_SEVERITY  2.664368e+07
26              MODIFIED_CONDITION_COUNT  9.609410e+01
27              MULTIPLE_CONDITION_COUNT  1.000506e+05
28                            NODE_COUNT  1.737601e+05
29    NORMALIZED_CYLOMATIC_COMPLEXITY  2.503332e+05
30                          NUM_OPERANDS  4.647511e+01
```

Figure 4. 1the individual Score of each features

In the second step, we are select the relevant and influential features based on their score.The highest score value of the attributes is the most important feature to build the predictive model. The results of feature selection tests with filter feature ranking and selection method and where the rankings of the attributes are shown for each dataset. First ***SelectKBest*** methods ranking each attribute based on their individual score and selecting the best attributes which have the highest score corresponding to with the output of the target class which is defective or defect-free.

In order to know, how many attributes are select from a given dataset, there is a formula proposed by many different researchers in this area: Number of feature selected $=\log_2 N$, where n=, N total number of features of the dataset. For example, the PC1 dataset has 39 features. So, we calculate the number of feature to select for the building model.
Number of selected feature = $\mathrm{Log}_2{}^N$, given N= 39
Number of selected feature = $\mathrm{Log}_2{}^{39}$ =6
Number of selected features = 6 features.

Therefore, we have selected the top sixfeatures fromthe dataset attributesbased on their scores for the selected six NASAMDP projects. The question in **RQ2, Which software features (attributes) are critical for class imbalance defect prediction is answered here.** Therefore, Effort to write the program (**E**)**,** Time estimator (***T***), Halted Volume(***V***), Total number of operand & operands (***N***), unique operand (***Uniq_Opnd)*** and **line of code** are top six software features of JM1, PC1, and KC1 for class imbalance defect prediction.

On the other hand,PC3, PC5, and MC1 dataset is another group that have thirty nine software features, and we have selectedthis features.***Essential Complexity***, ***Halstead Content***, ***Halstead Level***, ***Maintenance Severity***, ***Halstead Effort***, ***Number Of Unique Operator And Operands***are the top sixsoftware features of PC3, PC5, and MC1 dataset for class imbalance defect prediction. These selected software featureshave the strongest relationship with the output of the target class, which are defective or non-defective. Then, the proposed model is trained using the selected software features after SMOTE data processing is conducted.

4.4.3 SMOTE Sampling

In our experiments, we solved the class imbalance problem using SMOTE data sampling on the selected software features. It is done by altering or modify the training data distributions of the minority class.SMOTE works by selecting one existing minority class sample m, next it finds its k-nearest neighbors and should select at random one of the k samples, called n. Then calculate the difference between samples m and n and then multiply this with a random number between 0 and 1, and the resulting value is Synthetic data and it is added to the feature vector the train data.

For example: First, let us show the original dataset class distribution of the JM1 dataset, the blue line labeled in false non-defective, and the red defective data. This shows how imbalanced is our original dataset. Most of the transactions are non-defect. we do not want our model to assume, we want our model to detect patterns that give signs of defect. After using SMOTE sampling, the dataset is balanced as shown in the right figure4.2 right. Which means, the number of the minority class and the majority class training samples are equally. Now, the proposed model has enough data to learn from features in both classes. Then we use this data frame to build our predictive models and analysis we get an accurate result, and it delivers better performance than the previous dataset. Hence, both classes have an equall number of instances to learn the model.

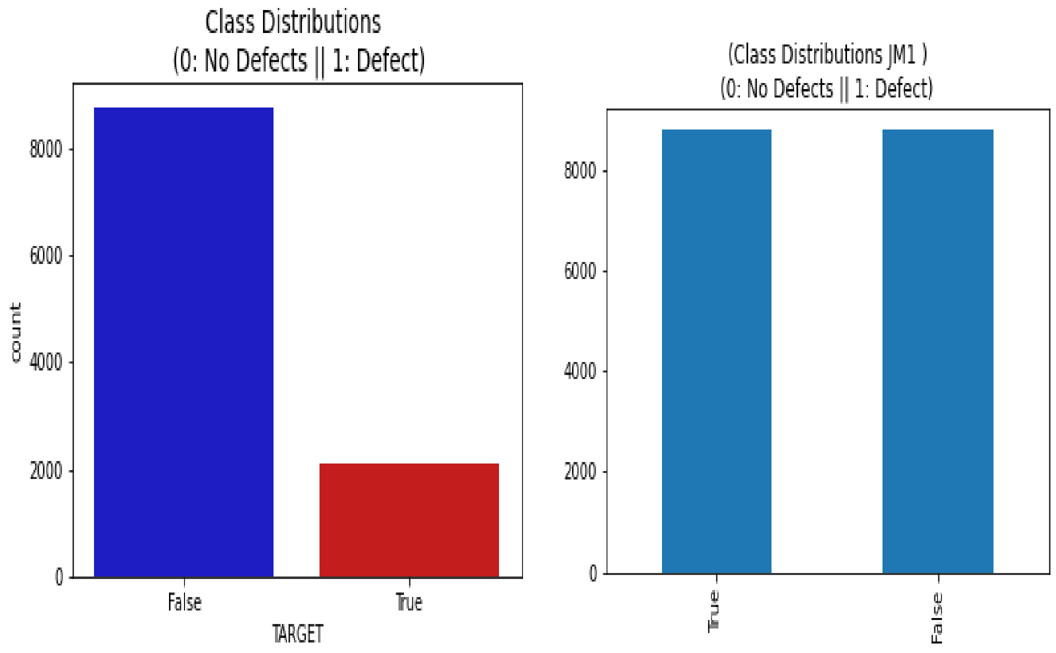


Figure *4. 2*Class Distribution before and after sampling

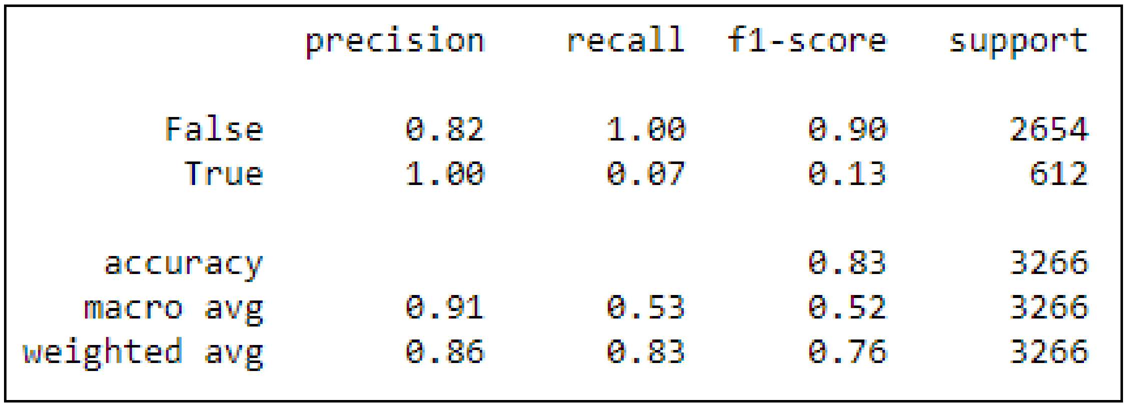
4.6Test Result and Discussion

4.6.1 Experimental Result for JM1 Dataset

In this section we present an experimental study on theJM1 dataset using the proposed approach for software defect prediction. The experiments have two scenarios, and the performance of the experimental results is evaluated using statistical, confusion matrix, and ROC-score curve metrics exhaustively.

I. Statistical performance analysis

Scenario one: making defect classification on the original dataset without feature selection and SMOTE sampling techniques are applied. Table 4.5 shows the result of the statistical performance for the JM1 dataset before data preprocessingusing the SVM classifier.

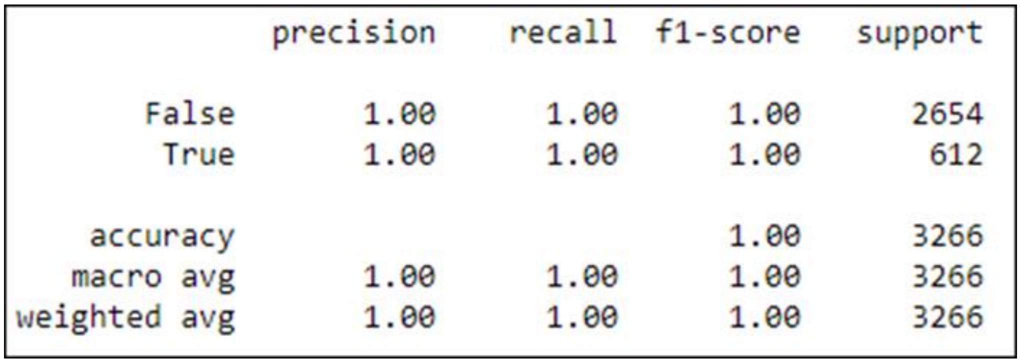


```
              precision    recall  f1-score   support

       False       0.82      1.00      0.90      2654
        True       1.00      0.07      0.13       612

    accuracy                           0.83      3266
   macro avg       0.91      0.53      0.52      3266
weighted avg       0.86      0.83      0.76      3266
```

Table 4. 5 Result of JM1 Dataset before data preprocessing

As shown in table 4.5, the performance of the majority class is higher than the minority class. For example Precision, Recall, and F1-score value of the majority class is 82%, 100% 90%. Whereas, the classification result of the minority class is very low when it compares with the majority class since Precision, Recall and F1-score value of the minority class is 100%, 7%, and 13% respectively. This shows the classification accuracy of the minority class is dominated by the majority class. In this case, the accuracy is not a sufficient metric for making comparisons even if the accuracy is high. Because there is an extremely high difference between the two class performance.

Scenario two: First, we conducted experiments of feature selection using filter-based feature selection techniques and selected the most important features on the JM1 datasets. Second data sampling experiments are conducted on the selected features using SMOTE sampling. After that, we train our model and classify the test data using the proposed class imbalance defect prediction model. Finally, we evaluate the statistical performance of the proposed model. **Table 4.6** shows the result of the statistical performance of the JM1 dataset.



```
              precision    recall  f1-score   support

       False       1.00      1.00      1.00      2654
        True       1.00      1.00      1.00       612

    accuracy                           1.00      3266
   macro avg       1.00      1.00      1.00      3266
weighted avg       1.00      1.00      1.00      3266
```

Table 4. 6 Result of JM1 Dataset after data preprocessing

It is clear that the result of the classification accuracy of the proposed model on the JM1 dataset is 100%. That is a promising result and this shows the total correctly (true positive) classified instances from the two (defective and non-defective) classes together. Now let us see the performance of the two classes individually using Precision, Recall, and F1-Scores, unlike accuracy.

As shown in Table 4.6, the minority class classification result of Precision, Recall, and F1-Scores are 100%, 100%, and 100%. At the same time, the classification result is similarto the majority class. This shows the proposed model achieved higher performance in both the majority and minority classes unlike scenario one results since the Recall and F1-score value of the minority class is 7% and 13% respectively. This indicates there is a very high difference in results when it is compared with the experimental scenario one. Therefore, the proposed class imbalance software defect prediction model achieved promising results by combining feature selection and SMOTE sampling methods since the result of the minority and majority classes have achieved higher performance in all performance metrics on the JM1 dataset.

II. Confusion matrix analysis for JM1 Dataset

A confusion matrix of binary classification is a two by two matrix formed by counting the number of the four outcomes of a binary SVM classifier.**Table 4.7** shows the confusion matrix performance analysis for the JM1 dataset using the proposed method.The JM1 dataset has10,885 instances (records), from the total 10,885 instances, we used 70% (7619) instances for training data and 30% (3266) instances for testing data.

As presented results in the confusion matrix, the proposed model achieved results with 2654 instances are true positive samples (TP= 2604). More clearly, 2654 software modules (classes) are classified correctly as defect-freeor non-defective data which are in fact non-defective software modules. Similarly, 612 instances are true negative samples = 612, which means 612 instances are classified correctly as defective software modules which is in fact defective software modules. Totally 3266 instances are correctly classified as intended from the given dataset from their intended class. Which means, all the testing data are classified properly with their corresponding defective and non-defective class.

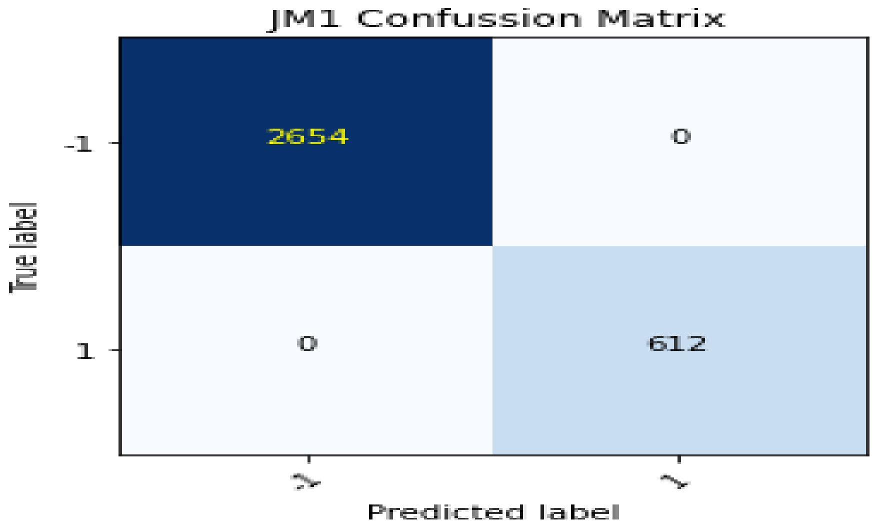


Table 4. 7 Confusion matrix for JM1 Dataset

On the contrary, as shown in the confusion matrix, both the values of false positive and false negative are zero. This shows that there is no data that are incorrectly classified. False-positive = 0, which means there is no defective data classified as defect-free software modules.Similarily, False-negative = 0, which means there is no non-defective data classified as defective software modules. Therefore, only these four misclassified data (false positive instances) have negatively affected the prediction result.

III. ROC analysis for JM1 Dataset

Figure 4.3 shows the result of the ROC-AUC curve for the JM1 dataset using the proposed defect prediction approach, where the precision rate and recall rate are compared to show the precise performance of the proposed model.

The red diagonal line indicates a random defect classifier. As shown in the figure, the red line is a default classifier that represents a completely uninformative test (weak classifier), which corresponds to an AUC of 0.5. At a threshold of 1.0, the classifier classifies no software modules are defective (all modules are non-defective) and hence have a recall and precision of 0.0. Which means that the value of true positive and true negative is equal.

The blue line in the upper curve represents the proposed model, now it is clear that the ROC-AUC of the JM1 dataset is equal to 100%. This shows the prediction accuracy of the proposed model is higher which means that the value of the true positive rate is greater than the value of the false-positive rate. A curve pulled close to the upper left corner indicates a better performing test.

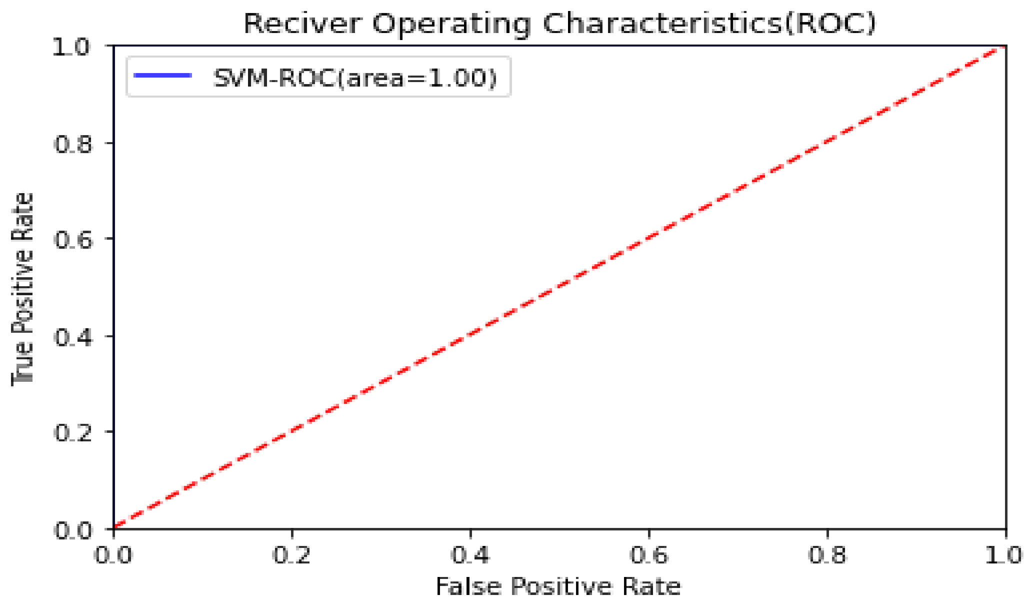


Figure 4. 3ROC analysis for JM1 Dataset

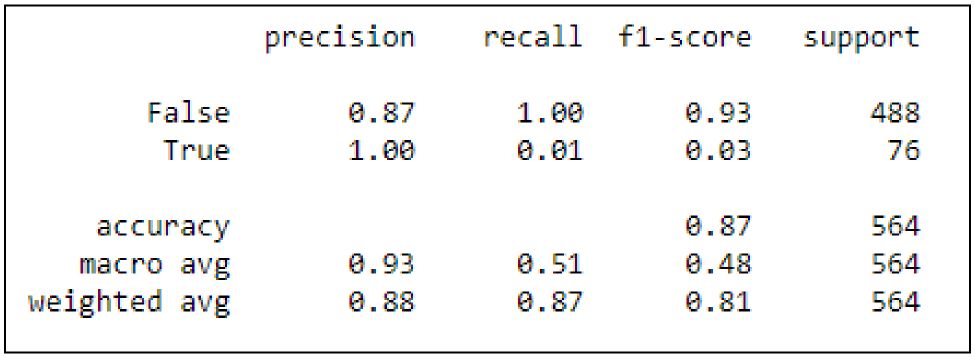
In the graph above, the ROC-AUC for the blue curve is 1.0, this is the perfect classification where the true positive rate is 100%. A high area under the curve represents both high recall and high precision, where high precision relates to a low false-positive rate, and high recall relates to a low false-negative rate. High scores for both precision and recall show the classifier is returning accurate results as well as returning a majority of all positive results of the experiments.whichmeans, at the same time, the proposed model is better at achieving a combination of precision and recall on the JM1 dataset since the ROC-AUC value is 1.

4.6.2 Experimental Result for PC3 Dataset

In this section, we present an experimental study on the PC3 dataset using the proposed approach for the software defect prediction.This experimenthas also two scenarios thatare similar with expressed in the above experimental scenarios of the JM1 dataset.

I. Statistical performance analysis

Scenario one:similarly, we made defect classification on the original dataset without feature selection, and the SMOTE sampling technique is applied. Table 4.8 shows the result of the statistical performance for thePC3 dataset before data preprocessing using the SVM classifier.

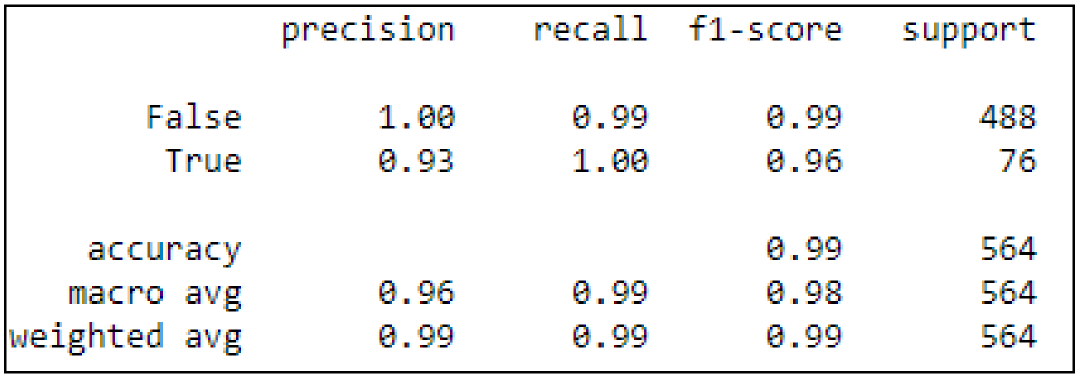


```
              precision    recall  f1-score   support

       False       0.87      1.00      0.93       488
        True       1.00      0.01      0.03        76

    accuracy                           0.87       564
   macro avg       0.93      0.51      0.48       564
weighted avg       0.88      0.87      0.81       564
```

Table 4.8 Result of PC3 dataset before Data processing

As shown in table 8, the performance of the majority class is higher than the minority class. For example Precision, Recall, and F1-score value of the majority class is 87%, 100% 93%. Whereas the performance of the minority class is very low when compared with the majority class since Precision, Recall and F1-score value of the minority class is 100%, 1%, and 3% respectively. This shows the classification accuracy of the minority class is dominated by the majority class. In this case, the accuracy is not a sufficient metric for making comparisons even if the accuracy is high. Because there is an extremely high difference between the two class performance.

Scenario two:we conducted similar experimentsas in the previous JM1 data set scenario of feature selection using SMOTE sampling techniques. **Table 4.9** shows the result of the statistical performance analysis for the PC3 dataset obtained by the proposed software defect prediction model.It is clear that the result of the classification accuracy of the proposed model on the PC3 dataset is 99%. that is a promising result and this shows the total correctly classified true positive and true negative instances from the two defective and non-defective classes together. Now let us see the performance of the two classes individually using Precision, Recall, and F1-Scores, unlike accuracy.



```
              precision    recall  f1-score   support

       False       1.00      0.99      0.99       488
        True       0.93      1.00      0.96        76

    accuracy                           0.99       564
   macro avg       0.96      0.99      0.98       564
weighted avg       0.99      0.99      0.99       564
```

Table 4.9 Result of PC3 Dataset after data preprocessing

As shown in table 4.9, the minority class classification resultof Precision, Recall, and F1-Scores are 93%, 100% and 96% respectively. It is almost similarto the majority class. This shows the proposed model achieved higher performance in both the majority and minority classes unlike scenario one results since the Recall and F1-score value of the minority class is 6% and 11% respectively. This indicates there is a very high difference in results when it is compared with the experimental scenario one. Therefore, the proposed class imbalance software defect prediction model achieved promising results by using feature selection and SMOTE sampling methods since the result of the minority and majority classeshaveachieved higher performance in all performance metrics on the PC3 dataset.

I. Confusion Matrix Analysis for PC3 Dataset

Table 4.10 shows the confusion matrix performance analysis of the proposed defect prediction model onthe PC3 dataset.

As presented results in the confusion matrix, the proposed model achieved results with 482instances are true positive(TP samples = 482). More clearly, 482 software modules (classes) are classified correctly as defect free-data which is in fact non-defective software modules. Similarly, 76 instances are true negative(TN samples =76), which means 76 instances are classified correctly as defective modules which are in fact defective software modules. Totally

558instances are correctly classified as intended from the given dataset. Therefore, the figure shows significant performance in terms of true positive rate.

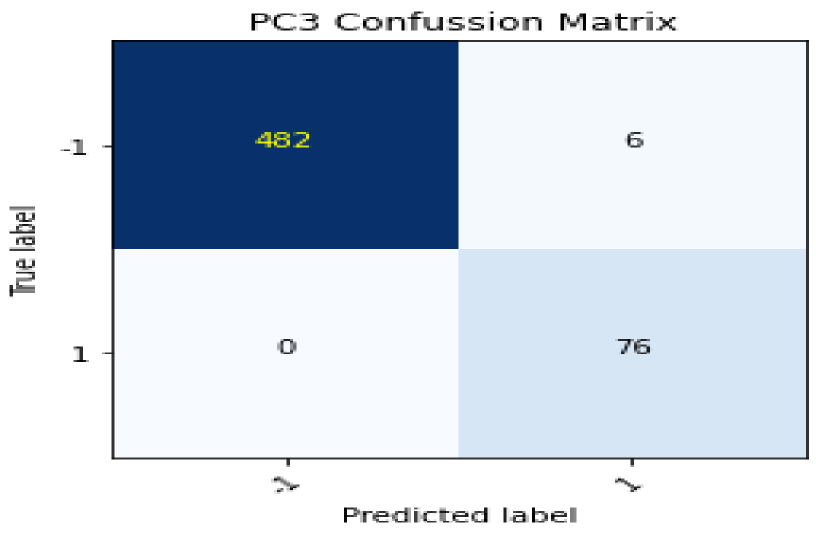


Table 4.10 Confusion matrix for PC3 Dataset

On the contrary, six instances are false negative which means these data are classified as *defective* data which is in fact defect-free software modules.Even if six instances are misclassified as a false negative, which hasa less negative effect on the prediction since the cost of false negative is usually much less than false positive. To be clear, those data are non-defective but classified as defective.

As we have seen in the table 4.10, the number of thefalse-positive instances are zero,that meansno instances are misclassified as false positive, which means there is no data are defective but classified as nondefective software module.Therefore, only these sixmis-classified data (false positive instances) have negatively affects the prediction result.

II. ROC analysis for PC3 Dataset

Figure 4.4 shows the result of the ROC-AUC curve for the PC3 dataset using the proposed defect prediction approach, where the precision rate and recall rate are compared to show the precise performance of the proposed model.

The blue line in the upper curve represents the proposed model, now it is clear that the ROC-AUC of the PC3 dataset is equal to 99.4%. This shows the prediction accuracy of the proposed model is higher which means that the value of the true positive rate is greater than the value of the false positive rate. A curve pulled close to the upper left corner indicates a better performing test.

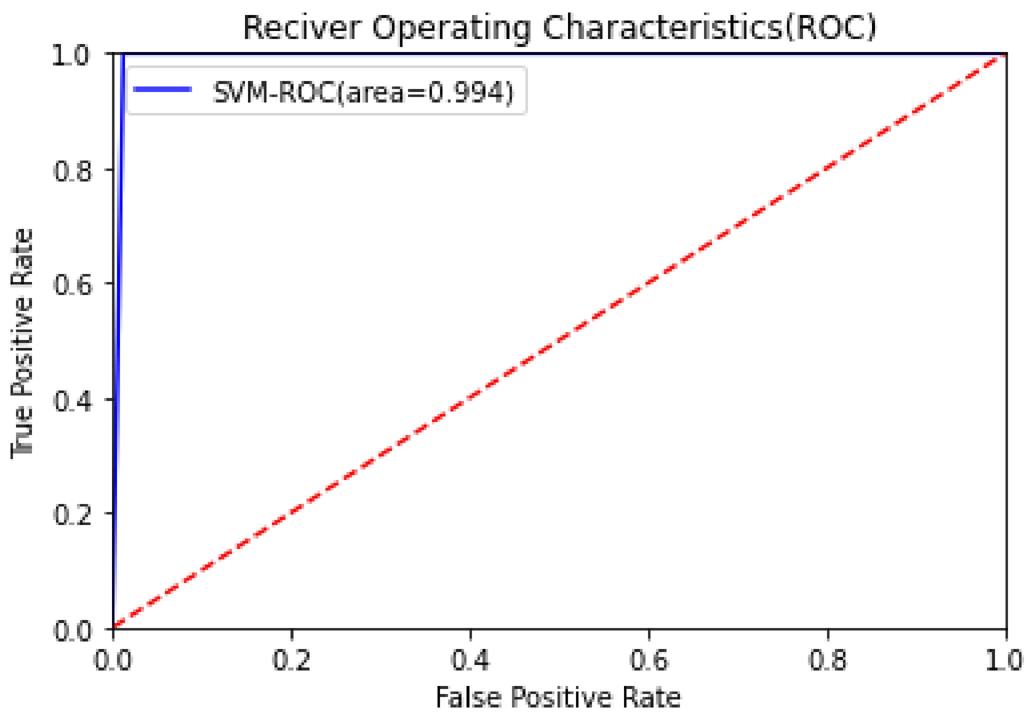
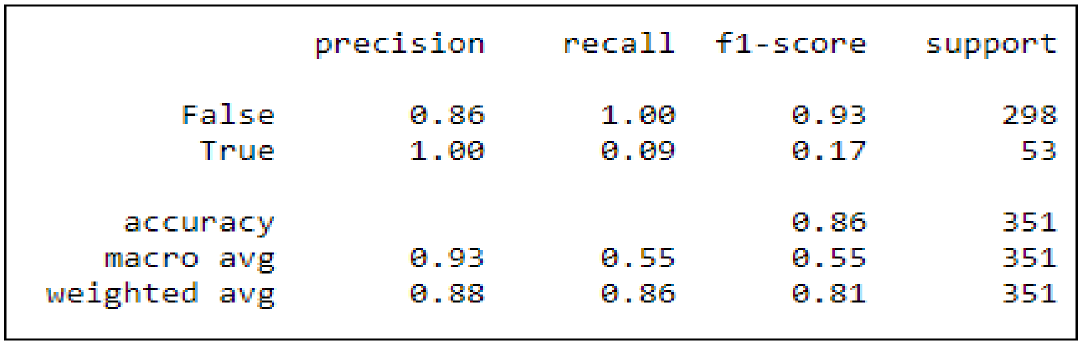


Figure 4. 4ROC analysis for PC3 Dataset

In the graph above, the ROC-AUC for the blue curve is 0.994(which is close to 1) meaning the proposed model is better at achieving a combination of precision and recall for the PC3 dataset. A high area under the curve represents both high recall and high precision, where high precision relates to a low false-positive rate, and high recall relates to a low false-negative rate. High scores for both precision and recall show the classifier is returning accurate results as well as returning a majority of all positive results of the experiments.

4.6.3ExperimentalResultfor PC1 Dataset

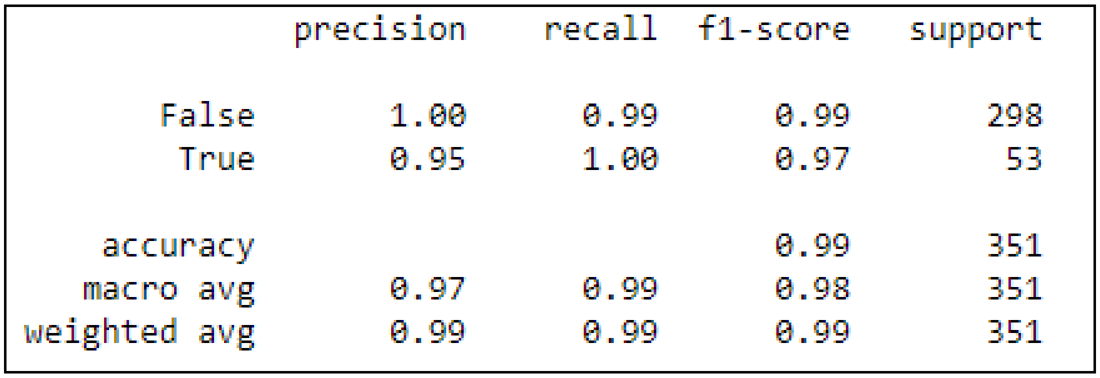
In this section, we present an experimental study on the PC1 dataset using the proposed approach for software defect prediction. The results of PC1 dataset are shown in Table 4.11, the accuracy of the majority class is high. It is reflected that before data preprocessing the results of theminority class is low. Moreover, recall, and F1-Score value are very low. Whereas the majority class are performed better than the minority class, but the cost of the minority class is higher than the majority class.



```
              precision    recall  f1-score   support

       False       0.86      1.00      0.93       298
        True       1.00      0.09      0.17        53

    accuracy                           0.86       351
   macro avg       0.93      0.55      0.55       351
weighted avg       0.88      0.86      0.81       351
```

Table 4.11 Result of PC1 Dataset before data preprocessing

As shown in table 4.11, the performance of the majority class is higher than the minority class. For examplePrecision, Recall, and F1-score value of the majority class is 90%, 100% 95%. It is a good classification result. However, in contrast, precision, recall and F1-score value of the minority class is 100, 0.07, and 0.17 respectively. It is a very low performance when compared with the majority class. This shows that the classification accuracy of the minority class is dominated by the majority class. Table 12shows the result of the numerical performance analysis for the PC1 dataset using the proposed software defect prediction model.



```
              precision    recall  f1-score   support

       False       1.00      0.99      0.99       298
        True       0.95      1.00      0.97        53

    accuracy                           0.99       351
   macro avg       0.97      0.99      0.98       351
weighted avg       0.99      0.99      0.99       351
```

Table 4.12 Result of PC1 Dataset after data preprocessing

As shown in the table, the minority class the classification result of Precision, Recall, and F1-Scores are 95%, 100%, and 97% respectively. It isalmost similarto the majority class. This shows the proposed model achieved higher performance in both the majority and minority classes, unlike scenario one results since the Recall and F1-score value of the minority class is 9% and 13%. This indicates there is a very high difference in results when it is compared with the experimental scenario one since the performance of minority class has a Recall value 100% and accuracy value 99% onthe PC1 dataset. It is a great improvement than the original dataset.

I. Confusion Matrix Analysis for PC1 Dataset

Table 4.13 shows the confusion matrix performance analysis for the PC1 dataset using the proposed method.Similarly, among the total PC1 dataset we have used 70% instances for training data and instances for testing data.

As shown results in the confusion matrix, the proposed model achieved results with 2593true positive instances. More clearly, 2593 software modulesare classified correctly as defect-free data which are in fact non-defective software modules.On the other hand, 52 instances are true negative, which means 52 instances are classified correctly as defective modules which is in fact defective software modules. Therefore, the result shows significant high performance in terms of true positive rate.

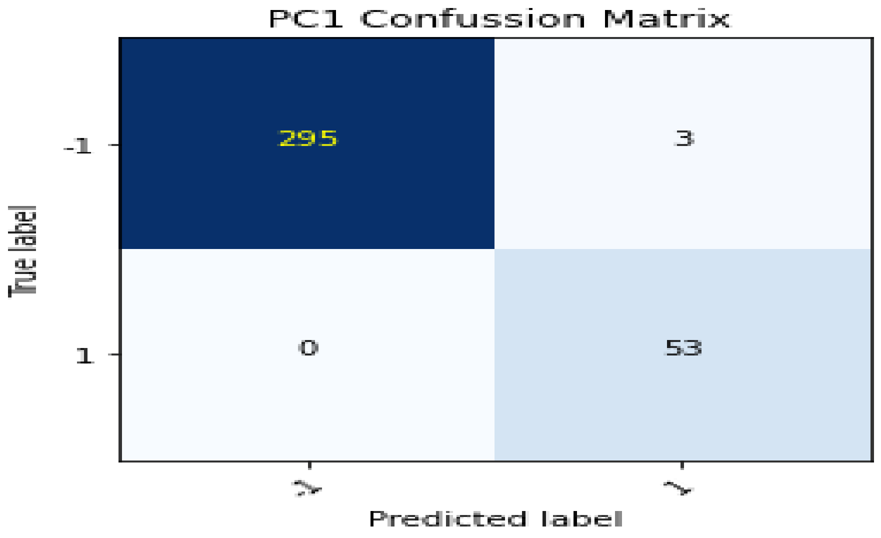


Table 4.13 Confusion matrix for PC1 Dataset

On the contrary, 52 instances are false-negative which means those data are classified as *defective* data which is in fact defect-free software modules. Even if 52 instances are misclassified as false-negative, which have a less negative effect on the prediction since the cost of false negative is usually much less than false positive. To be clear, those data are non-defective but classified as defective.As we have seen in the table, the number of **False positive instancesisthree, in which** 3 instances are misclassified as **False-positive**, which means those data are defective but classified as a non defective software module. Therefore, only these three misclassified data (false positive instances) have negatively affects the prediction result.

II. ROC analysis for PC1 Dataset

Figure 4.5 shows the result of the ROC-AUC curve for the PC1 dataset using the proposed defect prediction approach, where the precision rate and recall rate are compared to show the precise performance of the proposed model. As we have discussed in the previous section, the red diagonal line indicates a random or default defect classifier which represents awkward test (weak classifier).

The blue line in the upper curve represents the proposed model, it is clear that the ROC-AUC valueof PC1 dataset is 99.5%. This indicates the prediction accuracy of the proposed model is predicting the defects effectively. The higher ROC value shows that the value of the true positive rate is greater than the value of the false positive rate. A curve pulled close to the upper left corner indicates a better performing test.

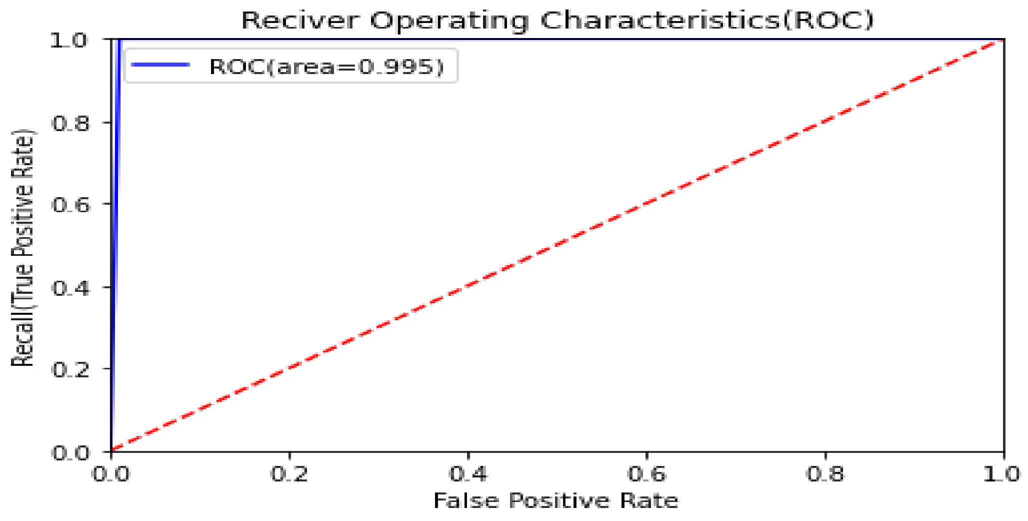


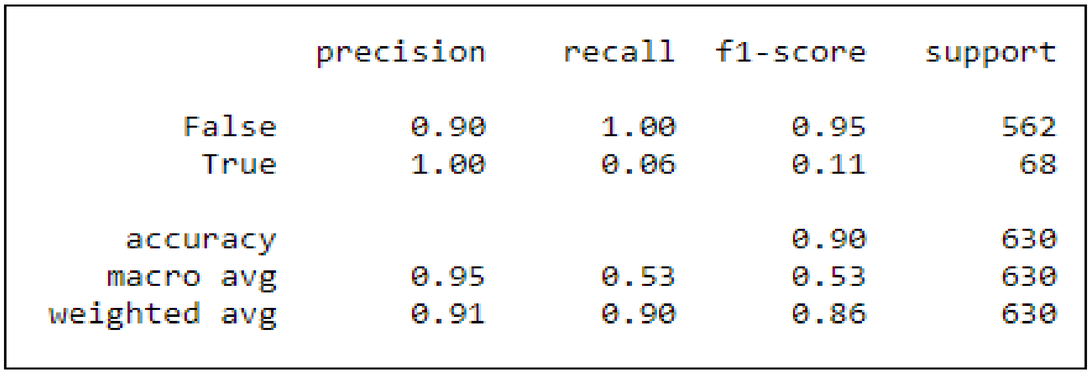
Figure 4. 5ROC analysis for PC1 Dataset

In the graph above, the ROC-AUC for the blue curve is 0.995which is close to 1 meaning the proposed model is better at achieving a combination of precision and recall for the PC1 dataset.A high area under the curve represents both high recall and high precision, where high precision relates to a low false-positive rate, and high recall relates to a low false-negative rate. High scores for both precision and recall show that the classifier is returning accurate results as well as returning a majority of all positive results.At a threshold of 0.0, the recall value of our model is 100%.This entails we find all defective software module with the defective. Therefore, the proposed model achievedthe highest performance in both the minority and majority class equallyon the PC1 dataset.

4.6.4Experimental Result for KC1 Dataset

I. Statistical performance analysis

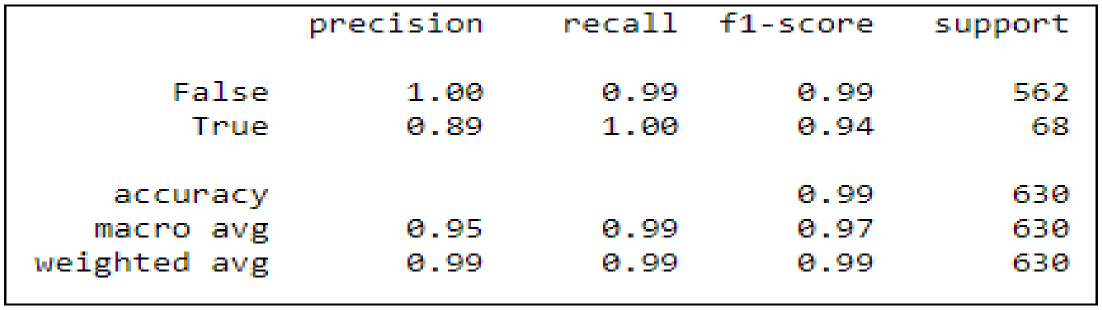
In this section, we present an experimental study on the KC1 dataset using the proposed CISDP model for software defect prediction. The results ofthe KC1 dataset are shown in Table 4.14. It is reflected that before data preprocessing the results of theminority class in Precision, Recall and F1-Score are 1.00, 0.06, and 0.11 respectively. Moreover, recall and F1-Score value is very low. Whereas the majority class is performed better than the minority class, but the cost of the minority class is higher than the majority class.



```
              precision    recall  f1-score   support

       False       0.90      1.00      0.95       562
        True       1.00      0.06      0.11        68

    accuracy                           0.90       630
   macro avg       0.95      0.53      0.53       630
weighted avg       0.91      0.90      0.86       630
```

Table 4.14 Result of KC1 dataset before Data processing

As shown in the table 4.14, the performance of the majority class is higher than the minority class. For examplePrecision, Recall, and F1-score value of the majority class is 90%, 100% 95%. It isa good classification result. However, in contrast, precision, recall, and F1-score value of the minority class is 1.00, 0.07, and 0.13 respectively. It is a very low performance when it compared with the majority class. This shows that the classification accuracy of the minority class is dominated by the majority class. Table 4.15shows the result of the numerical performance analysis for the KC1 dataset using the proposed software defect prediction model.



```
              precision    recall  f1-score   support

       False       1.00      0.99      0.99       562
        True       0.89      1.00      0.94        68

    accuracy                           0.99       630
   macro avg       0.95      0.99      0.97       630
weighted avg       0.99      0.99      0.99       630
```

Table 4.15 Result of KC1 Dataset after data preprocessing

As shown in table, the classification performance of the proposed model achieved higher accuracy in both the majority and minority classes. For example, the precision, recall, and F1-score value of the minority class are 89%, 100%, 94% respectively. This shows that the minority class has equally recognized by the classifier. The result shows thatthere is a high difference when it is compared with the experimental scenario one. The proposed defect prediction model achieved promising results with feature selection and data sampling methods since the performance of minority class has anuppermost result onthe KC1 dataset.

III. Confusion Matrix Analysis for KC1 Dataset

Table 4.16 shows the confusion matrix performance analysis for the PKC1 dataset using the proposed CISDP model. Similarly, among the total KC1 dataset we have used 70% instances for training data and 30% instances for testing data.

As shown results in the confusion matrix, the proposed model achieved results with 554instances are true positive (TP instances = 554). More clearly, 554 software modules are classified correctly as defect-free data which is in fact non-defective software modules.On the other hand, 68 instances are true negativewhich means 554 instances are classified correctly as defective modules which are in fact defective software modules. This result shows that the proposed approache achieved a significant high performance in terms of the true-positive rate.

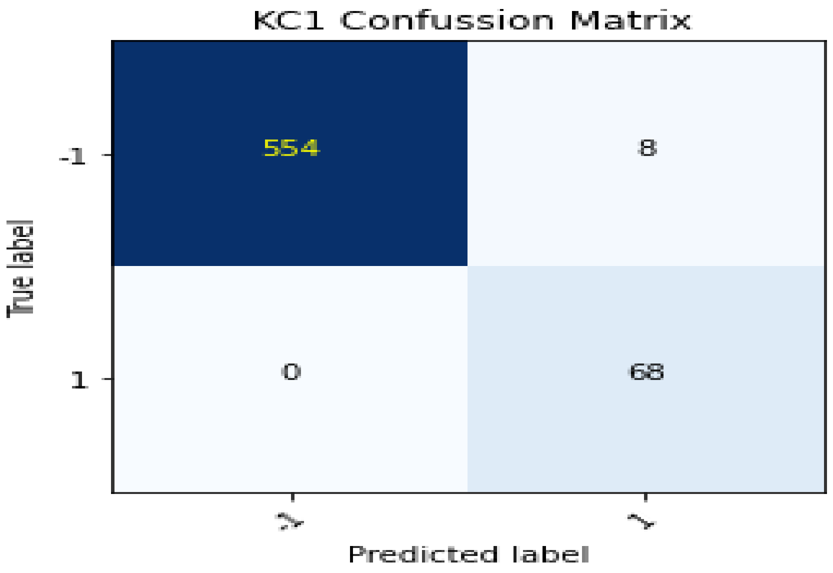


Table 4.16 Confusion matrix for KC1 Dataset

On the contrary, eight instances are false-negative which means those data are classified as defective data which are in fact defect-free software modules. Even if eight instances are mis-classified as false negative, which has a less negative effect on the prediction since the cost of false negative is usually much less than false positive. To be clear, these data are non-defective but classified as defective. As shown in the table, the number of false-positive instance are zero, which means there are no instances classified incorrectly. This indicates all the defective data are classified as a defective software modules. Therefore, among the total 630 test data, only eight misclassifieddata are false-positive instanceswhich have negatively affected the prediction result.

IV. ROC analysis for KC1 Dataset

Figure 4.6 shows the result of the ROC-AUC curve for the KC1 dataset using the proposed defect prediction approach, where the precision rate and recall rate are compared to show the precise performance of the proposed model.The blue line in the upper curve represents the proposed model, now it is clear that, the ROC-AUC of the KC1 dataset is equal to 99.3%. This shows the prediction accuracy of the proposed model is higher which means that the value of the true positive rate is greater than the value of the false-positive rate. A curve pulled close to the upper left corner indicates a better performing test.

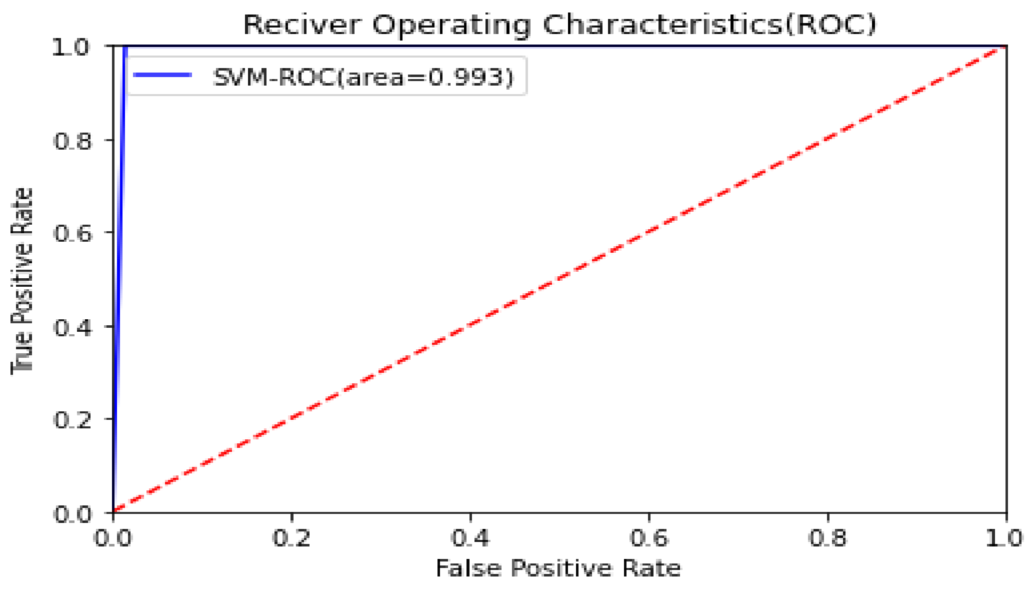


Figure 4. 6*ROC analysis for KC1 Dataset*

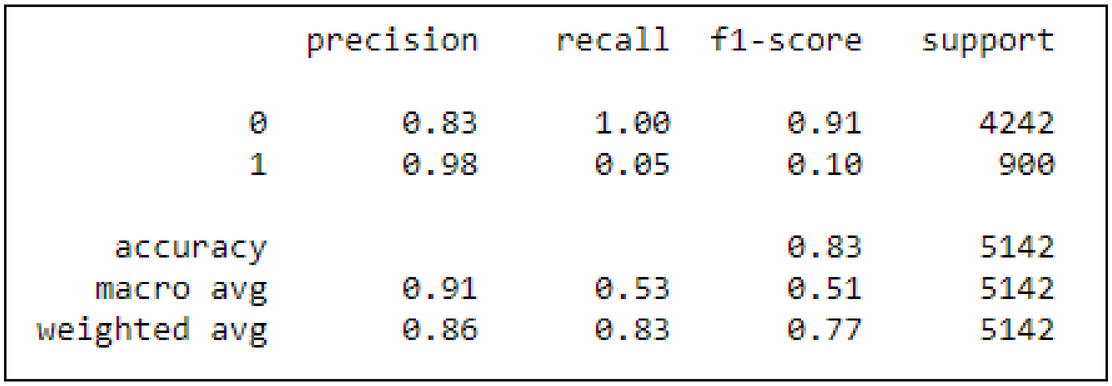
4.6

In the graph above, the ROC-AUC for the blue curve is 0.993which is very close to 1 meaning the proposed model is better at achieving a combination of precision and recall onthe KC1 dataset.

4.6.5 Experimental Result for PC5 Dataset

i. Statistical performance analysis

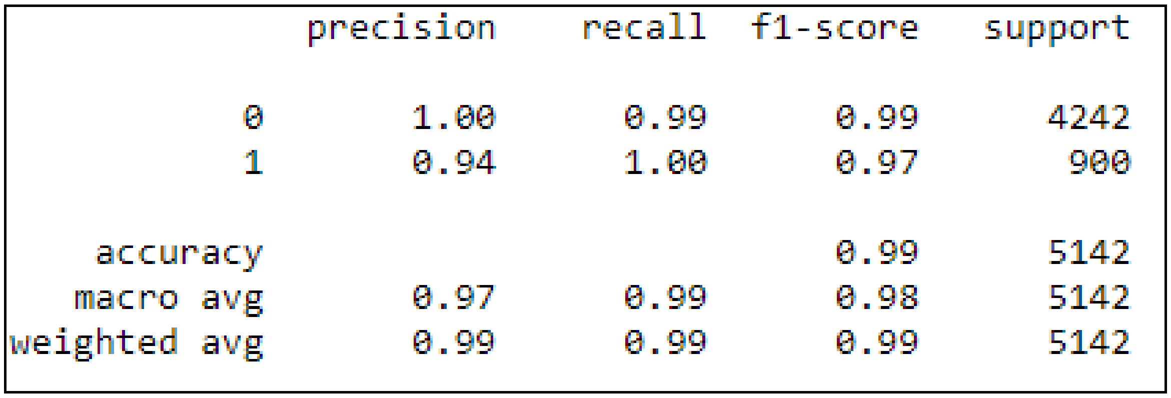
Table 4.17 shows the result of the statistical performance for the PC5 dataset before data preprocessing.We used 5142 data from both the majority and minority class to test the proposed defect prediction model. As shown in the table, it is reflected that before data preprocessing the classification result of the minority class in precision, recall, and F1-Score are 0.98, 0.5, and 0.1 respectively. Whereas the majority class is performed better than the minority class. This is due to that it is affected by imbalancethat the recall and F1-Score results of thePC5 dataset are low.



```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91      4242
           1       0.98      0.05      0.10       900

    accuracy                           0.83      5142
   macro avg       0.91      0.53      0.51      5142
weighted avg       0.86      0.83      0.77      5142
```

Table 4.17 Result of PC5 Dataset before data preprocessing

Table 4.18 shows the results of the PC5 dataset using the proposed defect prediction model. As we presented in table 18, the precision, recall, and F1-Score value of the minority class is 94%, 100%, and 97% respectively. Now, this shows that the recall and F1-Score value of the minority class is proportionally predicted with the majority class instances. Therefore, data preprocessing has a greatimprovement for the classification performance of the prediction model.



```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      4242
           1       0.94      1.00      0.97       900

    accuracy                           0.99      5142
   macro avg       0.97      0.99      0.98      5142
weighted avg       0.99      0.99      0.99      5142
```

Table 4.18 Result of PC5 Dataset after Data preprocessing

ii. Confusion Matrix Analysis for PC5 Dataset

Table 4.19 shows the confusion matrix performance analysis of the proposed CISDP model on PC5 dataset. PC5 is a large software project which has 17,186 software modules (instances). Among the total17,186 data, we used 70% of 12, 030 data for training data, and 30% of 5142 instances for testing data. As displayed results in the confusion matrix, the proposed model achieved results with 4759 true positive instances (TP = 4188). More clearly, 4188 software modules are classified correctly as defect-freedata which is in fact non-defective software modules. Similarly, 900 instances are true negative (TN = 900), which means 900 instances are classified correctly as defective modules which are in fact defective software modules. Totally 5088 instances are correctly classified as intended from the given dataset from their intended class. this means these testing data are classified properly with their corresponding defective and non-defective class. It also shows significant performance in terms of true positive rate.

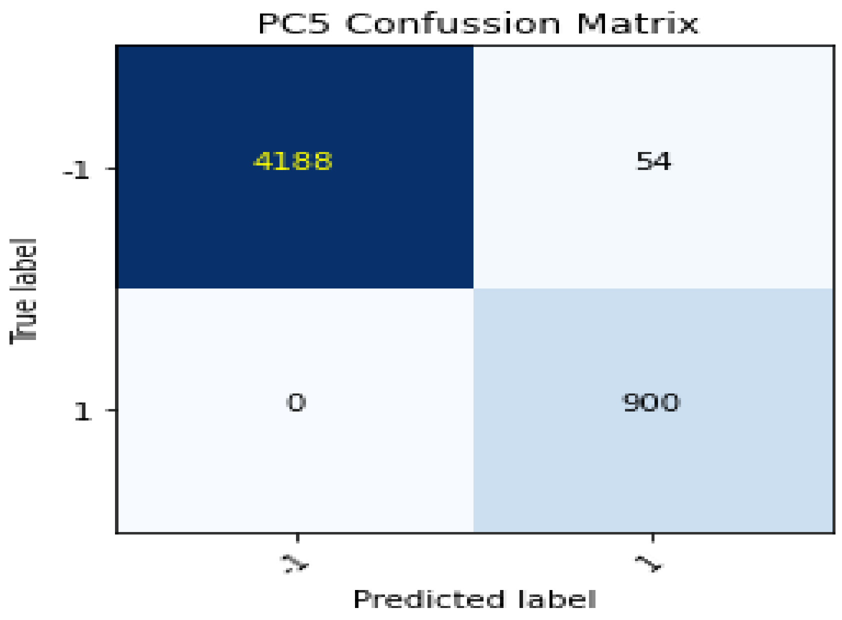


Table 4.19 Confusion matrix for PC5 Dataset

Contrasting 54 instances are false negative which means those data are classified as defective data which is in fact defect-free software modules.Which means those data are non-defective but classified as defective. In addition, false-positive value is zero, which means there no data are classified as defective but classified as a non-defective software module. Therefore, only false positive instances have negatively affected the prediction result.

i. ROC analysis for PC5 Dataset

Figure 4.7 shows the result of the ROC-AUC curve for the PC5 dataset using the proposed defect prediction approach, where the precision rate and recall rate are compared to show the precise performance of the proposed model. As shown in the graph, the blue line in the upper curve represents the proposed model ROC-AUC of the PC5 dataset is 99.4%. This shows the prediction accuracy of the proposed model is higher. This means the value of the true positive rate is greater than the value of the false positive rate. A curve pulled close to the upper left corner indicates a better performing test.

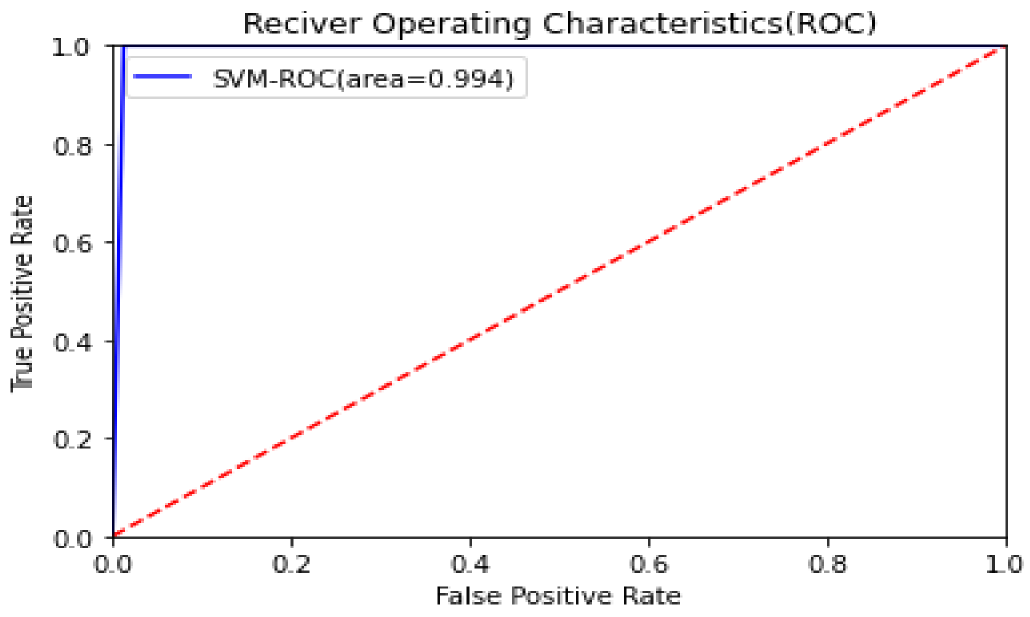
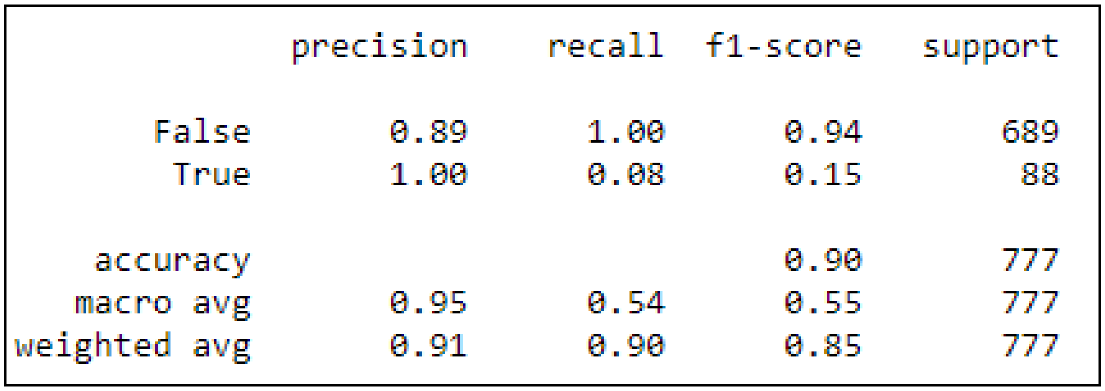


Figure 4. 7ROC analysis for PC5 Dataset

4.6.6 Experimental Result forMC1 Dataset

I. Statistical performance analysis

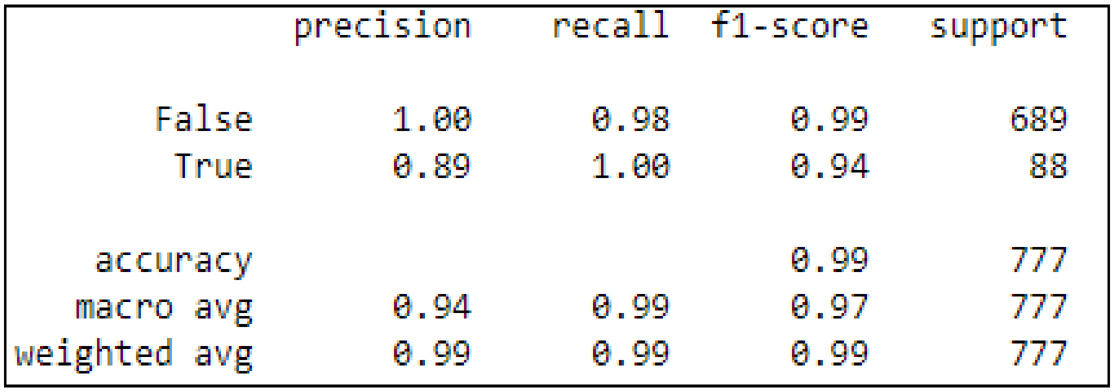
In this section, we present an experimental study on the MC1 dataset using the proposed approach for software defect prediction. The results of theMC1 dataset are shown in Table 4.20, the accuracy of the majority class is high. It is reflected that before data preprocessing the results of the minority class are very low. Moreover, recall and F1-Score value are too low. Whereas the majority class is performed better than the minority class since the recall, and Fl-score value of the minority class 8% and 15% correspondingly.



```
              precision    recall  f1-score   support

       False       0.89      1.00      0.94       689
        True       1.00      0.08      0.15        88

    accuracy                           0.90       777
   macro avg       0.95      0.54      0.55       777
weighted avg       0.91      0.90      0.85       777
```

Table 4.20 Result of MC1 Dataset before data processing

Table 4.21 shows the results of the MC1 dataset using the proposed CISDP model after data preprocessing. As we have seen in the table, the recall, and F1-Score value of the minority class is 0.08 and 0.15 respectively. But Now, It is reflected that the recall and F1-Score value of the minority class is 100% and 94%. This indicates there is very high difference in results when it is compared with the experimental scenario one. It is a great improving than the original MC1dataset.



```
              precision    recall  f1-score   support

       False       1.00      0.98      0.99       689
        True       0.89      1.00      0.94        88

    accuracy                           0.99       777
   macro avg       0.94      0.99      0.97       777
weighted avg       0.99      0.99      0.99       777
```

Table 4. 21 Result of MC1 Dataset after data preprocessing

II. Confusion Matrix Analysis for MC1 Dataset

Table 4.22 shows the confusion matrix performance analysis of the proposed CISDP model on MC1 dataset in which it contains 2,670data. We used 70% 1886 instances for training data and 30% instances for testing data. As displayed results in the confusion matrix, the proposed model achieved results with 678 true positive instances. More clearly, 678 software module are classified correctly as defect free data which is in fact non-defective software modules. Similarly, 88 instances are true negative. Totally 766 instances are correctly classified as intended from the given dataset from their intended class. That means, these testing data are classified properly with their corresponding class.

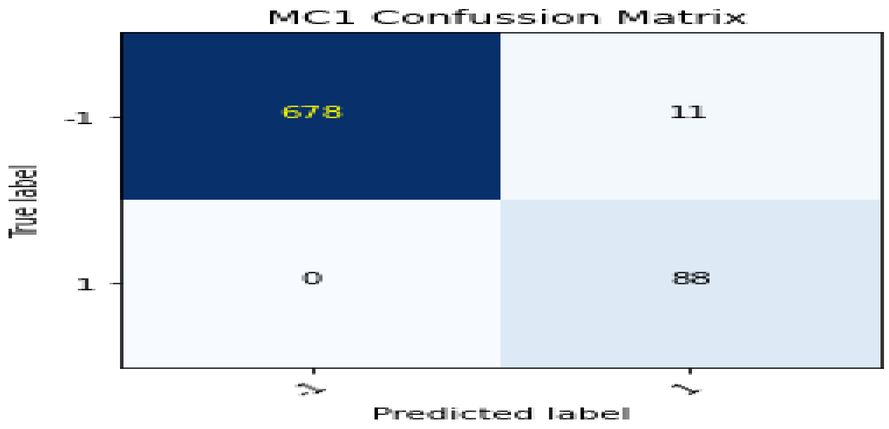


Table 4.22 Confusion matrix for MC1 Dataset

On the other hand, 11 instances are false negative which means those data are classified as defective data which is in fact defect-free software modules. Which means those data are non-defective but classified as defective. In addition, false positive value is zero, which means there no data are classified as defective but classified as non-defective software module. Therefore, only eleven false positive instances have negatively affects the prediction result.

III. ROC Analysis for MC1 Dataset

Figure 4.8 shows the result of ROC-AUC curve for MC1 dataset using proposed defect prediction approach, where the precision rate and recall rate are compared to show the precise performance of proposed model. The blue line in the upper curve represents the proposed model, now it is clear that, the ROC-AUC of the MC1 dataset is equal to 99.2%. This shows that the value of true positive rate is greater than the value of false positive rate.

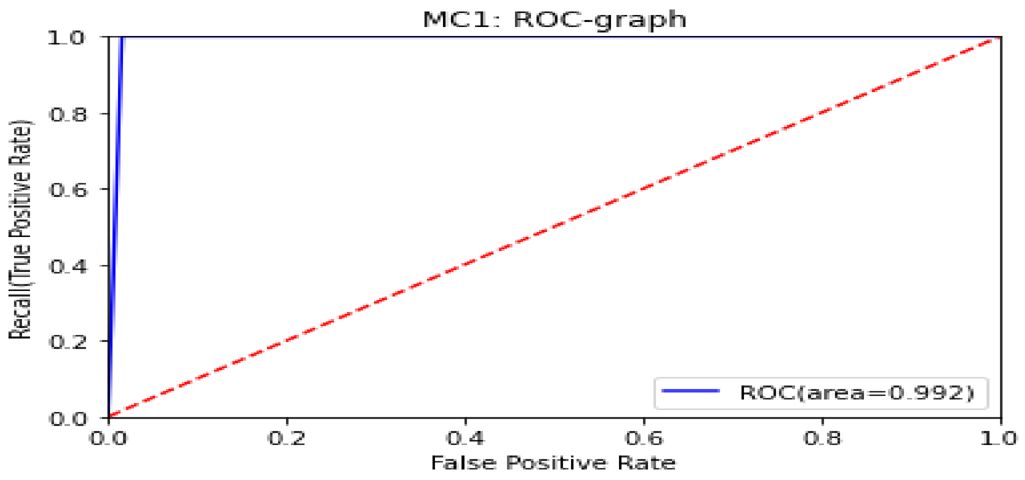


Figure 4. 8ROC curve for MC1 dataset

4.7 Comparison with Existing work

Table 4.23 shows the comparative study for PC3, PC1and KC1 datasets. As shown in the table, it is obvious that the proposed CISDP model achieved better results when compared with the state-of-art models. Thecomparative study shows that the model is skillful to predict defects in software modules since it attainedbetter results. The main reason behind the success of our proposed CISDP model is the practical application of feature selection, data sampling, and kernel function together. Itproduced the uppermost performance using Recall, Precision, and F1-Score performance metrics, unlike the state-of-art models. Because, for this study, the higher value of these software performance metrics is more useful than the higher accuracy.Therefore,we can say that the proposed CISDP model can be used to solve the class imbalance problem effectively, and delivers auspicioussoftware defect prediction result.

Dataset	Techniques	Precision	Recall%	F1-score%	Acc. %
PC3	Software defect prediction using feature selection and random forest algorithm (Ibrahim Rashid, et al, 2017)	--	--	--	94.08
	Empirical study to investigate oversampling methods for SDP using imbalanced data (Ruchika Malhotra, 2019)	86.5	97.6	77.4	--
	Deep neural network based hybrid approach for software defect prediction (Manjula and Florence, 2018)	91.5	91	94	97.76
	The Proposed Approach	**96**	**99**	**98**	**99**
PC1	Software defect prediction using feature selection and random forest algorithm (Ibrahim Rashid, et al, 2017)	--	--	--	97.7

	An empirical study to investigate oversampling method for SDP using imbalanced data (Ruchika, Kamal, 2019)	94.1	96.7	77.6	96.3
	Establishing a software defect prediction model via effective dimension reduction (Changzhen, et al, 2018)	82	94.11	87.67	95
	The proposed Approach	**97**	**99**	**98**	**99**
KC1	Establishing a software defect prediction model via effective dimension reduction (Changzhen, et al, 2018)	--	--	88	89.3
	SDP via cost-sensitive Siamese parallel fully-connected neural networks (Zhang Tang, et al, 2019)	**--**	**--**	91.2	88.8
	Deep neural network based hybrid approach for software defect prediction (Manjula and Florence, 2018)	90.22	93.2	96	97.8
	The proposed Approach	**95**	**99**	**97**	**99**

Table 4.23 performance comparison for PC3, PC1and KC1 dataset

Table 4.24 shows the comparative analysis for JM1, MC1, and PC5 dataset with state-of-art models. From the table, it is obvious that the proposed CISDP model achieved better results when compared with the state-of-art models. The comparative analysis shows that the model is skilled to predict defects in software modules since it attained better results. Specifically, in the JM1 dataset, an experimental study shows that the CISDP model produced the uppermost result using Accuracy, Recall, Precision, and F1-Score performance metrics. This is due to the fact that the JM1 dataset has an adequate instance of data to train the model.The main reason behind the success of our proposed CISDP model is the practical application of feature selection, data sampling, and kernel function together. This system delivers reliable and significant classification performance which can be used for the class imbalance software defect prediction model. In addition, the Recall value of the proposed system is higher than the other performance metrics. This is amore important result for imbalanced data because in real life the cost of false-positive is higher than other metrics. Therefore, we can say that the proposed CISDP model can be used to solve the class imbalance problem effectively, and delivers promising defect prediction result.

Dataset	**Techniques**	**Precision**	**Recall**	**F1-score**	**Acc.%**
PC5	Feature selection in software defect prediction: (Kakkar, 2016)	**--**	**--**	**--**	81
	An empirical study to investigate oversampling methods for SDP using imbalanced data (Ruchika, Kamal, 2019)	**97**	95.3	**--**	96.3

	The proposed Approach	**97**	**99**	**98**	**99**
JM1	Feature selection in software defect prediction: A comparative study (Kakkar, 2016)	--	--	--	81.66
	SDP via cost-sensitive Siamese parallel fully-connected neural networks (Zhang Tang, et al, 2019)	92	91	94	87.0
	An empirical study to investigate oversampling methods for SDP using imbalanced data (Ruchika Malhotra, 2019)	97	94.6	50	--
	The proposed Approach	**100**	**100**	**100**	**100**
MC1	An empirical study to investigate oversampling methods for SDP using imbalanced data (Ruchika Malhotra, 2019)	**94**	94.6	91	83.4
	Establishing a software defect prediction model via effective dimension reduction (Changzhen, et al, 2018)	73.91	94.44	82.93	86.3
	The proposed Approach	**94**	**99**	**97**	**99 99 99 99 9**

Table 4.24 performance comparison for PC5and JM1dataset

4.8 Comparisonwith ML Classifier

In this section, the performance of the proposed software defect predictor is compared with the commonly used machine learning classification algorithms for defect prediction.

Figure 4.9, presents a graphical performance analysis for software defect prediction. This figure shows a comparative analysis of DT, LR, RF, NB, etc., and proposed SVM classification. The graph shows the performance of the different ML classifiers based on the classification accuracy in terms of ROC-AUC scores attained by all classifiers. As shown in the graph, SVM and DT achieved better classification ROC score. From the graph, it is clear that the proposed approach using feature selection and sampling technique gives better accuracy when compared to other algorithms. Whereas NB, RF, and LR achieved ROC score 52.7%, 63.2%, 71.4% on PC5 datasets, which is plotted in red, yellow, and black lines. The green line represents the DT classifier achieved a ROC score of 95.1%, which attained good classification result next to the SVM. However, those standard classifiers have less classification accuracy than the proposed defect prediction approaches. From Figure 4.8, the blue line in the upper curve represents the proposed model, the ROC-AUC score of PC5 dataset is 99.4%. Therefore, it is clear the proposed approaches using the SVM classifier is the dominant classifier than the other ML classifier. In general, the comparative analysis shows that the proposed software defect prediction approaches obtained the highest classification accuracy for all the six NASA datasets than other ML algorithms.

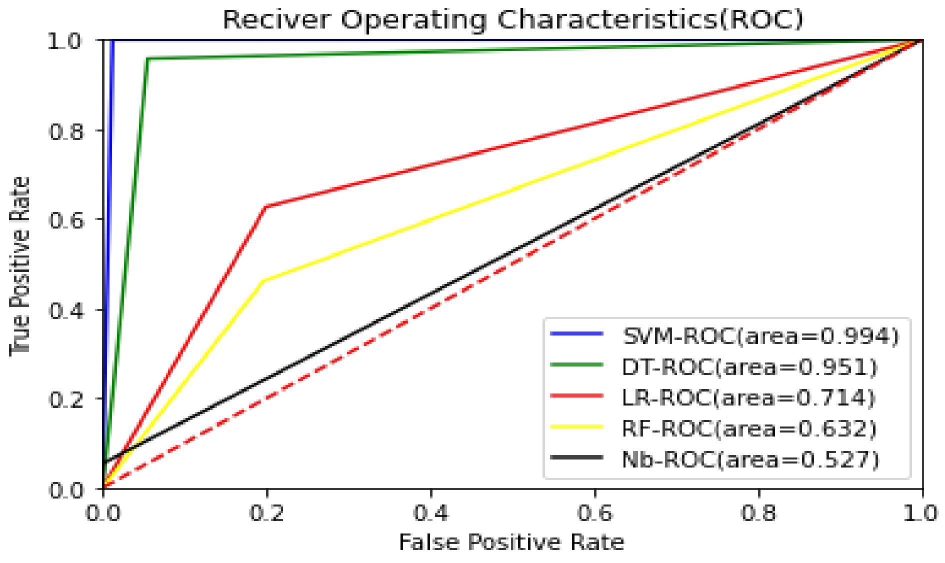


Figure 4. 9Comparison of Basic ML Classifier Performance

CHAPTER FIVE

Conclusion and Future Work

5.1 Conclusion

In software engineering, it is a common practice to predict defective software classes (modules) for the next release of the software product to effectively allocate the software testing resources. Identification of defect prone classes can help to prioritize the software classes for testing. This may lead to the saving of resources and the cost of the system. Therefore, the prediction of defects in software modules is a hot and vital activity in the software engineering community.

In a certain binary classification problem, selecting the relevant and significant features is a great challenge when the data has unequal class distribution and high dimensional. In this study, we present a software defect prediction model using feature selection and SMOTE sampling techniques on class imbalanced data. Feature selection is applied to deal with a selection of most relevance and important features with respect to the predictive actual class and SMOTE sampling techniques are applied to the training data to employ with the class imbalance problem.

The prediction result of proposed CISDP model is nearly perfect. The secret behind the uppermost accuracy is the application RBF kernel function that enables the SVM classifier to maximize the optimal marigion between the minority and the majority class.

The conclusions of this study entail that selecting the right set of software features (attributes) for class imbalance software defect prediction is very important and it is a critical accomplishment. From a practical point of this investigation, working with a smaller set of features for defect prediction modeling is more effective than working with a large number of software features.

Essential Complexity, ***Halstead Content***, ***Halstead Level***, ***Maintenance Severity***, ***Halstead Effort***, ***Number Of Unique Operator And Operands***are the most significant features of PC3, PC5, and MC1 datasets. From this, ***Essential Complexity*** *is one of* McCabe metrics feature.

The proposed method is applied to six NASA datasets with the context of defect prediction. Experimental results shows that the proposed method achieved better classification results. Therefore, we can conclude that the proposed approaches improve the classification performance of SDP approaches, and it provides a brand new way of dealing with the imbalanced data problem.

5.2 Contributions

The main contribution of this study is application of filtering feature selection and SMOTE sampling together that enables proposed model effectively solve the binary classification faults from both the minority and the majority class equally. The evaluation of the model shows that an impressive improvement is achieved in defect prediction performance when compared with the state-of art models on imbalanced software defect datasets. this research has three main contributions: scientific, organizational and resource optimization contribution.

Scientific:thisstudyhas a vital scientific contribution to the software engineering research field.

- ☞ We proposed a class imbalance software defect prediction (CISDP) model using synthetic minority oversampling and filter feature selection techniques that can effectively solve class imbalance problem of defect prediction.The proposed CISDP model is quite intelligent to handle binary classification challenges of software defect prediction.
- ☞ We demonstrate that filter feature selection techniques have a great impact on the performance of the defect prediction models, and it is verified that the noise and redundant defect data that is generated during feature extraction has a large negative impact on the performance of software defect prediction models.
- ☞ we investigate the impact of SMOTE methods in defect prediction process on an imbalanced dataset and analyze the performance and interpretation of the result to the goal of improving the performance of defect prediction model.

Organizational contribution: using the proposed CISDP model, software companies can deliver high quality and consistent software products to the customer. the proposed approaches have achieved better results in our experiments on NASA datasets, we believe that it could be used for a quick software defect prediction successfully, and it provides a great advantage for software enterprises to provide reliable software products prior to delivering to the customer.

Resource optimization: the proposed software defect prediction model is primarily supporting software developers, testers, and project managers to take effective decisions on the test resource allocation and assessment of software development. Moreover, early identification of defects can help project managers to handle the improper allocation of resources for testing and maintenance.

5.3 Future work

In this thesis, the proposed CISDP model can be used for the prediction of defects in software modules at the early stage of the software products before delivers it to the customer. Since it achieved better performance, we can say that the proposed model is fit to the existing requirements. However, there are two issues, and to provide possible scenes for future works.

The first is the class imbalance issue. It has been clearly showing in section 4.6 that the class imbalance problem has a great negative influence on the performance of defect predictors. Therefore, we will apply the proposed prediction approaches for other disciplines that are affected by class imbalance problems like natural language processing and image processing.

The second is we will apply the proposed CIDSP model for cross-project software defect prediction that will improve the classification performance of cross-project software defect prediction models.

References

Garcia, et al. (2012). On the effectiveness of preprocessing methods when dealing with different levels of class imbalance. *Knowledge-Based Systems 25*, 13–21.

Khoshgoftaar, et al. (2010). Attribute Selection and Imbalanced Data Problems in Software Defect Prediction. *22nd International Conference on Tools with Artificial Intelligence* (pp. 137-146). IEEE.

Tanujit , et al. (2019). Hellinger Net : A Hybrid Imbalance Learning Model to Improve Software Defect Prediction. *Software Engineering*, 45-51.

Aditya Krishna, et al. (2020, 5 21). *Offshare Software Testing*. Retrieved from A SACKSOFT COMPANY: https://www.360logica.com/blog/difference-between-defect-error-bug-failure-and-fault/#:~:text=%E2%80%9CA%20mistake%20in%20coding%20is,requirements%20then%20it%20Is%20Failure.%E2%80%9D&text=In%20other%20words%20Defect%20is,in%20the%20context%20of%20testing

Agarwal, Tomar. (2014). A Feature Selection Based Model for Software Defect Prediction. *International Journal of Advanced Science and Technology*, 2005-2012.

Ahmed, Umair. (2019). Performance Analysis of Machine Learning Techniques on Software Defect Prediction using NASA Datasets. *International Journal of Advanced Computer Science and Applications,*, 300-309.

Arora, I. (2015). Open Issues in Software Defect Prediction. *International Conference on Information and Communication Technologies*.

Bergmane, et al. (2017). A Case Study: Software Defect Root Causes. *Information Technology and Management Science*, 54-57.

Bushra Hamid, et al. (2015). Using Smote for Convalescing Software Defect Prediction. *Journal of Applied Environmental Science*, 53-61.

Changzhen, et al. (2018). Establishing a software defect prediction model via effective dimension reduction. *Elsevier Information Sciences*, 400-411.

CHEN, et al. (2016). SOFTWARE FAULT PREDICTION BASED ON ONE-CLASS SVM. *Proceedings of the 2016 International Conference on Machine Learning and Cybernetics* (pp. 1003-1008). South Korea: IEEE.

Cholmyong Pak, et al. (2017). An Empirical Study on Software Defect Prediction Using over sampling by SMOTE. *International Journal of Software Engineering*, 811-819.

Daskalantonakis. (1992). A Practical View of Software Measurement and Implementation Experiences Within Motorola. *IEEE Transactions on Software Engineering*, 999-1011.

David, et al. (2018). The Misuse of the NASA Metrics Data Program Data Sets for Automated Software Defect Prediction. *Computer Science*, 96-102.

Francisco Navarro. (2011). A dynamic over-sampling procedure based on sensitivity for multi-class problems. *Pattern Recognition 44 3*, 1821–1833.

Haixiang, et al. (2017). Learning from class-imbalanced data: Review of methods and applications. *Expert Systems With Applications*, 37-43.

Haonan Tonga, et al. (2018). Software defect prediction using stacked denoising autoencoders and twostage ensemble learning. *Information and Software Technology*, 94–111.

Haonan, et al. (2018). Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. *Science & Technology on Reliability & Environmental Engineering*.

Hongyu & Zhang. (2007). Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering*, 636-642.

Hualong, Zhao. (2013). ACOSampling: An ant colony optimization-based undersampling method for classifying imbalanced DNA microarray dat. *Neurocomputing 101*, 309–318.

Ibrahim Rashid, et al. (2017). Software Defect Prediction using Feature Selection and Random Forest Algorithm. *International Conference on New Trends in Computing Sciences* (pp. 252-258). IEEE.

IEEE. (2010). IEEE Standard Classification for Software Anomalies. *Software and System Engineering*.

Jointha. (2019, december 21). *testing class.com*. Retrieved from testing class.com: https://www.softwaretestingclass.com/top-10-reasons-why-there-are-bugs-defects-in-software/

Justin M and Taghi M. Khoshgoftaar. (2019). Survey on deep learning with class imbalance. *journal of Big Data*, 67-89.

Kalaivani, B. (2018). Overview of Software Defect Prediction using Machine Learning Algorithms. *International Journal of Pure and Applied Mathematics*.

Kumar Pandey, et al. (2018). Software Bug Prediction Prototype Using Bayesian Network Classifier: A Comprehensive Model. *Procedia Computer Science*, (pp. 1412-1421).

Liang & Yang. (2011). Learning from imbalanced data sets with a Min-Max modular support vector machine Algorithm. *Electric Electronic Engineering*, 56–71.

Linchang, et al. (2019). Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks. *Neurocomputing*, 65-71.

Logan Perreault, et al. (2018). Using Classifiers for Software Defect Detection. *Software Egineering*.

Manjula and Florence. (2018). Deep neural network based hybrid approach for software defect prediction using software metrics. *Cluster Computing*.

Menzies & Greenwald. (2007). Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*.

Misha & Sarika. (2016). Feature Selection in Software Defect Prediction: A Comparative Study. *6th International Conference - Cloud System and Big Data Engineering* (pp. 658-663). IEEE.

Misha Kakkar, S. J. (2016). Feature selection in software defect prediction: A comparative study. *International Conference - Cloud System and Big Data Engineering*.

Mohammad, K. (2019). Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques. *Journal of Software Engineering and Applications*.

Munir Ahmad, et al. (2019). Performance Analysis of Machine Learning Techniques on Software Defect Prediction using NASA Datasets. *International Journal of Advanced Computer Science and Applications*, 300-310.

Noreen Kausar, et al. (2016). A Review of Classification Approaches Using Support Vector Machine in Intrusion Detection. *International Journal of Advanced Research in Computer Engineering*.

Phuoc Huhyn, et al. (2019). Novel hybrid DCNN–SVM model for classifying RNA-sequencing gene expression data. *Journal of Information and Telecommunication*, 2475-1847.

Prasanth, et al. (2017). A critical review on feature selection techniques in software defect prediction. *International Journal of Pure and Applied Mathematics*.

Rana & Tarhan. (2018). Early software defect prediction: A systematic map and review. *The Journal of Systems & Software Engineering*, 216–239.

Rashid Ibrahim, et al. (2017). Software Defect Prediction using Feature Selection and Random Forest Algorithm. *International Conference on New Trends in Computing Sciences*, 252-259.

Ruchika, Kamal. (2019). An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data. *Elsevier Neurocomputing*, 121-130.

Satria Wahono, et al. (2014). Metaheuristic Optimization based Feature Selection for Software Defect Prediction. *JOURNAL OF SOFTWARE Engineering*, 1325-1332.

Seliya, N. (2010). Attribute Selection and Imbalanced Data:Problems in Software Defect Prediction. *International Conference on Tools with Artificial Intelligence*.

Shabrina Choirunnisa, B. M. (2019). Software Defect Prediction using Oversampling Algorithms. *IEEE* .

Shuib Basri, et al. (2019). Performance Analysis of Feature Selection Methods in Software Defect Prediction. *Journal of Applied Science*, 276-283.

Shuo, W. (2013). Using Class Imbalance Learning for Software Defect Prediction. *IEEE TRANSACTIONS ON RELIABILITY*.

Taghi, Kehan. (2009). Feature Selection with Imbalanced Data for Software Defect Prediction. *International Conference on Machine Learning and Applications*, 235-243.

Tao WANG, W. (2011). Software Defect Prediction on Classifier Ensemble method. *Journal of Information & Computational Sciences*.

Thant, Nyein . (2015). Software Defect Prediction using Hybrid Approach. *Information Technlogy*, 261-270.

Tian, H. (2018). An Empirical Study on Software Defect Prediction Using SMOTE. *International Journal of Software Engineering*.

Turabieh Hamza, et al. (2019). Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. *Expert Systems With Applications*, 29-38.

Vaseem, et al. (2020). Software Quality Prediction Using Machine Learning Application. *Smart Intelligent Computing and Applications*, 87-93.

Wang, H. (2017). Imbalanced Data Processing Model for Software Defect predicyion. *College of Information Engineering*.

Xiao, H. (2018). An Empirical Study on Software Defect Prediction Using Oversampling. *International Journal of Software Engineering*.

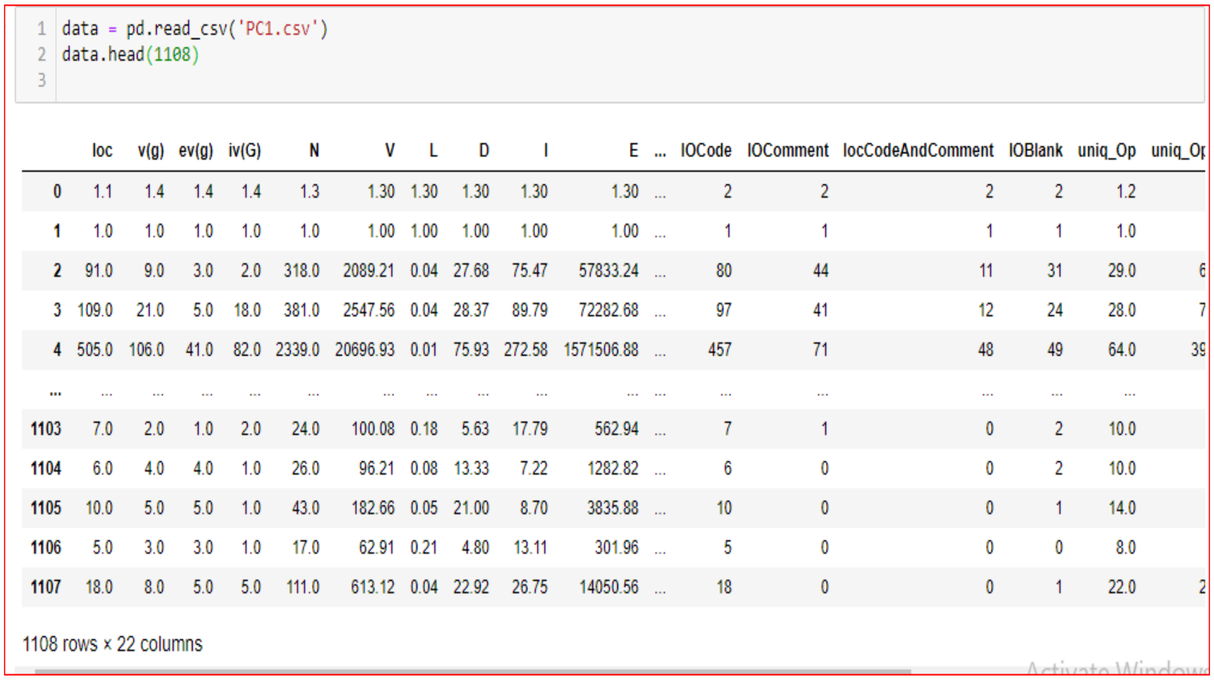
Xiao-Xiao Niu, Ching Suen. (2012). A novel hybrid CNN–SVM classifier for recognizing handwritten digits. *Pattern Recognition 45*, 318–1325.

Xuan, et al. (2019). On cost-effective software defect prediction: Classification or ranking? *Neurocomputing 363*, 339–350.

yeshinkuta, W. (2018). The Impact of Using Regression Models to Build Defect Classifiers. *IEEE Software engineering*.

Zhang Tang, et al. (2019). Software defect prediction via cost-sensitive Siamese parallel fully connected neural network. *ELSEVIER Neurocomputing*.

Appendix One: Dataset and values characterstics

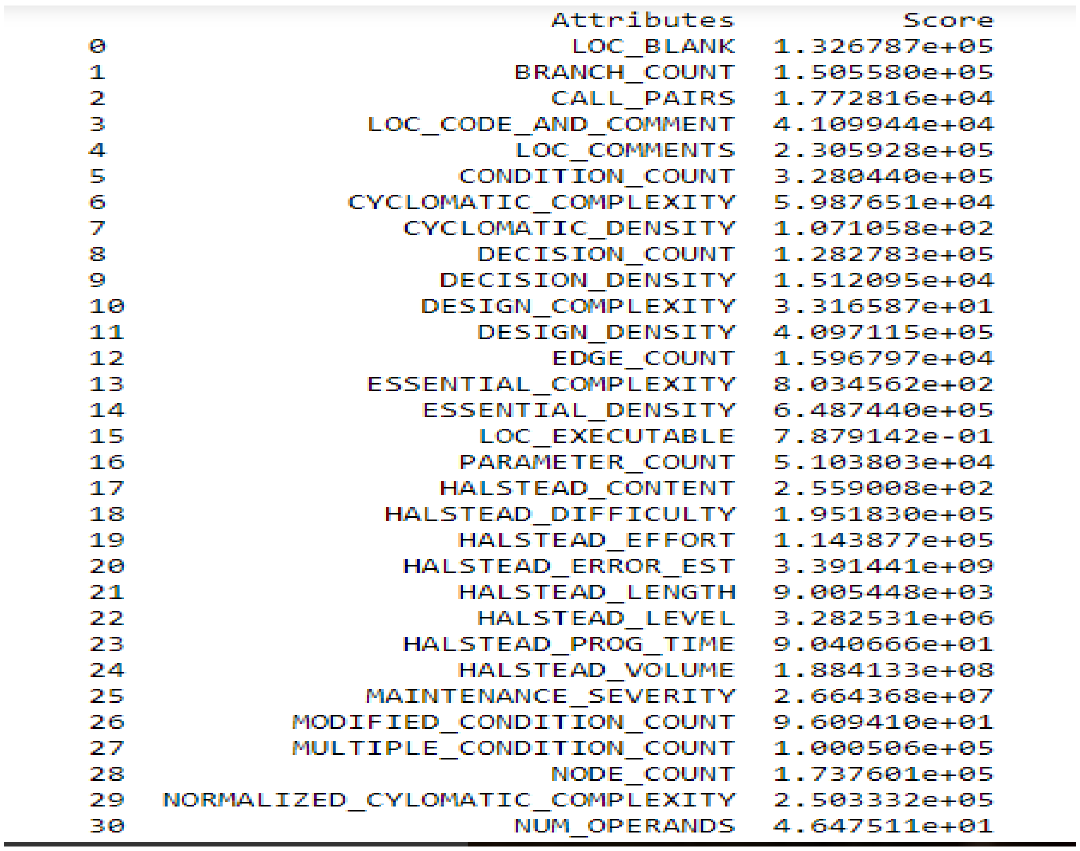


```python
data = pd.read_csv('PC1.csv')
data.head(1108)
```

	loc	v(g)	ev(g)	iv(G)	N	V	L	D	I	E	...	IOCode	IOComment	locCodeAndComment	IOBlank	uniq_Op	uniq_Op
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	1.2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	1.0	
2	91.0	9.0	3.0	2.0	318.0	2089.21	0.04	27.68	75.47	57833.24	...	80	44	11	31	29.0	6
3	109.0	21.0	5.0	18.0	381.0	2547.56	0.04	28.37	89.79	72282.68	...	97	41	12	24	28.0	7
4	505.0	106.0	41.0	82.0	2339.0	20696.93	0.01	75.93	272.58	1571506.88	...	457	71	48	49	64.0	39
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1103	7.0	2.0	1.0	2.0	24.0	100.08	0.18	5.63	17.79	562.94	...	7	1	0	2	10.0	
1104	6.0	4.0	4.0	1.0	26.0	96.21	0.08	13.33	7.22	1282.82	...	6	0	0	2	10.0	
1105	10.0	5.0	5.0	1.0	43.0	182.66	0.05	21.00	8.70	3835.88	...	10	0	0	1	14.0	
1106	5.0	3.0	3.0	1.0	17.0	62.91	0.21	4.80	13.11	301.96	...	5	0	0	0	8.0	
1107	18.0	8.0	5.0	5.0	111.0	613.12	0.04	22.92	26.75	14050.56	...	18	0	0	1	22.0	2

1108 rows × 22 columns

Appendix Two: Feature Selection



```
                            Attributes         Score
0                            LOC_BLANK  1.326787e+05
1                         BRANCH_COUNT  1.505580e+05
2                           CALL_PAIRS  1.772816e+04
3                 LOC_CODE_AND_COMMENT  4.109944e+04
4                         LOC_COMMENTS  2.305928e+05
5                      CONDITION_COUNT  3.280440e+05
6                CYCLOMATIC_COMPLEXITY  5.987651e+04
7                   CYCLOMATIC_DENSITY  1.071058e+02
8                       DECISION_COUNT  1.282783e+05
9                     DECISION_DENSITY  1.512095e+04
10                   DESIGN_COMPLEXITY  3.316587e+01
11                      DESIGN_DENSITY  4.097115e+05
12                          EDGE_COUNT  1.596797e+04
13                ESSENTIAL_COMPLEXITY  8.034562e+02
14                   ESSENTIAL_DENSITY  6.487440e+05
15                      LOC_EXECUTABLE  7.879142e-01
16                     PARAMETER_COUNT  5.103803e+04
17                    HALSTEAD_CONTENT  2.559008e+02
18                 HALSTEAD_DIFFICULTY  1.951830e+05
19                     HALSTEAD_EFFORT  1.143877e+05
20                  HALSTEAD_ERROR_EST  3.391441e+09
21                     HALSTEAD_LENGTH  9.005448e+03
22                      HALSTEAD_LEVEL  3.282531e+06
23                  HALSTEAD_PROG_TIME  9.040666e+01
24                     HALSTEAD_VOLUME  1.884133e+08
25                MAINTENANCE_SEVERITY  2.664368e+07
26            MODIFIED_CONDITION_COUNT  9.609410e+01
27            MULTIPLE_CONDITION_COUNT  1.000506e+05
28                          NODE_COUNT  1.737601e+05
29    NORMALIZED_CYLOMATIC_COMPLEXITY  2.503332e+05
30                        NUM_OPERANDS  4.647511e+01
```