Software Defect Prediction Using Hybrid Machine Learning Techniques

Dita Abdujebar Abrahim

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

June, 2022

Adama, Ethiopia

Software Defect Prediction Using Hybrid Machine Learning Techniques

Dita Abdujebar Abrahim

Advisor: Dr. Bahiru Shifaw

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

June, 2022

Adama, Ethiopia

DECLARATION

I hereby declare that this Master Thesis entitled ***"Software Defect Prediction Using Hybrid Machine Learning Techniques"*** is my original work. That is, it has not been submitted for the award of any academic degree, diploma or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Dita Abdujebar Abrahim	_____	_____
Name of the student	Signature	Date

RECOMMENDATION

I, the advisor(s) of this thesis, hereby certify that I have read the revised version of the thesis entitled ***"Software Defect Prediction Using Hybrid Machine Learning Techniques"*** prepared under my guidance by **Dita Abdujebar** submitted in partial fulfillment of the requirements for the degree of Master's of Science in Computer Science Engineering. Therefore, I recommend the submission of revised version of the thesis to the department following the applicable procedures.

Dr. Bahiru Shifaw	_____	_____
Major Advisor	Signature	Date
_____	_____	_____
Co-Advisor	Signature	Date

APPROVAL PAGE

I, the advisors of the thesis entitled ***"Software Defect Prediction Using Hybrid Machine Learning Techniques"*** and developed by **Dita Abdujebar**, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Dr. Bahiru Shifaw _____ _____

Major Advisor Signature Date

_____ _____ _____

Co-Advisor Signature Date

We, the undersigned, members of the Board of Examiners of the thesis by Dita Abdujebar have read and evaluated the thesis entitled ***"Software Defect Prediction Using Hybrid Machine Learning Techniques"*** and examined the candidate during open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Computer Science Engineering.

_____ _____ _____

Chairperson Signature Date

_____ _____ _____

Internal Examiner Signature Date

_____ _____ _____

External Examiner Signature Date

Finally, approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____ _____ _____

Department Head Signature Date

_____ _____ _____

School Dean Signature Date

_____ _____ _____

Office of Postgraduate Studies, Dean Signature Date

ACKNOWLEDGMENTS

In the name of **Allah**, the Most Gracious and the Most Merciful **Alhamdulillah**, all Praises to **Allah** for the strengths and His blessing in completing this research. Then, my special thanks go to my advisor, **Dr. Bahiru Shifaw** for his friendly approach, readiness, and willingness to advise on the thesis, his critical comments greatly improved this thesis.

Also, I would like to acknowledge the members of the Smart Software SIG of the CSE department.

Including the SIG supervisor **Dr. Mesifn Abebe** for his continued commitment to guide and support this research from its inception to completion.

Next, I really thank **Dr. Biruk Yirga** head of CSE department for encouraging me during this work when difficult humanity problem was faced me, by supporting me in idea and advise. **Dr. Biruk** may God bless you for all your kindness.

Afterthat, I would like to thank all my classmate students who passed good time with me during MSC studies. Also, I would like to thanks my best friend **Mr. Dedefo Bona** who support and advise me as his brothers until the completion of my thesis.

Finally, I want to express my gratitude to all my family members, especially, my mother for her successful responsibility by educating and supporting me, thanks for wishing me a bright future in every situation. May God bless you, wish you a long life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ACRONYMS

AdaBoost - Adaptive Boosting

AEEEM- Apache Lucene, Eclipse Equinox, Eclipse JDT, Eclipse PDE and Mylyn

AET- Adaptive boosting with Extra tree base learner

AUC-Area Under ROC Curve

BET- Bagging with Extra tree base learner

CFS- Correlation Based Feature Selection

CO- Correlation Filter

DBN - Deep Belief Network

DPDF – Deep Forest Defect Prediction

DT - Decision Tree

EM- Expectation-Maximization

ET- Extra Tree

FCBF- Fast correlation-based feature selection

FN - False Negative

FP - False Positive

FS- Feature Selection

GB - Gradient Boosting

GcForest- Multi-Grained Deep Forest

KNN- K-Nearest Neighbour

LC-Apache Lucene

LR- Logistic Regression

MBF- Markov blanket filter

MDP- Metrics Data Program

MLP-Multilayer Perceptron

ML-Mylyn

mRmR- Minimum redundancy, maximum relevance

NB - Naïve Bayes

PLS- Partial Least Square Regression

RF - Random Forest

RFE- Recursive Feature Elimination

SBE- Sequential backward elimination

SDF- Stacking by default final meta learner with base weak learners such as ET, RF and DT.

SDLC – Software Development Life Cycle

SDP - Software Defect Prediction

SFP - Software Fault Prediction

SFS-Sequential Forward Selection

SHAP- SHapley Addictive explanation

SMOTE- Synthetic Minority Over-Sampling Technique

SQA - Software Quality Assurance

SVM - Support Vector Machine

TN - True Negative

TP - True Positive

XGBoost - Extreme Gradient Boosting

ABSTRACT

Various researchers tried to develop methods of software defect prediction through applying different machine learning algorithms. However, the performance of those techniques on most publically available defect datasets is far from satisfactory. This is because of defect datasets which mostly are affected by two main problems such as high feature dimensionality and class imbalance. Five AEEEM projects of software defect datasets namely EQ, JDT, LC, ML and PDE are used in this research and affected by high feature dimensionality and class imbalance problem. To solve these problems, in this research proposed software defect prediction models using seven ensemble machine learning algorithms such as AdaBoost, GB, XGBoost, RF, ET, Bagging and Stacking with base classifier. And three feature selection methods namely CFS, SFS and CO for solving problem of high feature dimensionality and SMOTE data balancing technique to handle class imbalance problem are first used as part of preprocessing methods before implementing the above models. The experiment is performed and evaluated on 10- folds cross validation with performance metrics such as accuracy, recall, precision, F-measure and AUC. Results indicate that CO feature selection with ET ensemble learning algorithm is outperforming as compared to other models on all five datasets. The accuracy results are 93.1 %, 96.3 %, 99.2%, 98.2% and 97.8% for EQ, JDT, LC, ML and PDE data sets respectively. Additionally, other performance metrics have also demonstrated more than 91% for all datasets. Therefore, ET with CO model is recommended for software defect prediction model that classify software modules as defect or non-defect.

Keywords: *Software Defect Prediction, Ensemble Machine Learning, Extra tree(ET), Feature Selection, Correlation Filter(CO), SMOTE*

CHAPTER ONE

1. INTRODUCTION

1.1. Background of Study

In the twenty-one century, science and technology have grown rapidly in every aspects of human life and the life of human being become easy and supported by different technologies (Wan & Bi, 2019). As technology supports our day to day activities, several problems may face human being in different ways. To solve that problem and facilitate the process of life software is very important. Software is most crucial technology that used to solve problems of societies which prevail in different aspects of life and it is developed by using software engineering discipline with different frameworks and languages.

Software engineering is a discipline which used to develop different types of software through software development life cycle process and methods. As software is contributing in daily activities of people, the number of software is increasing. This development procedure should follow discipline of software engineering. That means software development process should be controlled and monitored to obtain quality software at a lower cost. To perform this, it is better to follow the method of Software Quality Assurance (SQA). SQA includes code walkthroughs, software testing, code inspection, and software defect prediction(Adrion et al., 1982; Johnson & Malek, 1988; Santosh S. Rathore & Kumar, 2019).

During development of software, software companies and software developers are facing challenges of software product by software defect problem as a part of process. Software defect is an error, flaw, fault, or bug in a system or source code that causes incorrect outcomes and this bug might happen for the reason of fault in the requirements, design or source code. This challenge affects functionalities of developed software and also confront when using software product. The problem also goes to the work of software testing which is to be performed at later stage though it is probable to clearly identify faults and generally, if identified, it will have significant effect on software quality, reliability and maintenance cost(Rawat & Dubey, 2012). If the faults happen on the testing stage, software development needs additional budget cost to fix the problem and this problem reduce software quality. This impact will make improper cost

and budget utilization of software development. And to deliver quality software for software users, testing software has got attention from tester and developer and this testing become hot topic and fast growing area. However, as software become complex and large the methods of controlling, managing, fixing, reducing and testing a defect of software are difficult (Alsawalqah et al., 2020). Therefore, controlling the defect during any phase of software development life cycle and early detection will support software developers, testers and software companies and deliver quality software(Rawat & Dubey, 2012). This task is performing by software fault prediction (Alsawalqah et al., 2020; Mehta, 2021).

Software Fault Prediction is one of the most supporting activities of the Testing Phase of SDLC. It is an important approach to minimize software development and maintenance costs, and it is a technique purposed to enhance software quality and testing efficiency by preparing timely detection of fault-prone software modules before the actual testing process starts(Alsawalqah et al., 2020; Rawat & Dubey, 2012). In this software fault detection mechanism, software metrics are used as input to describe property of specific software modules and it represented by quantitative form to identify defective software(Mishra & Catal, 2018; Shatnawi & Li, 2008).

Based on Software Bug Prediction, several researchers have studied different techniques to build and evaluate several software fault prediction model using classification, regression, unsupervised method and soon. The methods of fault prediction model that several researchers had used are common, which can be categorized into three namely binary class classification of fault(Alsawalqah et al., 2017, 2020; Zhan Li & Reformat, 2007; Mendes-Moreira et al., 2012; Santosh S. Rathore & Kumar, 2019; Vandecruys et al., 2008), number of bug/fault density prediction(Santosh S. Rathore & Kumar, 2015, 2017, 2019; Santosh Singh Rathore & Kuamr, 2016; Santosh Singh Rathore & Kumar, 2017) and severity of bug prediction(Sandhu et al., 2011; Shatnawi & Li, 2008). From them, mostly used types of prediction scheme is binary class classification, where software modules having one or more faults are distinguished as faulted and modules having zero faults are distinguished as non-faulted.

In this type of bug prediction scheme, most researchers have discovered and used supervised approaches such as ensemble methods(Alsawalqah et al., 2017; Huda et al., 2018; Jiang et al., 2008; Sun et al., n.d.), Decision Tree (DT)(Koprinska et al., 2007), Support Vector Machine (SVM)(Elish & Elish, 2008), Naïve Bayes (NB)(Menzies et al., 2007), Random Forest, Neural Network, Logistic Regression(Dhiauddin et al., 2012). There are also unsupervised techniques

which have been studied in this bug prediction scheme such as Fuzzy clustering(Yuan & Allen, 2000) and K-means clustering(Catal et al., 2009). Semi supervised techniques have been also studied such as Expectation Maximization (EM)(Seliya & Khoshgoftaar, 2007). However, all above mentioned approaches are far from satisfactory performance in classifying defects in the datasets that means by range their accuracy is appeared between 70% and 90% (Alsawalqah et al., 2020; Santosh S. Rathore & Kumar, 2019; Zhou et al., 2019). By different approaches NASA datasets obtained 90% accuracy performance in average (Mehta, 2021; Zhou et al., 2019), PROMISE datasets obtained 80% accuracy performance in average (Zhou et al., 2019), AEEEM datasets obtained 86% accuracy performance in average (A. O. Balogun et al., 2020; Zhou et al., 2019) and Relink datasets obtained 75% accuracy performance in average (Zhou et al., 2019).

Recently most of review studies that in the area of software fault prediction have used ensembles learning to suit more effective and better prediction performance(Aljamaan, 2020; Matloob et al., 2021). The researcher in (Alsawalqah et al., 2017) applied supervised ensemble classifier such as Random Forest, Bagging and Ada boost with basic classifier such as Multilayer Perceptron (MLP), C4.5 Decision Trees and Naive Bayes (NB) on NASA and PROMISE datasets projects and the results show that Adaboost with C4.5 DT is outperform others. The researcher in (Zhou et al., 2019) applied supervised ensemble and deep learning such as Cascade Deep Forest (DPDF is a combination of ensemble and deep learning) on 25 software projects of NASA, PROMISE, AEEEM, Relink datasets and the results show that DPDF is good performance with other comparison. The researcher in (Aljamaan, 2020) applied supervised ensembles classifier that based on the tree such as Random Forest(RF), Extra Trees(ET), Adaptive Boost(AdaBoost), Categorical Boosting(CatBoost), Extreme Gradient Boosting (XGBoost), Gradient Boosting(GB) and Histogram based Gradient Boosting(HGB) on NASA software projects datasets and the results show that RF and ET algorithms are outperform others. The researcher in (Mehta, 2021) applied Ensemble Learning such as Random Forest, Extra Tree, Adaptive Boost, Stacking, XGBoosting, and Bagging on NASA software projects datasets and the results show that XGBoost and Stacking outperform others. The researcher in (Esteves et al., 2020) applied Ensembles such as XGBoosting and Random Forest and another supervised on PROMISE datasets and the results show that XGBoost is good performance with other comparison.

Most publicly available software defect datasets are affected by two main problems such as high feature dimensionality and class imbalance problem (Aljamaan, 2020; Mehta, 2021; Zhou et al., 2019). In high feature dimensionality problem, some features of a datasets are less significant in fault decision and reduce accuracy of the model (A. O. Balogun et al., 2020; Zhou et al., 2019) from datasets more than 50% faults are found in 20% of the features(components) (Aljamaan, 2020), feature selection technique can solve this problem by identifying most relevant set of features and extract irrelevant set of features and in imbalance class distribution problem, the number of defect class instances of a datasets are very few in ratio when compare with non-defect class ratio that needs further solution in software fault prediction model(Aljamaan, 2020; Zhou et al., 2019), class balancing technique are solved imbalance problem. These two problems affect performance of ensemble machine learning classifier.

However, the best technique of solving current software defect prediction problems such as high feature dimensionality and class imbalance are not combined with best algorithms. This means that publicly available software defect datasets are impacted by high features dimensionality and the technique of identifying best set of features with different ensemble algorithms with effective performance will not be extensively covered. Moreover, class balancing to solve a biased classification problem for classifying bad instances well, were not explored more. When software system evolving from one version to another version, building defect prediction model using only code metrics is a another problem (Santosh S. Rathore & Kumar, 2019) this means in versions evolving if changes metrics between a versions, previous defect metrics and other versions updating metrics are not included in input of software defect prediction model and using only code metrics as the input/predictors of software defect prediction model that predict future defect, so that the prediction model is not correctly identify the defect modules and the model performance is low. Additionally, the prediction accuracy of fault prediction approach on software defect datasets of other individual(base) classifier of machine learning algorithms are low, with a high misclassification rate(Alsawalqah et al., 2020; Santosh S. Rathore & Kumar, 2019; Santosh S Rathore & Kumar, 2021).

Therefore, this research work propose software defect prediction approach that uses different types of ensemble algorithms with different types of base classifiers which increase performance of prediction methods with SMOTE class balancing methods and extensive features selection mechanisms as preprocessing stage on AEEEM project datasets of software defect to solve the

main problems of software defect datasets such as high dimensionality by identifying attributes strongly decide the fault and class imbalance by minimizing bias. And the model performance is compared and tested by cross validation and the proposed software defect prediction model is used to classify software modules into defect and non-defect.

1.2. Motivation of the Study

As the software is contributing to the daily activities of human being, the number of software components and versions are increasing with it to keep the customer or user satisfaction by adding new functionality and characteristics. The software metrics computed by concerning versions evolving of software system are large in number and leading to high feature dimensionality problem in case some features are less significant in fault decision. And this problem is causes poor performance on software defect prediction models(A. O. Balogun et al., 2020; Zhou et al., 2019). Therefore, solving high feature dimensionality problem specially exists on AEEEM datasets using different feature selection technique and improving software defect prediction performance is motivated me. Additionally, AEEEM software defect datasets which has high feature dimensionality problem is also affected by class imbalance problem in case of the number of defect class instances of a datasets are very few in ratio when compare with non-defect class ratio (Zhou et al., 2019) and this problem causes biased the classification of software defect prediction model and to solve this problem by using class balancing technique and perfectly classify software modules into defect or non- defect is also motivated me. Generally, above two mentioned, problem is the current problem of software defect prediction which giving bad performance and solving both problems and building improved performance model is another reason that motivated me to propose this research.

1.3. Statement of the Problem

In the development of software, the defects or faults may happen in different stages of the software development life cycle. When developed software is tested if the defect happens at the later stage of software it is very more expensive and difficult to maintain. Also when the defect is undetected in developed software and delivered to the user, the defect dissatisfies the customer or user and brings economic failures. Therefore, to produce quality software early detection of defects is very important before the testing phase begins. This means in any stage of software development the fault is detected and managed to minimize development and maintenance costs and then improve software quality. So, this timely detection fault is performed by the software

defect prediction approach. The software bug prediction approach is a model that classifies software modules into fault and non-fault. This approach is studied by several researchers using several techniques. However, still publicly available software defect datasets have two main problems such as high feature dimensionality and class imbalance problem(Aljamaan, 2020; Mehta, 2021; Zhou et al., 2019) because of these two problems, software defect prediction models are giving a poor performance. The best technique for solving those current software defect prediction problems such as high feature dimensionality and class imbalance is not combined with the best algorithms. This means that publicly available software defect datasets are impacted by high feature dimensionality and the technique of identifying the best set of features is not combined with the best ensemble algorithms that produce effective performance. Moreover, the class balancing technique that solves a biased classification problem for classifying bad instances well is not combined with the best ensemble algorithms. When a software system evolves from one version to another version, software metrics computed from two or more version of software projects includes source code metrics and change metrics between version and previous defects. In the case of publicly available datasets, many researchers use fixed faults that are provided from only source code metrics but when software systems evolve from one version to another version, building a defect prediction model using only code metrics is another problem (Santosh S. Rathore & Kumar, 2019) this means in versions evolving if changes metrics between versions, previous defect metrics and other versions updating metrics are not included in the input of software defect prediction model and using only code metrics as the input/predictors of software defect prediction model that predict future defect so that the prediction model does not correctly identify the defect modules and the model performance is low. Additionally, the prediction accuracy of the fault prediction approach on software defect datasets of another individual (base) classifier of machine learning algorithms is low, with a high misclassification rate(Alsawalqah et al., 2020; Santosh S. Rathore & Kumar, 2019; Santosh S Rathore & Kumar, 2021).

Therefore, this research work proposes a software defect prediction approach that uses different types of ensemble algorithms with different types of base classifiers which increases the performance of prediction methods with SMOTE class balancing methods and extensive features selection mechanisms as preprocessing stage on the AEEEM project datasets of a software defect.

Here are the problems that are going to be addressed in this paper

- Building defect prediction using evolving version of the software system
- Identifying attributes strongly decides the fault by using different feature selection
- Reduce biased training of imbalanced classes by using the class balancing technique
- Identify effective ensembles algorithms on version-changing software systems like the AEEEM project
- Perform different base classifiers on ensembles and identify the best performance
- Building the model of data integrated from different software defect datasets and carrying out its performances.

1.4. Research Questions

This study is planned to obtain an answer to the following four research questions.

- **RQ1.** Is SMOTE data balancing technique improve software defect prediction performance?
- **RQ2.** Which feature selection technique is best for software fault classification/prediction model?
- **RQ3.** What is the best set of features for software fault classification on software defect datasets?
- **RQ4.** Which machine learning algorithms are best for software defect classification/prediction with optimal performance?

1.5. Objectives of the Study

1.5.1. General Objective

The general purpose of this study is to build a model that can detect software defect using ensemble machine learning technique.

1.5.2. Specific Objectives

The following specific objectives are achieved to address the general objective of the study.

- Review different kinds of literature on software defects, techniques of software defect analysis, and behaviors including ensembles machine learning algorithms with different base classifiers and optimization principles

- Apply methods of pre-processing to clean and normalize the datasets to make it appropriate for analysis work.
- Solve class imbalance problem that exist on software defect datasets by using balancing techniques
- Identify feature selection methods that determine relevant set of features in software defect datasets for software defect prediction
- To analyze the structure and behavior of datasets after preprocessing to identify faults and non-faults.
- Develop necessary ensembles machine learning algorithms using evolving version of software system datasets through assigning different effective base classifier to realize the proposed model as a solution
- To evaluate the performance of the designed model and put necessary recommendations and suggestions useful for research works.
- To improve software defect prediction performance.
- Carry out and analysis the performance of software defect prediction models on data integrated from different software defect datasets.

1.6. Scope and Limitations of the Study

1.6.1. Scope of the Study

The scope of this study is to predict fault of software modules that exist in the datasets of version changing software systems like AEEEM project by using different types of ensembles machine learning algorithms with different effective base classifier with k-fold cross validation model evaluation, and SMOTE class balancing methods and extensive features selection mechanisms as preprocessing stage. And the datasets of AEEEM project have software metrics of two or more version of software projects that concern version evolution metrics like source code metrics, change metrics between version and previous defects metrics and soon.

1.6.2. Limitations of the Study

This research work will be delimited itself to the scope mentioned because of limited time as well as absence of sufficient financial support. The time as well as financial offer given to this research work is not that much enough. For this reason, this study does not include all publically available software defect datasets and also it does not include software defect prediction model concerning cross/heterogeneous software defect projects.

1.7. Application of Results

The research result may be applied to the following area:

- Software companies can use this work for testing and development purpose
- Developers and testers of software can use this work to minimize the cost of development and maintenance. It also helps them deliver software products timely
- Researchers or scholars can use this work for further reference.

1.8. Organization of the Thesis

In this section, the organization of the whole thesis report which includes seven chapters is indicated as follows.

Chapter One: describes the introductory part of this study including background and motivation behind this study, statement of the problem that this research going to address, research questions to be answered, objectives to be achieved, scope and limitations of the study and its application area.

Chapter Two: define and discusses the theoretical background of a software defects, definition of software defects from researcher, method of modules classification, software metrics, machine learning, machine learning in software engineering, feature selection, literature related to software defect prediction using ensemble machine learning algorithm.

Chapter Three: demonstrates the source of data and datasets description, software metrics exists in AEEEM projects dataset, various data preprocessing steps including feature selection and data balancing technique, and identifying machine learning models used to construct software fault/defect prediction model, including model evaluations are stated to achieve the goal of this research.

Chapter Four: describes the proposed architecture for software defect prediction. In this section, the ensemble machine learning model architecture for software defect prediction is designed. How to achieve the proposed solution, prototyping, and how it works is also fully stated?

Chapter Five: describes all about the hardware used in this study, the working environment, experimental tools, dataset feature description and implementation of dataset preprocessing, feature selection, data balancing and building machine learning models are implemented. In addition to that, model testing and evaluation implementation is also done.

Chapter Six: describes the obtained results from experimentation and implementation that means dataset preprocessing result, feature selection result, data balancing results and models evaluation results were discussed in detail for all selected ensemble machine learning models with selected feature selections for comparison to figure out the most performing model.

Chapter Seven: is the final chapter that concludes this thesis report, then the recommendation and future research direction of this study are clearly stated.

CHAPTER TWO

2. LITERATURE REVIEW AND RELATED WORKS

This chapter reviews relevant literature to further comprehend and investigate the research problem. This review is concerned about definition of software defects, the reason of software bugs, software metrics, existing techniques used to predict software defect, and related works studied on the problem.

2.1. Software Defects

During the process of software testing, researchers are unable to recognize the difference between software failures, errors, defects, faults, mistakes, wrong, bugs and flaw. To have a wider view of what software defect is, definitions stated and cited by different researchers and sources are compiled and provided below.

2.1.1. Definition of software defects from researchers

- ***Definition 1:*** Software defect is an error, flaw, fault, mistake, failure or bug in a system or source code that causes incorrect outcomes. Software defects are a software bug that might happen for the reason of fault in the requirements, design or source code. That affect software reliability and software quality(Rawat & Dubey, 2012).
- ***Definition 2:*** Software defect elaborates the flaw or failure status that happens at the system or program level which causes to incorrect results or outputs and unexpected process and allows the software to act in an unplanned way(LEVENDAL, 1990).
- ***Definition 3:*** Software defect is unintended result of software project that exist during deployment which losses huge product(Zhou et al., 2019).
- ***Definition 4:*** Software defect is a fault caused by incorrect specification, wrong logic of programming and absence of testing skills in developed software(Alsawalqah et al., 2017).
- ***Definition 5:*** Software fault is an error happened on software product due to incorrect requirement specification and unsuitable configuration development that consequences security polices violation(Abaei & Selamat, 2014).

2.1.2. Method of modules classification

The researchers classify modules/instances of software in three method of categories: binary class classification of fault(Alsawalqah et al., 2017, 2020; Zhan Li & Reformat, 2007; Mendes-Moreira et al., 2012; Santosh S. Rathore & Kumar, 2019; Vandecruys et al., 2008), number of

bug/fault density prediction(Santosh S. Rathore & Kumar, 2015, 2017, 2019; Santosh Singh Rathore & Kuamr, 2016; Santosh Singh Rathore & Kumar, 2017) and severity of bug prediction(Sandhu et al., 2011; Shatnawi & Li, 2008).

From them, mostly used types of classification scheme is binary class classification, where software modules having one or more faults are distinguished as faulted and modules having zero faults are distinguished as non-faulted.

2.2. Software Metrics

Software metrics has been used to describe the complexity of the program and, to estimate software development time. Software metrics explain quantitative measurements of the software product or its specifications(Rawat & Dubey, 2012). These activities scope concern quantitative representation of source code properties up to model representation that used to predict quality of software and resources requirement of software. The concept also contain quality assurance and management represent by quantitative form that deal with controlling, recording, reducing and managing the defect during any phase of software development life cycle and testing(Fenton & Neil, 2000). In simple, software metrics are quantitatively measurement of software features that needed to measuring the performance of software system, measuring productivity, estimating the cost, planning work items, estimating software quality, detecting fault in code(Mehta, 2021).

Software metrics are used as input in software fault detection mechanism and build fault prediction model to identify defective and non-defective software modules and preparing timely detection of fault-prone software modules before the actual testing process starts to enhance software quality. This software metrics are categorized into two major group that are process metrics and product metrics. However, some metrics are used as both process metrics and product metrics(Santosh S. Rathore & Kumar, 2019).

2.2.1. Product Metrics

Product metrics are a group of software metrics that are calculated from different characteristics of developed software project which are used to check software product as it is confirms norms of ISO-9126 standard. This product metrics are categorized into object-oriented, traditional and dynamic metrics.

A. Object-Oriented (OO) metrics: mainly include metrics measurement of software project that developed by object oriented paradigm. Object-Oriented (OO) metrics are include software

product developed by Object Oriented characteristics and this OO metrics proposed by Chidamber and Kemerer(CK) and other researchers but most popular OO metrics are CK metrics suite. CK metrics suite includes Lack of Cohesion in Methods, coupling between Object class, Response for a Class, Number of Children, Depth of Inheritance Tree and Weighted Method Count(Mehta, 2021; Mishra & Catal, 2018; Santosh S. Rathore & Kumar, 2019). Also other OO metrics are such as number of line code, Number of other classes that reference the class, Number of other classes referenced by the class, Number of attributes, Number of public attributes, Number of private attributes, Number of attributes inherited, Number of methods, Number of public methods, Number of private methods and Number of methods inherited(Ambros & Lanza, 2010).

B. ***Traditional metrics:*** are a metrics mainly calculate software complexity and functional size. This traditional metrics includes Size metrics like, Halstead metrics, Quality metrics and System complexity metrics like McCabe Complexity, Structural complexity and Cyclomatic Complexity(Mehta, 2021; Mishra & Catal, 2018; Santosh S. Rathore & Kumar, 2019).

C. ***Dynamic metrics:*** are a metrics collected and measured from executing program and components. Dynamic metrics are opposite of static metrics that are measured from non-executing components. Dynamic metrics are include Yacoub metrics suite like Import Object Coupling and Export Object Coupling and Mitchell metrics suite like Degree of dynamic coupling between two classes at runtime, Dynamic CBO for a class and etc. (Santosh S. Rathore & Kumar, 2019).

2.2.2. Process Metrics

Process metrics are a group of software metrics that are calculated during software development life cycle. This metrics concern everything on software development process. This metrics includes Change metrics, Code churn metrics, Code delta metrics, Developer metrics, Network metrics and Requirement metrics(Santosh S. Rathore & Kumar, 2019).

Above all mentioned software metrics are used in different methods by different researchers in different software defect prediction approaches to enhance efficiency of prediction model.

(Moser, 2008) Compared two set of metrics that used for defect predictor. They have chosen one set of process related metrics called change metrics and one set of product related metrics called code metrics to classify java file of eclipse project whether defective or non-defective. In here, they used only 31 code metrics were used by originator researcher and 18 change metrics

extracted from versioning system of CVS eclipse project repository with annotation of original dataset. After performed experiment they identified that change metrics are more efficient than code metrics in bug prediction at file level.

(Hassan, 2009) compare and predict occurrence of bugs in software system using predictors of entropy(complexity) of change metrics and historical predictor such as prior modification and prior defects. The result shows that complexity change metrics are better predictor of defect than historical predictor by using change history of six project data.

(Ambros & Lanza, 2010) proposed novel approaches and prepare large cover comparison of defect prediction techniques which uses earlier metrics proposed by different researchers in error prediction approaches together with new software metrics computed by novel approaches by using public benchmark of five software system datasets of eclipse and apache that used for defect prediction.

In this approach defects predict work is on the class level not on the package level or subsystem level. Hence, for a given release of software system it has a date of released and the work is to predict the number of post release bugs of every class of released version of the software system from time of released up to six months after released. For all software system they provide extracted data from change versioning log, the bug extracted from bug repository and bi-weekly version of the system in the form of object oriented because all software has written in java.

Based on five software system dataset as a benchmark they used combination of 17 Source code metrics of CK (Chidamber and Kemerer) and other OO (Object Oriented) metrics, 5 previous defects, 5 Entropy of change metrics proposed by (Hassan, 2009), and their own novel approaches namely Churn of Source code metrics and Entropy of Source code metrics to predict number of post release defects. By two their own novel approaches they proposed 17 Churn of Source code metrics and 17 Entropy of Source code metrics by calculating from source code metrics for prediction number of post release defects.

They concluded that experimental results of two novel approaches that they proposed namely LDHH (Linearly Decayed Entropy of source code metrics) and WCHU (Weighted Churn of source code metrics) are obtained optimal performing approach. Additionally, combination metrics of CK+OO, past bug fixes, LDHH, WCHU are obtained very good performance on all five datasets as the whole. And also source code metrics and past defects are simple possibility with whole good performance.

Many other researcher of fault prediction techniques used metrics proposed by Chidamber and Kemerer suite like wmc, dit, rfc, noc and etc. (Ambros & Lanza, 2010; Santosh S. Rathore & Kumar, 2019) and others use other Object Oriented metrics like fanout, fanin, noa, nopa, loc, nom and etc. Some researcher use combination of Chidamber and Kemerer with other Object-Oriented Metrics. And also most of the time several researchers use Line of Code (LOC) metrics because of this metric is optimal in bug prediction (Ambros & Lanza, 2010).

2.3. Machine Learning

Machine learning is an AI (artificial intelligence) application that allows computers to learn and advance on their own without having to be programmed explicitly. Machine learning allows computer programs to access data and utilize it to learn for themselves, and it is concerned with the creation of computer programs. One of the research directions to tackle unpredictable circumstances at runtime is machine learning for modification of adaptation space (Sah, 2020). Machine learning is the subset of AI in the field of computer science, and engineering and is concerned with an algorithm that can learn automatically through experience "E", concerning classes of a task "T", and performance evaluation "P", if it's the performance at tasks in "T", as evaluated by "P", improves with experience "E," and building intelligent machines which involve learning from the dataset (Tu, 2019). Machine learning algorithms are often divided into supervised learning, unsupervised machine learning, semi-supervised learning, and reinforcement learning.

2.3.1. Supervised Learning

Supervised learning is a type of machine learning in which train the machine using well-trained data. The labeled training data helps you to predict outcomes for unforeseen data (Sah, 2020; Sathya & Abraham, 2013). The system can give class for any unseen data after enough training. By comparing the output of the learning algorithm with the correct result supervised learning finds errors to modify the model accordingly. Supervised learning is categorized into a classification for category output and regression for real value (Sah, 2020; Sathya & Abraham, 2013).

2.3.2. Unsupervised Learning

Unsupervised learning is the type of machine learning technique, in which no need to supervise the model. It deals with unlabeled data. Unsupervised learning is more unpredictable than other learning techniques. This technique allows complex processing tasks when compared to

supervised learning (Sah, 2020; Sathya & Abraham, 2013). Instead, it needs to allow the model to work independently to discover information from the given model. It also mainly deals with unlabeled data to do tasks based on user requirements. Clustering and association are two types of unsupervised learning (Sah, 2020; Sathya & Abraham, 2013).

2.3.3. Semi-supervised Learning

Semi-supervised learning is a mixture of supervised and unsupervised learning. Used when your training data have few labels when there is no enough resource to label the dataset. They use classification for identifying the data and clustering for grouping the data into different groups, the semi-supervised model can increase the size of the unlabeled data (Sah, 2020).

2.4. Machine Learning in Software Engineering

Machine learning appears in many disciplines as its intersection between computer science, engineering, and statics. Machine learning can be used to solve many problems, any field can use machine learning to interpret and work on data (Masutani et al., 2012). Software engineers can use machines that assist them in software development. Machine learning-based solution can be developed for software engineering problems.ML has been applied to predict, estimate, classify, generate software engineering processes (Z., 1995).

Table 2.1: Machine Learning Applications in Software Engineering Artifacts

Software Engineering Works	***Machine learning Model and Applications***
Software defect prediction	ML models are used to classify or predict software modules into defective or non-defective(Aljamaan, 2020; Zhiqiang Li et al., 2018; Zhou et al., 2019).
Software reliability prediction	ML models are used in estimating and predicting system failing in given time spam(Jaiswal & Malhotra, 2016).
Software quality prediction	ML models have been used for predicting internal quality attributes of software(Pattnaik & Pattanayak, 2016).
Software cost and effort estimation	ML models have been used to predict the resource and effort required to develop a software system such as budget and schedule(Wen et al., 2012).

Software reusability prediction	Identifying the probability of reusing existing software components(Maggo & Gupta, 2014).
Software Size Estimation	Estimating size of software component using LOC metrics(Regolin et al., 2003).

2.4.1. Machine Learning in Software Defect Prediction

Based on Software Bug Prediction, several researchers have studied different techniques to build and evaluate several software fault prediction model using classification, regression, unsupervised method and soon. Although, the method of fault prediction model that several researchers had used is a common to predict or detect fault of software modules. That is categorized into three: binary class classification of fault(Alsawalqah et al., 2017, 2020; Zhan Li & Reformat, 2007; Mendes-Moreira et al., 2012; Santosh S. Rathore & Kumar, 2019; Vandecruys et al., 2008), number of bug/fault density prediction(Santosh S. Rathore & Kumar, 2015, 2017, 2019; Santosh Singh Rathore & Kuamr, 2016; Santosh Singh Rathore & Kumar, 2017) and severity of bug prediction(Sandhu et al., 2011; Shatnawi & Li, 2008). From them, mostly used types of prediction scheme is binary class classification, where software modules having one or more faults are distinguished as faulted and modules having zero faults are distinguished as non-faulted.

In this type of bug prediction schema, most researchers have discovered and used supervised approaches such as ensemble methods(Alsawalqah et al., 2017; Huda et al., 2018; Jiang et al., 2008; Sun et al., n.d.), Decision Tree (DT)(Koprinska et al., 2007), Support Vector Machine (SVM)(Elish & Elish, 2008), Naïve Bayes (NB)(Menzies et al., 2007), Random Forest(RF), Neural Network (NN), Logistic Regression(RF) (Dhiauddin et al., 2012).

Different researchers were used several different machine learning techniques and built the model for software defect prediction to improve quality of software development that based on timely detection of fault in software components or modules (early detection of fault in SDLC) and to identify best appropriate classification algorithms as discussed below.

Support Vector Machine(SVM): It is a kernel based learning methods of supervised learning approach that used to solve classification and regressions problems. It uses hyperplane decision boundary to classify between instances classes of a data. The purpose of this methods is to search or find optimal decision planes that identify the classes in well. The study of software defect

prediction proposed by (Elish & Elish, 2008) were used SVM because of this approach obtained very good performance, suitable with high dimensions features and prevents outliers.

Naive Bayes(NB): It is a probability based learning methods of supervised machine learning technique that depend on theorem of Bayes to classify a problem of multi class and binary class domains. This algorithm treats all inputs attributes as independent and equally contribute for prediction. It calculates the probability of each classes of instances based on new instances and then select the class of highest posterior probability for prediction(Studies et al., 2014). Defect prediction model proposed by (Menzies et al., 2007) were used Naive Bayes built model to identify defective modules because of this approach works with small sized data, easy to perform/implement, rapidly train and predict.

Decision Tree(DT): It is one of supervised learning approach that used to solve classification and regressions problems by using tree traversal from root node to leave node after data information processed by decision rules. The aim of this approach is to build a model of decision tree that based on decision rules generated during training data to classify or predict the decision or the value of target variables. In tree of decision tree data structure each node represent a features of the data, branches represent a value and classification or identification of an instances starts from root node and go downward to leave node (Ali et al., 2012).

K-Nearest Neighbor(KNN): It is instance based classifier of supervised learning approach that used to solve regressions and classification problems. It categories/identifies instances exists in the dataset by calculating distances between each instances and unknown/new instances, and then assigning the class label of the nearest neighbor to every new instances based on number of k's majority vote. Mostly used distance formula in KNN are Euclidean and Manhattan rule of distance calculation (Studies et al., 2014).

Ensembles Learning Algorithms: It is a method of combining or joining decision of multiple machine learning classifier as base classifier to get better performance of prediction(Read et al., 2012; Studies et al., 2014). It creates single most strong and reliable model from combination of weak and strong classifier to enhance prediction performance and minimize variance and bias of the model. Ensembles are categorized into bagging, boosting and stacking.

1) Boosting Techniques

Boosting algorithms is a sequential based ensemble learning algorithms that change weak base classifier to strong classifier(learner) by iteratively fixing error of previous base learner in the

next base learner that works better and minimize a bias. It is a sequential homogeneous weak classifier ensemble. Some example of Boosting technique is: XGBoost, Gradient Boosting, AdaBoost and etc. and they are described on the following.

- ***AdaBoost:*** Adaptive Boost(AdaBoost) is a one of most familiar boosting approach that build an ensemble by doing several iterations every time with different observation (instance) weights and change slightly to achieve more correct result to the errors returned by learners(classifiers) from previous occurring iterations (Freund et al., 1996; Schapire, 2013). Updating the weights of training instances in every iteration pushes the learning classifier to put more attention(value) on instances that incorrectly classified in previous iteration and less attention(value) on instances that correctly classified in previous iteration. That means, weights of incorrectly classified instances are become increased, but weights of error free classified instances are become decreased. This shows in next iteration wrong classified instance measure more heavily. AdaBoost uses the forecasts of several weak classifiers and provide a final prediction via aggregated voting on approaches. Weak classifier algorithm is originally stated by Yoav Freund and Robert E. Schapire and it is a classifier that work appropriate than random guessing (Freund & Schapire, 1997).
- ***Gradient Boosting(GB):*** **GB** is a generalization of adaptive boosting(AdaBoost) ensemble algorithm that can be construct by utilizing a diversity of loss function. Adaboost assigns the weight to incorrectly classified observations(instances) as next perform model (next iteration) predicts correctly whereas Gradient boosting utilizes gradient to fit new base learner. Gradient boosting (GBM) work by boosting techniques that aggregating previous predict errors with every subsequent numerous weak learner to provide strong classifier and it is decision tree based algorithms. Specially, Regression trees are act as a base classifier for both classification and regression problems and provides more robust, interpretable manner and competitive (Friedman, 2014). Using Gradient boosting can improve the quality of predicting (fitting) the base learners(classifiers), although, it is not operating quickly during processing time and it needs more memory.
- ***XGBoost (XGB):*** XGBoost is denote to extreme gradient boosting that applies gradient boosting tree approaches to build ensemble technique (Chen & Guestrin, 2016). It is a gradient boosting tree ensemble that employed the second-order differentials(derivatives) of the loss function to get the optimal base learner more efficiently and accurately. Whereas

Gradient Boosting utilizes gradients to fit a new base classifier, XGB uses the second-order derivatives gradients. This machine learning approach is decision trees based by using gradient boosting framework algorithmic improvement (i.e., Weighted Quantile Sketch, Sparsity Awareness, Regularization) and hardware and software optimization methods (i.e., Tree Pruning, Parallelization). These enhancements produce higher results by using few computing resources within short time. XGB build reliable and accurate model by creating effective classifier from a group of weak decision tree algorithms. then create decision tree ensemble that utilizes the output of the weak classifier in the last prediction.

2) Bagging Techniques

Bagging algorithms is sometimes known as Bootstrap Aggregation algorithms. It is a parallel based ensemble learning algorithms which all base learners are independent of each other to minimize the variance in the last forecast (prediction). The Bagging algorithms is construct random subsets of instances(sub-samples) from original dataset to execute/train multiple base model in parallel on created each subsets and then aggregates the prediction of all learner(model) to provide strong ensemble learner that smaller over-fitted. Bagging algorithms is effective if it works with unstable machine learning classifier. It is homogeneous base learners. Some example of Bagging technique is: Random Forest, Extra Trees, Bagging Meta-estimator and etc. and they are described on the following.

- **Random Forests (RF):** The RF algorithm is a special case of Bagging techniques that holds a collection of decision tree structure algorithm. RF chooses random attributes in order to construct bootstrap learner using decision trees (Breiman, 2001). RF constructed from numerous decision trees by choosing a subset of data and attributes randomly. Randomly selects a subset of observation from chosen variables and given to decision tree learning. then RF algorithm chooses the prediction that has highest votes from all decision trees output in the forest to produce final prediction. Random forest combine output from several trees and this combination will provide high accuracy on prediction. Random forest algorithm uses randomization in two important places: First, every decision tree is executed/trained by taking random sub-sample with replacement from original dataset. Second, when executing individual decision trees algorithms (at every node) random subset of attributes is used to find best splits. Making randomization is minimizing the correlations between decision trees, which enhance performance of prediction. Additionally, working with randomness on

datasets and attributes are decrease the risk of model overfitting. RF uses decision tree algorithm as a base estimator and assigned by default in RF algorithms.

- **Extra Trees (ET):** Extra Trees denotes to extremely randomized trees that uses bagging technique and almost the same to Random Forest, but with additional randomness. ET algorithm is different from RF algorithm in two things: The first, every decision tree is constructed by using entire dataset, and the second at every node in the tree, ET algorithm chooses the splits randomly without finding the best splits. ET algorithm uses the top down method to construct unpruned trees classifier ensemble technique. This algorithm is used for supervised regression and classification problems and it accuracy and computational performance is good (Geurts et al., 2006).
- **Bagging Meta-estimator:** Bagging Meta-estimator is an ensemble machine learning algorithm that follows a bagging techniques procedure to enhance accuracy and stability of machine learning classifier by aggregating prediction output of several weak learners(Bbeiman, 1996). Bagging algorithms are more appropriate and effective with unstable machine learning techniques that means when small changes happen on training set the consequence of that changes is huge on predictions output. Example of unstable machine learning techniques are decision tree and artificial neural network. This algorithm uses user-specified classification algorithms on base estimator.

3) Stacking Technique

Stacking algorithms, sometimes recognized as stacked generalization technique. It is one of an advanced ensemble machine learning algorithms that combines several different machine learning algorithms through meta learning algorithm that is either a meta-regressor or meta-classifier. The base weak learner level machine learning algorithms are executed/trained on the whole training dataset, and after that the meta learner(model) is accept output prediction from base weak learner level algorithms as inputs. The base weak learner models are occurred at level-0 model and meta learner that aggregates and accept the output predictions of base weak learners as input are occurred at level-1 model to produce final prediction. level-0 algorithms are diverse. This meta learner is trained to combine the learner predictions in best manner to

provide a final prediction(Funda et al., 2017). Stacking technique performs it task parallel and it uses heterogeneous base weak learner algorithms.

Table 2.2: Advantages and disadvantages of mostly applied machine learning algorithms[1,2,3,4] [61]

Machine Learning Techniques	***Advantages***	***Disadvantages***
SVM	Computationally cheap, it can supply non-linear solutions, it performance not more affected by outliers and not easily affected by overfitting, it works good in a higher dimension, and it is appropriate for a case of binary classification	To obtain good performance, it needs kernel knowledge , Sensitive effects by tuning parameters, In large amount of dataset, and it needs huge amount of time to perform a task because it is slow in speed
NB	Performs good in small amount datasets, It has high speed in prediction, Scalable with huge amount datasets, Not affected by irrelevant features, Effective for Multi class prediction**,** and It obtains best performance on data of large number of features	Assumption of independence among features, Poor quality estimator, and the way of preparing input data is sensitive
DT	It can control/hold categorical features, It does not need data scaling and normalization, It can control missing values, It selects important features automatically, It is easy to visualize, and Humans can easily understand learned outputs	It can easily happen overfitting, It has sensitive effect to data, and It needs huge time to train
KNN	Easy to implement and understand**,** It can solve Multi-class problem**,** It does not have	It has low speed for large amount of dataset**,** It does not comfortable for imbalanced dataset, It does not

[1] https://towardsdatascience.com/pros-and-cons-of-various-classification-ml-algorithms-3b5bfb3c87d6
[2] https://towardsdatascience.com/a-practical-guide-to-stacking-using-scikit-learn-91e8d021863d
[3]https://cppsecrets.com/users/79279710010511610510711111610410512197108554848641031099710510846991111109/Python-AdaBoost-Advantages-and-Disadvantages-of-AdaBoost.php
[4] https://medium.com/analytics-vidhya/introduction-to-the-gradient-boosting-algorithm-c25c653f826b

	assumption about the data**,** and Constantly evolving model	appropriate with the data of large number of features, It is sensitive to noisy/outliers, Can't work with missing values, and It needs to decide number of neighbors "k" manually by user
AdaBoost	It is not more affected by overfitting**,** It works with several classifier and boost their performance, It categorizes images, text and binary classification problems, and It doesn't have parameters to adjust	It wants quality of data, and It is sensitive to noisy/outliers
GB	It gives high prediction accuracy, It can control missing values, It is very changeable/flexible, and Most of the time it does not require data preprocessing	It is not operating quickly during processing time, It needs more memory, and During improving to reduce all errors it can cause overfitting and outliers
XGBoost	It does not need more feature engineering, It produces feature importance that used as feature selection, It models performance is good, It is a fast to interpret, Outliers have small impact, It uses and control large amount of datasets in well, It has good running speed, and It is not more affected by overfitting	Difficult to tune parameters due to have several hyper-parameters, and It may happen overfitting because of improperly tuned parameters
RF	It is an algorithm that use bagging technique to improve accuracy by minimizing variance, It can de-correlate trees, It minimized error, It performance is good on Imbalanced datasets, It controls/holds large amount of datasets, It can control missing data/values very well, Outliers have small impact, It does not have overfitting problem, and It is useful to get feature importance that used as feature selection	Predictions of the trees require to be uncorrelated, It appears as Black Box due to trying different parameters, It constructs complex structure due to RF creates a lot of trees, Features require to have some predictive power, and It needs much time to train

ET	It is one of bagging technique algorithm that use further randomization to improve accuracy by minimizing variance, It computational performance is good, It is a faster in execution, It is useful to get feature importance that used as feature selection, It minimizes bias by constructing decision tree by using entire dataset, It is similar with RF except it uses additional randomness, and It is tune to problem by selecting appropriate parameter due to it uses randomization for selection of features and cut-point while splitting a tree	It constructs complex structure due to ET creates a lot of trees
Bagging Meta-estimator	It is an algorithm that uses bagging technique to improve accuracy by using combination of several weak model to provide strong classifier, It better than individual strong classifier/learner, It minimizes a variance that prevents overfitting, It produces stable classifier from unstable learners, and Users can defined different base machine learning classifier specifically based on it is own problem	It computationally expensive, and If it is improperly modelled, it may produce high bias and under-fitting
Stacking	It can produce improvement performance, and It minimizes a variance and build high robust model by joining the predictions of several learners.	Producing predictions by using it models will be slower and computationally expensive, and It models can hold significantly longer time to train and need high memory

2.5. Feature Selection Technique

In the area of software bug prediction software defect datasets have many features(attributes) that are irrelevant and redundant(Ni et al., 2017; Xu et al., 2016). Direct building of machine learning models by these datasets are a problem, because irrelevant and redundant attributes exists in the datasets may lead the models to more time consume during training, lacking sufficient performance of prediction, increase complexity of the model and increase model overfitting(Mehta, 2021; Ni et al., 2017; Xu et al., 2016). The method of avoiding unnecessary attributes and selecting important features are called feature selection. Feature selection is a method of identifying most benefit attributes and eradicating noisy features from huge set of attributes in the datasets. It is also a method of automatically selecting best attributes that have strong relation with the target variable for machine learning algorithms depending on type of problems that are attempting to solve. It aims to prepare and appropriate best attributes for machine learning algorithms as those selected attributes are lead to increase performance, increase interpretability of the model, reduce computational cost and decrease overfitting(Mehta, 2021).

Under supervised machine learning algorithms feature selection methods are divided into three groups: filter, wrapper and combination of both(embedded) methods.

Filter Methods: This technique is always occurring at preprocess stage and to evaluate appropriate features by using rank of score procedure that consequently extracts low point score attributes. This method is independent from machine learning classifier, fast in execution, simple in computational and scalable. It drops attributes depending on a relation they have with target attribute(Cunningham & Delany, n.d.; Jovi et al., n.d.; Wah et al., 2018). Some examples of filter feature selection methods are ANOVA, Pearson's Correlation, Mutual Information, Chi-Squared, Information Gain, Correlation Based Feature Selection, etc.

Figure 2.1: Filter Method Working Flow

Wrapper Methods: This technique is based on classification machine learning algorithms to select subset of necessary attributes. So, it is classification machine learning dependent feature selection methods which provide better outcome when compared with filter methods, however,

it is tend to high computational cost when number of attributes becomes very high. This method is firstly selected subset of attributes from complete attribute set and then train the model by using selected subset of attributes. Based on output of previous model the attributes to add or extract from selected subsets are decided and then train again using the model by updated attribute subset. It creates attributes subsets by using greedy technique and evaluates each attributes subsets (Cunningham & Delany, n.d.; Jovi et al., n.d.; Wah et al., 2018). Some popular examples of wrapper feature selection methods are recursive feature elimination, backward feature elimination, forward feature selection, etc.

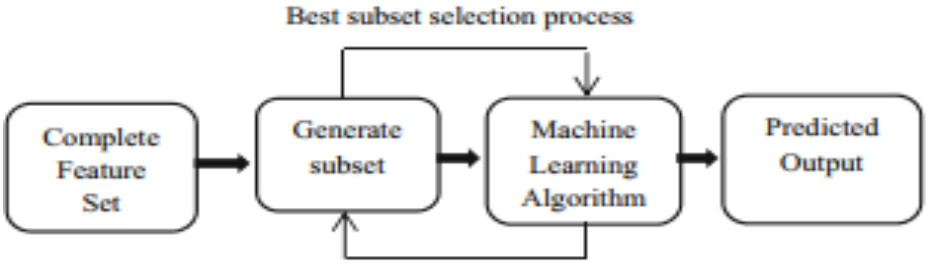

Figure 2.2: Wrapper Method Working Flow (Best subset selection process)

Embedded Methods: This technique is provided from combination or hybrid of wrapper and filter methods. This method combines most appropriate properties of both wrapper and filter methods to produce or provide best attribute subsets. Algorithms used for this method implementations have their own specific built-in feature selection approach (Cunningham & Delany, n.d.; Jovi et al., n.d.). Ridge and Lasso Regression are most popular among example of embedded feature selection methods.

.

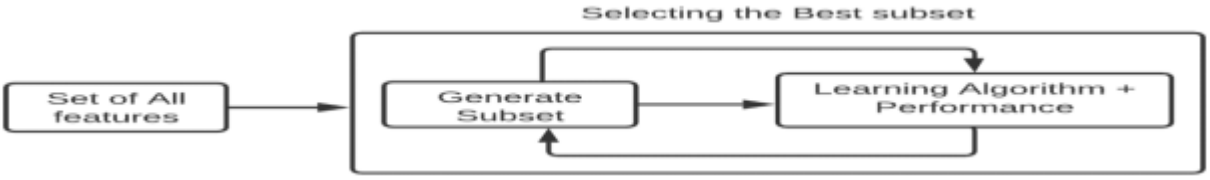

Figure 2.3: Embedded Method Working Flow

Table 2.3: Taxonomy of feature selection technique [59], [60], [62]

FS Technique		***Advantages***	***Disadvantages***	***Example***
Filter Methods	Univariate	It is a faster, It is scalable, It is independent of classifier, and It is computationally simple	It ignores feature dependencies, It ignores interaction with the ML classifier, and It leads to poor variable subsets	T-test, Information gain, Correlation, Gain ratio, Chi-square, Symmetrical uncertainty, Fisher score, and Relief and ReliefF
	Multivariate	It Models feature dependencies, It is independent of the classifier, and It has better computational complexity than wrapper methods	It is slower than univariate techniques, It is less scalable than univariate techniques, and It ignores interaction with the classifier	CFS, FCBF, MBF, and mRmR
Wrapper Methods	Deterministic	It is Simple, It Interacts with the classifier, It models feature dependencies, It has less computationally intensive than randomized methods, and It is a best with high number of attributes	It has risk of over fitting, It is more prone than randomized algorithms to getting stuck in a local optimum (greedy search), and It is a classifier dependent selection	SFS, SBE, RFE, Plus L Minus R, and Beam search
	Randomized	It is less prone to local optima, It interacts with the classifier, and It models feature dependencies	It computationally intensive, It is a classifier dependent selection, and It has a higher risk of over-fitting than deterministic algorithms	Randomized hill climbing, Genetic algorithms, Estimation of distribution algorithms, and Simulated annealing

Embedded Methods	It interacts with the classifier, It better computational complexity than wrapper methods, It models feature dependencies, and It can choose high quality attributes set	It is a classifier dependent selection, and It does not consider correlation between features that may exist	Decision trees (Random forest), RIGDE and LASSO regression, Weighted naïve Bayes, and Feature selection using the weight vector of SVM

2.6. Related Works

This section elaborates the reviews of research works dealing with all aspects of software defect prediction.

In (Alsawalqah et al., 2017), the proposed software bug/defect prediction model by hybridizing ensembles machine learning algorithms with SMOTE class balancing method to solve the problem of class imbalance distribution of poor classification. The proposed approach firstly applied ensembles algorithms with imbalanced datasets and then they obtained and selected optimal performing ensemble. After that imbalance datasets are balanced by SMOTE method as preprocessing stage and these balanced datasets join together with best ensemble that previously selected without data balanced and retrain a model to classify bugs in the software modules.

Algorithms used in proposed approach are ensemble classifiers such as Random Forest, Bagging and Adaboost with basic classifiers like Multilayer Perceptron (MLP), C4.5 Decision Trees and Naive Bayes (NB).

They used four NASA software projects which obtained from PROMISE Repository that are publically available.

The experiments organized and performed on three levels by using 10-fold cross validation; the first held on basic classifier by this J48 Decision Tree is selected as good performance for Bagging and Adaboost ensembles when compared with other base classifiers, the second held on ensemble classifiers with data of imbalance class by this Adaboost classifier is selected as a best ensemble result with J48 when compared with Bagging and RF, and the final is performed

on best ensemble classifier with SMOTE class balancing method. The result of experiments appeared that SMOTE with ensembles have enhanced performance of prediction.

The limitation of this approach is they does not apply method of feature selection that improve performance and the proposed approach performed only on 4 NASA datasets and it does not include others datasets.

(Zhou et al., 2019) proposed new deep forest approach to enhance performance of software fault prediction that namely DPDF. This approach uses new cascade strategy to identify more significant fault attributes by transforming random forest and completely random tree forest into structure of layer by layer form. Every layer of the cascade structure holds forest classifier such as random forest, completely random tree forest and XGBoost but in this approach they use only random forest as base learning. Deep forest is hybrid combination of deep learning and ensemble learning instead of deep neural networks that have a problem of determining suitable hyper-parameters which is difficult and needing very huge data.

They used 25 software projects that were obtained from NASA, PROMISE, AEEEM, Relink datasets and normalize the attribute by z-score standard to detect defective instances on the modules, classes, and methods of a software system.

The researchers perform comparison of the experimental result of DPDF with others algorithms such as Multi-Grained Deep Forest(gcForest), Random Forest(RF), Deep Belief Networks(DBN), Support Vector Machine(SVM), Naïve Bayes (NB) and Logistic Regression(LR) by using 2-fold cross validation. The result shows that defect prediction using cascade deep forest increased AUC performance measurement by 5% when compare with others algorithms.

The researchers have limitations like lack of feature selection methods that improve performance, they did not apply data balancing techniques for imbalanced datasets that solve biased model result, finding most optimizing parameter setting and fault prediction of cross projects that uses one project for training and other project for testing.

(Aljamaan, 2020) proposed software fault prediction model which analysis and applied tree based ensembles learning approaches to classify fault in software systems. In this investigation seven tree based ensemble algorithms are compared together by using only decision tree as base classifier. Those are two algorithms from Tree-based bagging and five algorithms from Tree-based boosting. Two Tree-based bagging algorithms which contributed in this analysis are

Random Forest(RF) and Extra Trees(ET), and five Tree-based boosting algorithms which contributed in this analysis are such as Adaptive Boost(AdaBoost), Categorical Boosting(CatBoost), Extreme Gradient Boosting (XGBoost), Gradient Boosting(GB) and Histogram based Gradient Boosting(HGB).

They used 11 MDP NASA software datasets that are publically available. In addition, they used gain ratio feature selection method and SMOTE data balancing method in preprocessing stages. They were performed model validation by using 10-fold cross validation and experimental result shows that prediction performance of two Tree-based bagging algorithms (RF and ET) are outperform others Tree-based boosting ensemble algorithms. However, individual decision tree performance was lower than every Tree-based ensemble. In addition, AdaBoost Tree-based ensemble performance was worst when compared to others Tree-based ensembles.

The researchers have the following limitations: they applied only single feature selection called gain ratio and they did not analysis the effect of others different feature selection, they used only NASA datasets and others datasets did not contributed, they performed only ensembles based on decision tree as base classifier and they did not compare others base classifiers with it , they used only homogenous ensembles and they didn't examine heterogeneous ensembles effect ,and they used only default hyper-parameters.

(Esteves et al., 2020) proposed bug prediction approach that based on few features which it consequence has correct model predictive and model elaboration/explainability that define the reason of software modules or instances to be faulty state. Their approach analysis concerning elaboration of model decision that support software developers to know the reason behind the fault model by searching and discovering features of the software that used for fault understanding. And this approach classifies the modules of software into fault and non-fault.

They used eight software projects which developed by java programming language that collected by Jureczko and publicly available in PROMISE repository. In their approach they applied random searching model space of XGBoost ensemble algorithm with SHAP (SHapley Addictive exPlanation) that used to provide model elaboration/explanation with minimum set of features that identified based on contribution of those features have on prediction. And also they performed large cover comparison of defect prediction approaches by using machine learning algorithms such as Logistic Regression, Decision Tree, K- Nearest Neighbor, Support

Vector Machine, Neural Network, Naïve Bayes, Random Forest and XGBoost with the full set of software features.

Model evaluation was performed by using 10-fold cross validation and appropriate hyper-parameters, and experimental result shows that XGBoost using random searching and minimum set of features (not more than 3 features) with SHAP model elaboration outperform others algorithms applied in comparison.

Limitation of the researchers are they used only few projects dataset that developed by single programming language and the result is not include projects developed by others programming languages , they performed within project dataset and they did not perform cross project defects datasets, they did not applied data balancing mechanism for imbalanced datasets that solve biased model result, they performed model elaboration only by SHAP and did not include other tools like LIME and Break Down, and their results not generalized in all machine learning classification algorithms exist and also used few ensemble algorithms.

(Mehta, 2021) proposed software defect prediction model that uses combination of feature selection, feature extraction, class balancing mechanism and classification techniques that based on ensembles algorithms to classify software modules into fault and non-fault.

The model aims to solve the problem of high dimensionality and data imbalance. In this method they used feature reduction and data balancing mechanism with z-score data normalization and missing value processing before training the model.

In feature reduction techniques they used Partial Least Square (PLS) Regression as feature extraction mechanism and combined with feature selection techniques. In feature selection mechanisms they compared several techniques such as Correlation-based feature selection, Recursive Feature Elimination (RFE), Boruta, ElasticNet, Ridge and Lasso. From that Recursive Feature Elimination outperform other feature selection techniques in most datasets and this technique combined with PLS Regression.

And combination of both used as feature reduction and also these reduced features combined with data balancing mechanism called SMOTE (Synthetic Minority Oversampling). After that the combination of feature selection, feature extraction and balancing SMOTE of each datasets inputs to some of machine learning algorithms such as Decision Trees, Logistic Regression, Support Vector Machines, K-nearest neighbor, and Ensemble Learning like Random Forest,

Extra Tree, Adaptive Boost, Stacking, XGBoosting, and Bagging to train and classify software modules as fault and non-fault. They also used only DT as base classifier for ensemble learning. They used 4 NASA software projects datasets that are publically available in PROMISE Repository to classify fault and non-fault instances.

The experimental result shows that XGBoost and Stacking ensemble algorithms are the highest results than others and these algorithms accuracy are more than 0.9 in all datasets.

Limitation of the researchers are they performed the model only on 4 NASA datasets and did not include other datasets, they didn't perform data integration of all used NASA datasets and others and also the performance of data integration is not analyzed, they did not perform hyper-parameter mechanism that may increase performance, they did not use others base classifier, they did not build the model using cross project datasets and they did not include effect of all feature selection technique recommended in software defect prediction area.

(Santosh S Rathore & Kumar, 2021) proposed software defect prediction approaches by using empirical study of new improved ensembles algorithms. Researchers used new improved ensembles algorithms that not explored previously in the area of software defect predictions such as Grading, MultiBoostAB, RealAdaBoost, Decorate, Ensemble Selection, Rotation Forest, and Dagging by empirically evaluating their performance. And they used base classifiers such as J48 (decision tree), Naive Bayes and logistic regression for these ensembles.

They used 28 software projects of PROMISE datasets that publically available in PROMISE repository to categorize module of the software as a fault and non-fault.

They were performed model validation by using 10-fold cross validation and analysis of experimental result shows that prediction performance of Rotation Forest ensemble with J48 base classifier are outperform other new improved ensemble algorithms in most case. In some cases, Decorate, Dagging and MultiBoostAB ensembles given effective performance. And Naïve Bayes as base classifier was resulted lower performance while J48 (decision tree) as base classifier was improved performance.

Limitation of the researchers are they did not apply feature selection methods, they did not apply data balancing techniques for imbalanced datasets that solve biased model result, they did not apply data normalization methods, and they used only projects of PROMISE dataset that developed by single programming language called java which have only 21 software metrics and the result is not include projects developed by others programming languages.

(A. Balogun et al., 2020) proposed software fault prediction approach that join together homogeneous ensembles algorithms with SMOTE class balancing technique. The researchers used boosting and bagging homogeneous ensembles techniques with base classifier such as Bayesian network(BN) and Decision tree(DT).

They used 5 NASA software projects datasets that are publically available in NASA Repository to classify fault and non-fault instances.

They were performed model validation by using 10-fold cross validation and analysis of experimental result shows that prediction performance of proposed approaches that join together Boosting with DT as base classifier and with SMOTE are better than others.

Limitation of the researchers are they did not apply feature selection methods, they did not apply data normalization methods, they performed the model only on 5 NASA datasets and did not include other datasets and they did not perform hyper-parameter mechanism that may increase performance.

Table 2.4: Summary of Related Works

Authors	***Base Classifier***	***Methods(Ensemble Learning)***	***Datasets***	***Feature Selection***	***Data Balancing***	***Gaps***
(Alsawalqah et al., 2017)	MLP, C4.5 DT and NB	-RF, Bagging and Adaboost. -Adaboost with C4.5 DT is outperform others	4 NASA	No	SMOTE	-No feature selection, others datasets are not included and no hyper-parameter mechanism
(Zhou et al., 2019)	completely RF and XGBoost	- Deep Forest(DPDF) is good performance with other comparison	9 NASA 8 PROMISE 5 AEEEM 3 Relink 25 Totally	No	No	-No feature selection, data is not balanced, not finding most optimizing parameter setting and not performed fault prediction of cross projects.
(Aljamaan, 2020)	DT	-RF, ET, AdaBoost, CatBoost,	11 NASA	Gain ratio	SMOTE	-Didn’t analysis the effect of others feature selection techniques, others datasets are

		XGBoost, GB and HGB -RF and ET outperform others				not included, didn't use others base classifier, didn't examine heterogeneous ensembles effect and no hyper-parameter mechanism
(Esteves et al., 2020)	DT	-XGBoost is good performance with other comparison	8 PROMISE	SHAP used as FS and select only maximum 3 features	No	- Data is not balanced, others effective ensemble algorithms not included, others programming languages projects are not included, not performed fault prediction of cross projects, did not include other tools like LIME and Break Down as feature selection
(Mehta, 2021)	DT	-RF, ET, AdaBoost, Stacking, XGBoost, and Bagging - XGBoost and Stacking outperform others	4 NASA	PLS as feature extraction with CFS, RFE, Boruta, ElasticNet, Ridge and Lasso FS	SMOTE	-Others datasets are not included, didn't perform data integration of all used NASA datasets and others and also the performance of data integrated is not analyzed, didn't perform hyper-parameter mechanism, didn't build the model using cross project datasets, didn't use others base classifier and did not include effect of all FS technique recommended in software defect prediction area
(Santosh S Rathore	J48 (DT), NB and LR	-Grading, MultiBoostAB, RealAdaBoost,		No	No	-No feature selection, data is not balanced, did not apply data normalization method and

& Kumar, 2021)		Decorate, Ensemble Selection, Rotation Forest, and Dagging - Rotation Forest with J48 outperform than others	28 software projects of PROMISE			The result is not included projects developed by others programming languages .
(A. Balogun et al., 2020)	Bayesian network(BN) and DT	-Boosting and Bagging - Boosting with DT is outperform others	5 NASA	No	SMOTE	-No feature selection, didn't apply data normalization methods, didn't include heterogeneous ensemble learning, didn't include other datasets and didn't perform hyper-parameter.

Existing empirical studies shows that several researchers implemented different approaches on different datasets with different problems. However, still there is a gap on AEEEM projects dataset which has a problem of high feature dimensionality in case of some features of a datasets are less significant in fault decision that needs a method of identifying most relevant set of features and imbalance class distribution problem in case of the number of defect class instances of a datasets are very few in ratio when compare with non-defect class ratio that needs further solution in software fault prediction model. And the target is to improve the performance of these datasets using machine learning algorithms or base classifiers. While, the performance of ensembles machine learning algorithms become more efficient in software defect prediction rather than individual base classifier. Therefore, this study proposes software defect prediction approach by employing combination of class balancing methods, feature selection mechanisms and ensemble learning algorithms on AEEEM software projects dataset.

CHAPTER THREE

3. RESEARCH METHODOLOGY

In this chapter, we write about methodology for the development of software defect prediction that uses different types of ensembles machine learning algorithms. In this section, we elaborate dataset sources, description of datasets uses, mechanism of data preprocessing used to build software defect prediction, procedure of model selection and method of model evaluation metrics to fulfill purpose of the research.

3.1. Data Source and Dataset Description

We use datasets of AEEEM [5,6] projects that gathered from different version of software system like eclipse and apache which are publicly available online and was collected by (Ambros & Lanza, 2010). These datasets were used for software defect prediction by different approach(Ambros & Lanza, 2010; A. O. Balogun et al., 2020; Wang, 2016; Zhou et al., 2019).These project datasets were developed by java object oriented programming and have various software modules and software metrics that based on class level projects and evolution version of a system.

In AEEEM datasets, five software projects exist in which each software projects dataset has totally 62 attributes. They include one dependent attribute and 61 independent software metrics attributes which prepared from combination of Source code metrics of CK (Chidamber and Kemerer) and OO (Object Oriented) metrics, previous defects, Entropy of change metrics, Churn of Source code metrics and Entropy of Source code metrics are used and represented (Ambros & Lanza, 2010) and refer it for further information. Those metrics are described below:

1) **Source Code Metrics:** It is combination of 6 CK (Chidamber and Kemerer) and 11 OO (Object Oriented) metrics, and then totally 17 metrics are represented as CK_OO metrics. Those are **CK_OO metrics** of (weight method count, depth of inheritance tree, response of class, number of children, coupling between objects, lack of cohesion, number of line code, Number of other classes that reference the class, Number of other classes referenced by the class, Number of attributes, Number of public attributes, Number of private attributes,

[5] http://bug.inf.usi.ch
[6] https://zenodo.org/record/3362613

Number of attributes inherited, Number of methods, Number of public methods, Number of private methods and Number of methods inherited).

2) **Previous Defects Metrics:** It is a metrics of past defect that used to predict future defect. Those are classified as All bugs, non-trivial bugs (severity>trivial), major bugs (severity>major), critical bugs (critical or blocker severity) and high priority bugs (priority and default), and totally they are 5-metrics.
3) **Entropy of Change Code Metrics:** It is a complexity of change code metrics of time interval file. Those are History of Complexity Metric, Weight History of Complexity Metric**,** Exponentially Decayed History of Complexity Metric, Linearly Decayed History of Complexity Metric and Logarithmically Decayed History of Complexity Metric, and totally they are 5-metrics.
4) **Churn of Source Code Metrics:** It is a metrics that calculated from source code metrics by using code churn deltas instead of line of code churn. And represent calculation of churn of each source code metrics as **WCHU** (Weighted Churn of source code metrics) with all source code metrics and totally 17-metrics. Those are **WCHU metrics** of (weight method count, depth of inheritance tree, response of class, number of children, coupling between objects, lack of cohesion, number of line code, Number of other classes that reference the class, Number of other classes referenced by the class, Number of attributes, Number of public attributes, Number of private attributes, Number of attributes inherited, Number of methods, Number of public methods, Number of private methods and Number of methods inherited).
5) **Entropy of Source Code Metrics:** It is a metrics that calculate entropy from source code metrics instead of entropy of change code. And represent calculation of Entropy of each source code metrics as **LDHH** (Linearly Decayed Entropy of source code metrics) with all source code metrics and totally 17-metrics. Those are **LDHH metrics** of (weight method count, depth of inheritance tree, response of class, number of children, coupling between objects, lack of cohesion, number of line code, Number of other classes that reference the class, Number of other classes referenced by the class, Number of attributes, Number of public attributes, Number of private attributes, Number of attributes inherited, Number of methods, Number of public methods, Number of private methods and Number of methods inherited).

Above all mentioned software metrics are independent attributes. However, the datasets also have target attributes that used to classify software module or class module into defect or non-defect. The Table 3.1 represents detail description of AEEEM datasets of five software project with number of attributes, number of modules(instances), programming language of the projects developed, number of defective and percentage of defective. Additionally, total integration is represented on the table.

Each AEEEM software projects were developed and used for different purpose as the following description:

- **EQ(Eclipse Equinox)[7]:** It is an eclipse implementation of OSGi framework specification software that implement different option of OSGi services and others infrastructure for executing OSGi-based systems. It is an implementation used for all tasks related to OSGi specification and it is a key framework specification services and extensions needed for executing eclipse and OSGi services.
- **JDT Core[8]:** It is a software used for Java infrastructures of Java IDE application development. It is used for eclipse builder Java compiler and used for code formatter. It is used for support of code select and code assist, and navigation of API java application.
- **LC(Apache Lucene)[9]:** It is a Java library that developed for full-featured searching engine and high-performance purposes. It provided a technology used for full-text searching, spell correction, structured searching and nearest-neighbor searching across many dimensions and even if for others application which required these tasks.
- **ML(Mylyn)[10]:** It is one of eclipse framework that used to manage the application lifecycle and task for eclipse. It supplies such as task management tool, revolutionary task-focused interface and ecosystem of application lifecycle management (that known by ALM) integration.
- **PDE UI[11] :** It is an eclipse user interface of Java IDE application development framework that provides set of tools to develop, create, test, fragments, update sites, features, debug and deploy Eclipse plug-ins. It also includes OSGi tooling that makes PDE for an ideal

[7] https://projects.eclipse.org/projects/eclipse.equinox
[8] https://www.eclipse.org/jdt/core/index.php
[9] https://lucene.apache.org/
[10] https://www.eclipse.org/mylyn/
[11] https://www.eclipse.org/pde/pde-ui/

environment purpose of component programming. It also provides the following things for Eclipse SDK such as New Project Creation Wizards, Import Wizards, Export Wizards, Form-Based Manifest Editors, Launchers, Conversion Tools, User Assistance Tools, Integration with JDT and soon.

Table 3.1: Description of AEEEM datasets of software projects

Dataset of Software Projects	***Programming Language***	***Number of Features***	***Number of Instances(Software Modules)***	***Defective***	***Percentage of Defective***
EQ	**Java**	**61**	**324**	**129**	**39.8**
JDT	**Java**	**61**	**997**	**206**	**20.7**
LC	**Java**	**61**	**691**	**64**	**9.3**
ML	**Java**	**61**	**1862**	**245**	**13.2**
PDE	**Java**	**61**	**1497**	**209**	**13.96**
Total(Integrated)	**Java**	**61**	**5371**	**853**	**15.9**

3.2. Software Fault Prediction Modeling

3.2.1. Data Preprocessing

Data preprocessing is important task that provide a data for machine learning algorithms in appropriate manner. Because, machine learning algorithms wants quality data to train, and to produce good performance, and it is one of the most important steps to build the dataset with suitable format as it is appropriate for the machine learning algorithm. Therefore, a dataset should be providing in appropriate manner before inputting into machine learning for training purpose. It is a method which perform cleaning a data noisy/error, control missing values, normalize different value of the attributes of a data in the same order of magnitudes, identify important features for target variables decision and balancing imbalanced appearance of a class as suitable and increase performance of machine learning algorithms.

Data Cleaning: This work is happening when the dataset needed to maintain from error, noisy, incomplete, missing, garbage, corrupted and duplicate. Missing value is one of the problems included under here and raised on a dataset, when specified instances value absent from the datasets due to the following reasons such as corrupted data, errors from data collector, having its own meaning, having noisy data etc. The method of finding and controlling missing values

of the datasets are either by ignoring missed values, delete missed value rows or fill the missed values by data imputation or other ways. For this research, we analysis and check our dataset for machine learning training as whether missing values exist or not in the dataset. After checked, AEEEM datasets we used for this study doesn't have missed values.

Data Normalization: It is very necessary part for model development of machine learning algorithms which transforms the values of variables that appeared in different magnitudes/scales into the same order magnitudes as all features have equal influence on the building model. For this study, we used this method to controlling outliers of the values(Mehta, 2021; Wang, 2016). Because, the dataset we used in this research have the problem of different scales on the value of features and outliers value on some features. Therefore, we have used Z-Score data normalization technique for this study based on preliminary works benefit (Zhou et al., 2019) to make the values of variables on the equal magnitudes/scales as all variables contribute equal influence for machine learning models. Z-Score data normalization is used to transforms the dataset feature result into the value of mean zero and standard deviation one

3.2.2. Methods of Feature Selection

In machine learning model development, the performance of the model can be degrading by very large number of input attributes/features. Not all original attributes/features are important for model development. Therefore, the method of avoiding unnecessary attributes/features and selecting important features that called feature selection techniques are important. Feature selection is a method of identifying most benefit attributes/features and eradicating noisy attributes/features from huge set of attributes/features in the datasets. It aims to prepare and appropriate best attributes/features for machine learning algorithms as those selected attributes/features are lead to increase performance, increase interpretability of the model, reduce computational cost and decrease over fitting. This method is divided into three groups: filter, wrapper and combination of both (embedded) methods.

Filter Methods: are one of the methods of feature selection technique that are independent from machine learning algorithms to select important attributes from original features by using rank score procedure or by using correlation with the dependent variables.

Wrapper Methods: are one of the methods of feature selection technique that are dependent on machine learning algorithms to select important subset attributes from original features. This method is firstly selected subset of attributes from complete attribute set and then train the model

by using selected subset of attributes. Based on output of previous model the attributes to add or extract from selected subsets are decided and then train again using the model by updated attribute subset.

Embedded Methods: are also amongst methods of feature selection technique that provided best attributes subsets from hybrid of wrapper and filter methods.

In this study, we used three kind of feature selection methods such as Correlation Based Feature Selection (CFS), Correlation-Filter(CO) and Sequential Forward Selection(SFS) based on their effectiveness and using in preliminary works benefit (A. O. Balogun et al., 2020). Correlation based Feature Selection(CFS) is included in filter methods which assess a whole attribute subset based on correlation between independent variables and target variables. Correlation Filter(CO) is also included in filter method that determines the merit of an attribute by calculating its Pearson correlation coefficient to the dependent variable and it is statistical based method. Sequential Forward Selection(SFS) is included in wrapper method that depend on classification algorithms and it begins to operate from an empty set and consequentially or gradually put attributes chosen by specified evaluation criteria.

3.2.3. Data Balancing Technique

Data balancing technique is used to solve class imbalance problem of a datasets by equalizing number of classes exists in a dataset that appeared in different quantities and biased the datasets which leads to influence the prediction results of machine learning algorithms. This technique is mainly including two methods i.e., under-sampling and oversampling:

Under-sampling technique equalizes the class distribution of a datasets by randomly deleting the dataset samples from majority class.

Oversampling technique equalizes the class distribution of a datasets by increasing number of minority class instances by randomly duplicating the minority class instances. Based on previous studies the most prominent example of oversample technique that mostly applied in software defect prediction area and produce excellent performance is SMOTE balancing technique (Aljamaan, 2020; Alsawalqah et al., 2017; A. Balogun et al., 2020). In this study, we used AEEEM software defect datasets which has a class imbalance problem on the class distribution of the datasets in which the ratio number of defect prone class are fewer than non-defect class modules. For this reason, we applied oversampling SMOTE balancing technique based on preliminary works benefits (Aljamaan, 2020; Alsawalqah et al., 2017; A. Balogun et

al., 2020) to solve class imbalance problem of software defect datasets by producing samples for defect prone modules (minority class) of class distribution of the datasets to balance defect prone modules with non-defect prone modules.

3.2.4. Machine Learning Models

In this study, we planned to predict software defect prediction using ensembles methods of machine learning algorithms using different individual base classifier to enhance quality of software and reduce maintenance cost, based on AEEEM software defect datasets that publically available software systems projects datasets. Therefore, we select different types ensembles machine learning algorithms such as Stacking, XGBoost, Bagging, Gradient Boosting, AdaBoost, Extra Trees and Random Forest for this investigation based on previous studies achievement result, good performance, minimize overfitting and better output produce than single classifier (Aljamaan, 2020; Mehta, 2021).

3.3. Evaluation of Predictive Model

It is an effective, very crucial step, and an integral part of the machine learning model development process to estimate the performance of predictive modeling. It is also used to measure how well the chosen model will work on unseen data. It is intended to evaluate the generalization accuracy of the model on unseen data. Cross-validation is the technique used for evaluating model performance on a test dataset that is not seen by the model, which helps to evade overfitting in predictive models especially in the case when the amount of data may be very limited.

3.3.1. Cross-Validation

K-Fold Cross-Validation: The dataset is split into k number of folds where the k-1 fold is used for training and 1 fold for testing. K value represents the number of folds that a given data sample is to be split into. It works as follows:

1. Randomly split the data into k-folds for training
2. Model is trained using K -1 folds and validate the model using the remaining Kth fold.
3. Step 2 is repeated until every K-fold serves as the test set.
4. Averages of the recorded scores will be the performance metric for the model.

For this study, we used K-fold cross validations that a number of k is 10 to evaluate software defect predictions.

3.3.2. Performance Evaluation Metrics of Proposed Model

To evaluate proposed prediction model developed, there are several measurements such accuracy, recall, precision, F-measure, AUC (Area Under ROC curve). In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error metrics, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is confusing two classes which is commonly involving mislabeling one as another. That means the developed algorithm has to identify instances as the following demonstration:

- **False Negative (FN)** represents the number of fault software instances that are wrongly classified as clean.
- **False Positive (FP)** represents the number of pure software instances that are mistakenly classified as faulty.
- **True Negative (TN)** represents the number of pure software instances that are rightly classified as clean.
- **True Positive (TP)** represents the number of fault software instances that are rightly classified as faulty.

Based on the above mentioned measurements, it is possible to represent them in the form of confusion metrics to measure prediction result of software fault prediction model.

Confusion Matrix is a method that summarizes the classification performance of the models concerning some test datasets, and the outcome can be binary classes. It is used to classify the data in a matrix format which is indexed in one dimension by the true class of an object, and the class that classifier assigns in the others dimension. The confusion matrix is used with binary class classifications in this study in which software defect prediction is classified as Fault and Non-fault.

Table 3.2: Confusion Matrix for Software Defect Prediction

		Predictive	
		Non-fault	**Fault**
Actual	**Non- fault**	**TN**	**FP**
	Fault	**FN**	**TP**

Based on this confusion metrics, the following criteria are used for evaluation of software defect prediction model:

Recall is the total number of true positive divided by the number of all relevant samples, or it is the ratio of true positive to a true positive plus false negative.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Equation 3.1. Recall

Precision is the ratio of the entire number of true positive results to the number of positive results predicted by the classifier model, and it is how close a measured value is to each other.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Equation 3.2. Precision

Accuracy (ACC) is the most common and popular performance evaluation metric for classification tasks and is defined as the percentage of correct predictions for the test dataset. It can be calculated by dividing the number of correctly classified by the number of total predictions.

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

Equation 3.3. Accuracy

F-Measure is Sometimes known as f1-score. It is the harmonic mean of the precision and recall, whereas the F-measure reaches its best values at 1 i.e., perfect precision and recall, and worst at zero.

$$\mathrm{F-Measure} = 2 \times \left(\frac{\mathrm{Recall} \times \mathrm{Precision}}{\mathrm{Recall} + \mathrm{Precision}}\right)$$

Equation 3.4. F-Measure

Area Under Curve (AUC) is the percentage of the area that is underneath the Receiver Operator Characteristic (ROC) curve. The AUC metric calculates the area under the curve, where a large area indicates high performance. AUC can have a value between 0 and 1, where 1 indicates the best performance and zero is considered the worst performance. The ROC curve is a plot of Sensitivity (True Positive Rate) vs (1 – Specificity).

CHAPTER FOUR

4. PROPOSED SOLUTION FOR SOFTWARE DEFECT PREDICTION

In this chapter, the proposed software defect prediction using ensemble machine learning technique to enhance quality of software and reduce maintenance cost is discussed in detail whilst answering the defined research questions. The system architecture of the proposed solution is demonstrated hereunder to show clearly the details of each and every component that found and happen within the proposed model.

4.1. Proposed Architecture of Software Defect Prediction

The proposed architecture of software defect prediction is intended to classify software modules into defect and non-defect by using ensemble machine learning algorithms based on software metrics as inputs. These inputs are datasets measured and gathered from software systems during any phase of software development life cycle. As shown in the **figure 4.1** below, the proposed architecture holds different methods and techniques that are used for software defect model such as data cleaning methods and data normalization methods in the first preprocessing stage. The goal is to provide quality datasets for the next process. After the first preprocess is completed, the second preprocess aim is to prepare a dataset as appropriate for different ensemble learning algorithms. In the second preprocessing stages, high feature dimensionality and data class imbalance which are the two main problems of the AEEEM software defect datasets which are used for this research are solved.

To solve high feature dimensionality problem, different feature selection mechanisms are applied based on features characteristics. The aim is as it is obvious to choose or select best attributes by identifying attributes that strongly decide the fault (target attribute) or by finding those attributes having strong relationship of each independent variables with dependent variables. Performing feature selections have several benefits for the model. These include increasing speed of the train, minimizing model overfitting, producing effective model on predictive power and improve accuracy. Therefore, for this paper, three feature selection techniques are used namely, Correlation Based Feature Selection (CFS), Sequential Forward Selection (SFS) and Correlation-Filter (CO) based on their effectiveness.

Class balancing technique is also applied to solve the problem of class imbalance of AEEEM defect datasets of software projects used for this research. The aim of class balancing approach

is to equalize number of classes that exist in the datasets specially to increase the performance of minority class by minimizing model bias. For this study, Synthetic Minority Oversampling Technique (SMOTE) is applied as class balancing mechanisms. These two methods, feature selection and class balancing, produce and give appropriate dataset to different ensemble learning algorithms for model training. Regarding the ensemble machine learning algorithms, Adaptive Boosting, Gradient Boosting, Extra Gradient Boosting, Random Forest, Extra Tree, Bagging and Stacking techniques are applied in this research. When appropriate data is trained by different types of ensemble algorithms, additionally it is good to make different types of individual classifiers work with ensemble learning as the base classifier based on characteristics of ensemble learning techniques.

Those different types of base classifiers machine learning algorithms applied in this research are such as Support Vector Machine, Naive Bayes, Decision Tree and K-Nearest Neighbor. And all ensembles machine learning algorithms and base classifiers algorithms mentioned above are used to construct predictive model that categorize or identify software modules into defect and non-defect. Predictive performance/efficiency of all models output is tested by using 10-fold cross validation and the best performance model is chosen/selected by using different evaluation metrics results. Lastly, software modules classification prototype model that is used to identify tested dataset of software modules into defect and non-defect by inputting software metrics is created from selected best performance/efficient model.

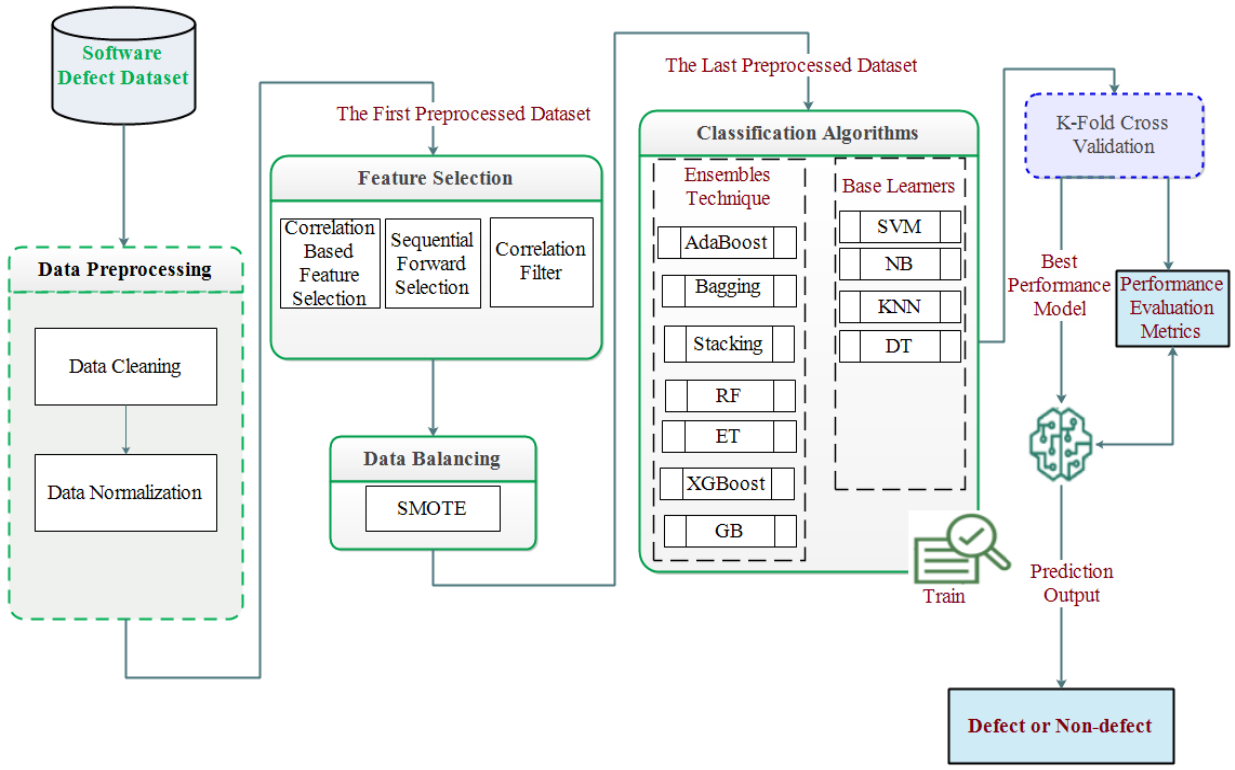

Figure 4.1: General Proposed Architecture of Software Defect Prediction

4.2. Data Preprocessing

Data preprocessing is important task that provide a data for machine learning algorithms in appropriate manner. It is a method which perform cleaning a data noisy/error, control missing values, normalize different value of the attributes of a data in the same order of magnitudes, identify important features for target variables decision and balancing imbalanced appearance of a class as suitable and increase performance of machine learning algorithms. Therefore, the general data preprocessing (before data given for training process) workflow have shown in the following diagram.

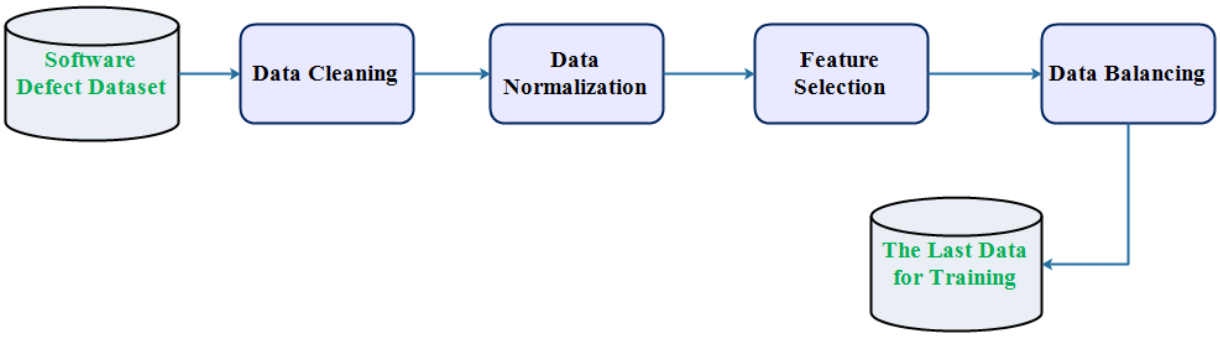

Figure 4.2: General data preprocessing workflow diagram

4.2.1. Data Cleaning

Data cleaning is one of the task of data preprocessing which is used to maintain dataset from error, noisy, incomplete, missing, garbage, corrupted and duplicate. Missing values are one of the problems maintained in data cleaning task which are specified instances value absent from the datasets because of different reasons such as corrupted data, errors from data collector, meaning, noisy data and etc. The method of finding and controlling missing values of the datasets are either by ignoring missed values, delete missed value rows or fill the missed values by data imputation or by any other ways. In this paper, datasets for missing values are checked and our datasets AEEEM we used for this study doesn't have missed values.

4.2.2. Data Normalization Technique

Data Normalization is an extremely important part for model development of machine learning algorithms which transforms the values of variables that appeared in different magnitudes/scales into the same order magnitudes as all features have equal influence on the building model. This method is also used to controlling outliers of the values. The dataset used in this research have the problem of different scales on the value of features and outliers value on some features. Therefore, we use Z- Score data normalization technique for this study to make the values of variables on the equal magnitudes/scales.

Z-Score data normalization **is o**ften called standardization. It is one of data normalization technique that transforms the dataset feature result into the value of mean zero and standard deviation one. All standardized value is calculated by subtracting the average of the corresponding attribute/feature from each values exists on the feature and after that dividing the result to that feature standard deviation value.

$$x_{standard} = \frac{x - \mu}{\sigma}$$

Equation 4.1. Standardization

Where, x represents each instances value of the specified feature, μ represents a mean value of all instances of the feature and σ represents a standard deviation value of all instances of the feature. Of course, the Z-Score result characteristics the ranges from -3.00 to 3.00 when the input data is normally distributed.

4.2.3. Feature Selection Technique

After the first preprocessing such as controlling missing values and data normalization have performed their output results provide the datasets for the next process/task such as feature selection and data balancing technique to produce suitable dataset for machine learning training as the second preprocessing. Feature selection technique is a method included in second preprocessing to analysis and select most important attributes that have strong relationship with target attributes to improve performance of a models, to avoid irrelevant and redundant attributes and to eradicating noisy features from the dataset. The purposes of feature selection techniques are to provide appropriate best attributes for machine learning algorithms as those selected attributes lead to increased performance, produce effective model on predictive power, reduce computational cost, decrease overfitting and increase interpretability of the models. For this reason, in this research we used three feature selection techniques such as Correlation Based Feature Selection (CFS), Sequential Forward Selection(SFS) and Correlation-Filter(CO) based on their effectiveness.

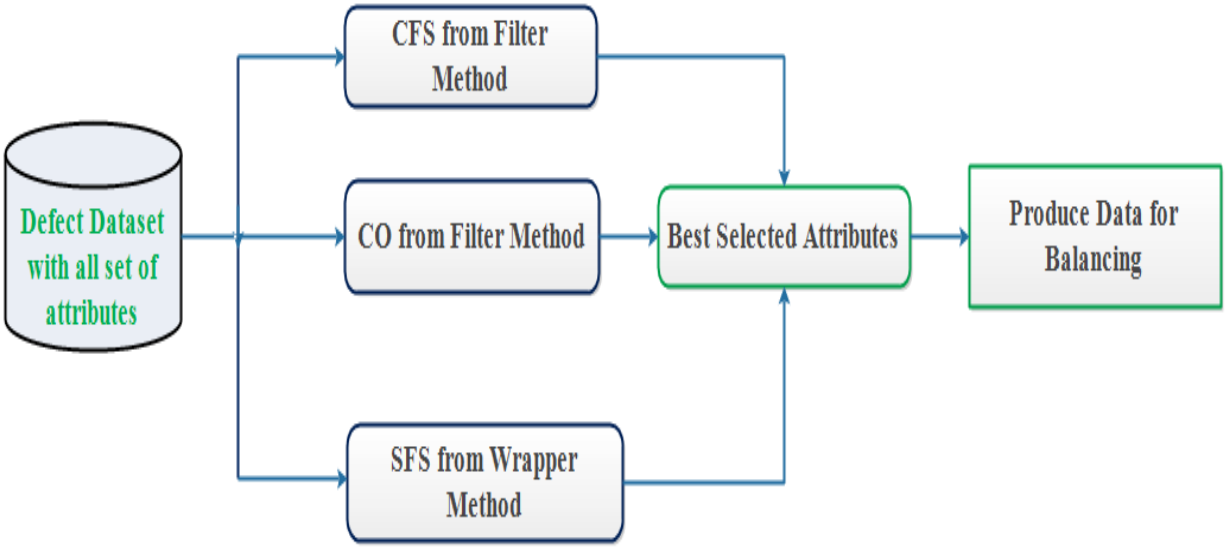

Figure 4.3: General Feature Selection workflow diagram

Correlation Based Feature Selection (CFS)

Correlation based Feature Selection(CFS) – is one of multivariate filter methods which assess a whole attribute subset based on correlation between independent variables and target variables. It deals with the attributes that have redundancy amongst the attributes. This technique finds the attributes that have high correlation with target attribute, but have low correlation with others

attributes by using correlation coefficient(Hall, 1999; Piao et al., 2012). This technique involves computing correlation relations between each pairs of attributes. Thus, the first attribute to be selected are based on the attributes that have highest correlation coefficient. Equation of this technique is defined as the following (Cunningham & Delany, n.d.; Liu, 2002; Wah et al., 2018):

$$M_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}}$$

Equation 4.2. Heuristic Merit of an Attribute Subset

where, M_S is the heuristic merit of an attribute subset containing k attributes, $\overline{r_{ff}}$ is the average inter-correlation between the attributes, and $\overline{r_{cf}}$ is the average of the correlation between the attributes and the dependent variable.

Correlation Filter (CO)

Correlation Filter(CO) is one of univariate filter method that determines the merit of an attribute by calculating its Pearson correlation coefficient to the dependent variable. It is statistical based approach. This technique selects the attributes based on their rank results. Pearson correlation coefficient equation is defined as the following(Guyon, 2003):

$$R(i) = \frac{cov(X_i, Y)}{\sqrt{var(X_i)var(Y)}}$$

Equation 4.3. Pearson Correlation Coefficient

Where, cov represent the covariance and var represent the variance.

Sequential Forward Selection(SFS)

Sequential Forward Selection (SFS) is one of deterministic wrapper method that depends on classification algorithms. This technique begins to operate from an empty set and progressively put attributes chosen by specified evaluation criteria. In every loop, the attribute to be included in the attribute set is chosen from the remaining available attributes of the attribute set which is not added to the attribute set. Therefore, the new extended attributes set should give a minimum misclassification when compared with the addition of any other attribute, and this work continues until a pre-defined number of the attributes are chosen. This technique is broadly important for their speed and simplicity (Andina, 2010). This technique is good in few number of attributes.

4.2.4. Data Balancing Technique

Data balancing technique is a methods used to balance class distribution of the datasets. It is a technique that is used to solve class imbalance problem of a datasets by equalizing number of classes exists in a dataset that appeared in different quantities and biased the datasets which leads to influence the prediction results of machine learning algorithms. This technique is mainly including two methods i.e., under sampling and oversampling. Under-sampling technique equalizes the class distribution of a datasets by randomly deleting the dataset samples from majority class, whereas the oversampling technique equalizes the class distribution of a datasets by increasing number of minority class instances by randomly duplicating the minority class instances. Most prominent example of oversampling technique is Synthetic Minority Over-Sampling Technique(SMOTE). SMOTE is mostly applied in software defect prediction and improve performance of prediction.

Synthetic Minority Over-Sampling Technique(SMOTE)

SMOTE is one of oversampling technique that produces synthetic samples for minority class of class distribution of the dataset instead of replacement oversampling. Synthetic instances/samples are produced by processing in feature space. In this, minority class is increased in sample by taking each sample and producing synthetic instances along the line segments that join any/all of the k minority class nearest neighbors. The technique starts by selecting k nearest neighbors after that synthetic instances are produced by calculating the difference between the feature vector of the instance samples under processing and with their nearest neighbor. After that, the technique multiplies the difference result by a random value between 0 and 1. Then add it with the feature vector under processing. Therefore, a random value is chosen along the line segment between two specific attributes/features. Consequently, SMOTE widest the data region area of the minority class instances and forces/pushes the decision region area of the class to begin to be more general (Chawla et al., 2002). This technique supports to prevent the overfitting problem.

For this study, SMOTE is used to balance class distribution of the defect with non-defect class. Because, in a dataset of AEEEM projects mostly non-defect class is very huge in number whereas defect class is very small in number that means defect class (encoded 1) is minority in number and non-defect class (encoded 0) is majority in number. In different ratio appearance they do not have equal influence on machine learning algorithms training. Therefore, SMOTE

balancing technique is used to duplicate ratio of minority class(defect) to equalize with ratio of majority class(non-defect) as both classes have equal impact on machine learning model.

4.3. Machine Learning Used for Model Building

The aim of this research is to build or construct effective software defect prediction model to classify software modules/instances into defect and non-defect based on software metrics inputs that measured and gathered from software systems during any phase of software development life cycle to mitigate maintenance cost and enhance quality of software using ensembles machine learning algorithms with compatible base learning algorithms. Therefore, models of machine learning based are built from ensembles machine learning algorithms such as Adaptive Boosting, Gradient Boosting, Extra Gradient Boosting, Random Forest, Extra Tree, Bagging and Stacking algorithms. These algorithms are uses different individual base classifiers such as Support Vector Machine, Naive Bayes, Decision Tree and K-Nearest Neighbor based on their characteristics. Effective final model is selected and deployed from these ensemble learning models for software defect prediction of binary class classification by evaluating and comparing each algorithms model performance using 10-fold cross validation and others performance evaluation metrics.

4.3.1. Adaptive Boost(AdaBoost) Algorithm

It is a one of most familiar boosting approach that build an ensemble by doing several iterations every time with different observation (instance) weights and change slightly to achieve more correct result to the errors returned by learners(classifiers) from previous occurring iterations (Freund et al., 1996; Schapire, 2013). Updating the weights of training instances in every iteration pushes the learning classifier to put more attention (value) on instances that incorrectly classified in previous iteration and less attention(value) on instances that correctly classified in previous iteration. That means, weights of incorrectly classified instances are become increased, but weights of error free classified instances are become decreased. This shows in next iteration wrong classified instance measure more heavily. AdaBoost uses the forecasts of several weak classifiers and provide a final prediction via aggregated voting on approaches. AdaBoost algorithm is applied for this software defect prediction because of this algorithm is not more affected by overfitting**,** it works with several classifier and boost their performance and it doesn't have parameters to adjust.

4.3.2. Gradient Boosting(GB) Algorithm

It is a generalization of adaptive boosting(AdaBoost) ensemble algorithm that can be construct by utilizing a diversity of loss function. Adaboost assigns the weight to incorrectly classified observations(instances) as next perform model (next iteration) predicts correctly whereas Gradient boosting utilizes gradient to fit new base learner. Gradient boosting (GBM) work by boosting techniques that aggregating previous predict errors with every subsequent numerous weak learner to provide strong classifier and it is decision tree based algorithms. Specially, Regression trees are act as a base classifier for both classification and regression problems and provides more robust, interpretable manner and competitive (Friedman, 2014). Gradient Boosting algorithm is applied for this software defect prediction because of this algorithm gives high prediction accuracy, it can improve the quality of predicting (fitting) the base learners(classifiers), it can classify complex data, it can try to control both under-fitting and overfitting problem and it is very changeable/flexible.

4.3.3. Extreme Gradient Boosting (XGBoost) Algorithm

It applies gradient boosting tree approaches to build ensemble technique (Chen & Guestrin, 2016). It is a gradient boosting tree ensemble that employed the second-order differentials (derivatives) of the loss function to get the optimal base learner more efficiently and accurately. Whereas Gradient Boosting utilizes gradients to fit a new base classifier, XGB uses the second-order derivatives gradients. This machine learning approach is decision trees based by using gradient boosting framework algorithmic improvement (i.e., Weighted Quantile Sketch, Sparsity Awareness, Regularization) and hardware and software optimization methods (i.e., Tree Pruning, Parallelization). These enhancements produce higher results by using few computing resources within short time. XGB build reliable and accurate model by creating effective classifier from a group of weak decision tree algorithms. then create decision tree ensemble that utilizes the output of the weak classifier in the last prediction. XGB algorithm is applied for this software defect prediction due to above reasons and also it models performance is good and it doesn’t more affected by overfitting.

4.3.4. Bagging Meta-Estimator Algorithm

It is an ensemble machine learning algorithm that follows a bagging techniques procedure to enhance accuracy and stability of machine learning classifier by aggregating prediction output of several weak learners(Bbeiman, 1996). This algorithm work with both regression and classification problems. Bagging algorithms are more appropriate and effective with unstable machine learning techniques that means when small changes happen on training set the consequence of that changes is huge on predictions output. The following steps are concluded bagging meta-estimator algorithms:

1) Generate random subsets(sub-samples) from dataset or bootstrapping (sampling of dataset)
2) The subsets(sub-samples) of data contains all attributes(features)
3) A user-specified classification algorithms of base estimator is train with each of generated subsets in previous step
4) Prediction from every one classifier(model) are aggregated(combined) to obtain final output and use voting to forecast.

Bagging algorithm is applied in this study due to it improve accuracy by using combination of several weak model to provide strong classifier, it produces stable classifier from unstable, it minimizes a variance, it is a better than individual strong classifier and it can work with numerous machine learning classifiers as a base learner.

4.3.5. Random Forests (RF) Algorithm

It is a special case of Bagging techniques that holds a collection of decision tree structure algorithm. RF chooses random attributes in order to construct bootstrap learner using decision trees (Breiman, 2001). RF constructed from numerous decision trees by choosing a subset of data and attributes randomly. Randomly selects a subset of observation from chosen variables and given to decision tree learning. Then, RF algorithm chooses the prediction that has highest votes from all decision trees output in the forest to produce final prediction. Random forest combine output from several trees and this combination will provide high accuracy on prediction. Random forest algorithm uses randomization in two important places: First, every decision tree is executed/trained by taking random sub-sample with replacement from original dataset. Second, when executing individual decision trees algorithms (at every node) random subset of attributes is used to find best splits. Making randomization is minimizing the

correlations between decision trees, which enhance performance of prediction. Additionally, working with randomness on datasets and attributes are decrease the risk of model overfitting. RF uses decision tree algorithm as a base estimator and assigned by default in RF algorithms. Random Forest algorithm is applied in this study because of it improve accuracy by minimizing variance, it minimized error, it doesn't have overfitting problem and it performance is good on imbalanced datasets.

4.3.6. Extra Trees (ET) Algorithm

It is one of bagging technique ensemble algorithm that uses randomization for selection of features and cut-point while splitting a tree (extremely randomized tree) and almost the same to Random Forest, but with additional randomness. ET algorithm is different from RF algorithm in two things: The first, every decision tree is constructed by using entire dataset, and the second at every node in the tree, ET algorithm chooses the splits randomly without finding the best splits. ET algorithm uses the top down method to construct unpruned trees classifier ensemble technique. This algorithm is used for supervised regression and classification problems and it accuracy and computational performance is good (Geurts et al., 2006). Extra Tree algorithm is applied in this study because of above mentioned reasons and also uses further randomization to improve accuracy by minimizing variance and it minimizes bias by constructing decision tree by using entire dataset.

4.3.7. Stacking Algorithm

It is one of an advanced ensemble machine learning algorithms that combines several different machine learning algorithms through meta learning algorithm that is either a meta-regressor or meta-classifier. The base weak learner level machine learning algorithms are executed/trained on the whole training dataset, and after that the meta learner(model) is accept output prediction from base weak learner level algorithms as inputs. The base weak learner models are occurred at level-0 model and meta learner that aggregates and accept the output predictions of base weak learners as input are occurred at level-1 model to produce final prediction. level-0 algorithms are diverse. Stacking algorithms can minimize the variance or bias based on the learning algorithms applied in level-0. Stacking algorithms is operating effective ensemble technique in which the output predictions that are produced by applying multiple various machine learning algorithms are employed as inputs to next level algorithm (meta learner). This meta learner is trained to combine the learner predictions in best manner to provide a final prediction(Funda et

al., 2017). This algorithm performs it task parallel and it uses heterogeneous base weak learner algorithms. Stacking algorithm is applied in this study because of it can produce improvement performance, it minimizes a variance and build high robust model by joining the predictions of several learners.

Above all mentioned ensembles algorithms are uses different machine learning algorithms on their base classifier based on their characteristics. So, machine learning algorithms such as Support Vector Machine(SVM), Naive Bayes(NB), Decision Tree(DT) and K-Nearest Neighbor(KNN) are applied on base classifiers based on their benefits. From ensembles algorithms such as Gradient Boosting, Extreme Gradient Boosting (XGBoost), Random Forest and Extra Tree algorithms are uses only decision tree(DT) algorithms on their base classifier whereas ensembles algorithms such as Adaptive Boosting(AdaBoost), Bagging Meta-Estimator(Bagging) and Stacking algorithms are may uses SVM, NB, DT, KNN and also others algorithms on their base classifier depending on effectiveness of those algorithms have on the prediction power and performance improvement.

4.4. Model Evaluation Procedure

Model evaluation is the last process in construction of proposed solution that is used to approximate the generalization performance of the model on whole or sample dataset. In this research, model evaluation is implemented on the whole datasets of each software defect projects to examine whether the predictive model is correctly trained or not. In this research, as described in chapter three cross validation model evaluation procedure are used to assess performance of the model with other performance measurements. Therefore, k-fold cross validation methods are applied in this study with the k=10 to compare the result and identify best performance model from all models. And 10-fold cross validation performance of the model is evaluated and worked with other performance metrics such as accuracy, recall, precision, confusion matrix, F-measure, AUC (Area Under ROC curve).

4.5. Model Prototype

In this research, we intended to predict or classify software modules/instances into defect and non-defect based on software metrics inputs that measured and gathered from software systems in quantitative form during any phase of software development life cycle to mitigate maintenance and development cost, and enhance quality of software. Because, when software testing performing at later stage the faults probable happen and identified in developed software

have effect on software quality, reliability and maintenance cost(Rawat & Dubey, 2012). And this impact cost can affect our life and budget loss. Therefore, controlling the defect during any phase of software development life cycle and early detection will support software developers, testers and software companies and deliver quality software(Rawat & Dubey, 2012). And this mechanism is easily performing by using software defect prediction approach that build a prediction models from machine learning algorithms. Software defect prediction models are an approach used to identify software modules into defect and non-defects. Then after the models built from machine learning algorithms, users such as software developers, testers and software companies are uses this models to predict new dataset in real world. To facilitate this models for the users we trained, compared and selected best performance classifier model for deployment on the flask server using GUI interfaces called HTML Webpage.

HTML webpage form are used to insert new data as input and this HTML webpage is connected to the flask server and the flask server is connected to the best performance model that is selected in model performance evaluation stage and finally classify new software modules into whether defect or non-defect output.

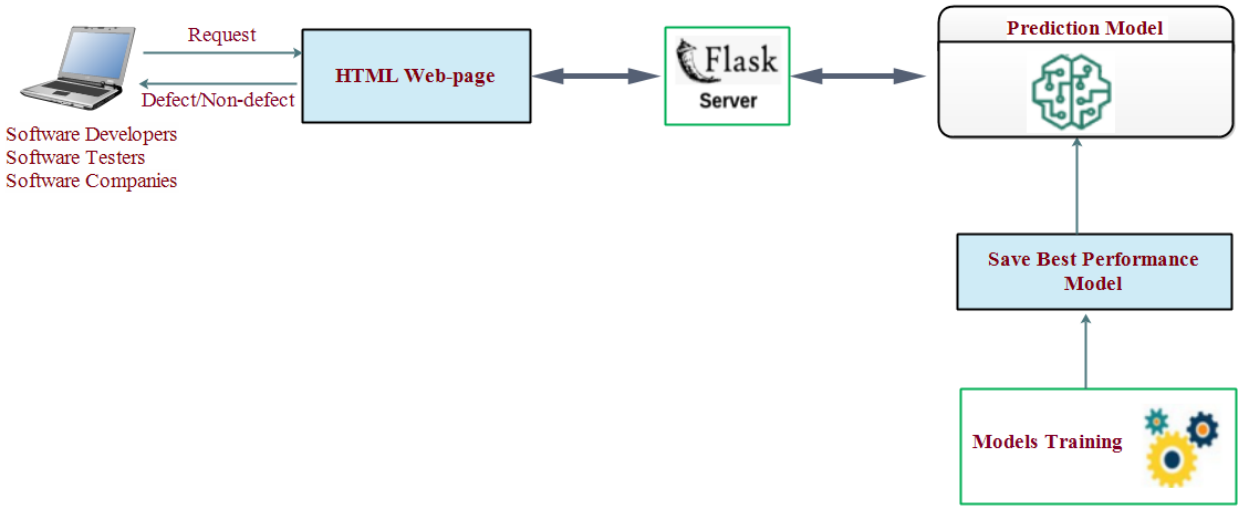

Figure 4.4: Deployment Architecture for Software Defect Prediction

CHAPTER FIVE

5. IMPLEMENTATION AND EXPERIMENT

In this chapter, we discuss implementation and evaluation of proposed system of software defect prediction and required tools for performing experimentation to accomplish proposed system of software defect prediction that enhance quality of a software and mitigate maintenance costs by providing early detection of defect prone software modules before actual testing processes start by using software metrics as input to ensemble machine learning algorithms. The aim of study is used for software companies, developers and testers to predict whether software modules are defect or non-defect based on software metrics of a software dataset that describe property of software project in quantitative form. Hereunder we firstly describe implementation tools, software package and working environment before real implementation of proposed system.

5.1. Working Environment

To achieve the aim of this research, the following different mechanisms, development tools, environment setup and programming languages are used.

Hardware Specification

Hardware specification used in this research are: Laptops with the following specification:

- Processor: Intel(R) Core(TM) i5-4300U CPU @ 1.90GHz 2.50 GHz
- Installed RAM: 8.00 GB(Gigabyte)
- System type: 64-bit operating system, x64-based processor
- Hard Disk Storage: 1TB(Terabyte)

Software Specification

- Operating System: Windows 10 Enterprise is used to hold, manage and control the whole computer system.
- Anaconda IDE is used for machine learning programming implementations.
- MS Office is used for writing a research documents and prepare a data.
- Edraw Max is used for drawing and design of every diagram and figure.

5.1.1. Implementation Environments

To implement the aim of the research, python programming language is used that is currently popular in machine learning and data science technology. Because python programming language is the best in categorical and numerical classification or prediction and also it contains different important package, library and frameworks. This programming language is crucial in data processing science starting from data preprocessing stage up to model development stage. As a result, for this implementation the following packages, tools and libraries are used.

Table 5.1: Explanation of Tools, Module packages used in the implementation and in Deployment.

Tools	***Version***	***Description***
Package Manager and Environments Tools		
Anaconda Navigator[12]	4.10.1	It is a graphical user interface that help to launch developing applications and manage all packages of the environment without command line.
Jupyter Notebook[13]	6.0.1	It is an open-source web program application that support us for creating machine learning model implementation, sharing document, visualizations, data cleaning, equations, numerical simulation, transformations and soon. It works with anaconda navigator.
Python[14]	3.7.0	It is an object-oriented, high-level and interpreted programming language which has a dynamic semantics. It is a strength programming language that is used to develop machine learning model and application and easy to learn.
Data preparation, Data balancing and Training Tools and Packages		
Microsoft Excel	2016	It is one of individual package of Microsoft Office that was used for this research for data preparation purpose as suitable for implementation

[12] https://docs.anaconda.com/anaconda/navigator/
[13] https://jupyter.org/
[14] https://www.python.org/

Pandas[15]	1.3.4	It is an open-source package of python language that is used for data reading, writing, manipulations and handling data frame. It prepares high performance data analysis and manipulation using powerful and fast data structure. It supports us to import dataset in multiple file formats and easy to use.
Scikit-learn[16]	1.0.1	It is an open-source machine learning package that is used for regression, classification and clustering purpose in machine learning. It is also used for predictive result analysis in machine learning in simple and efficient way with built upon library such as Scipy, Matplotli and Numpy. It is commonly known as Sklearn
Numpy[17]	1.21.2	It is a fundamental python module used for scientific computing. This package is used to provide and support multidimensional array object, strings and numbers, and provide tools working with them.
Xgboost	1.5.1	It is a package module used to import or train Extreme Gradient Boosting algorithm
Imblearn[18]	0.8.1	It is a data balancing package module that is used to balance class distribution of the datasets. It is a module that is used to solve class imbalance problem of a datasets by equalizing number of classes exists in a dataset that appeared in different quantities.
Data and Result Visualization Package		
Matplotlib[19]	3.5.0	It is a plotting package used for data and predictive result visualization.

[15] https://pandas.pydata.org/
[16] https://scikit-learn.org/stable/
[17] https://numpy.org/
[18] https://imbalanced-learn.org/stable/
[19] https://matplotlib.org/

Seaborn[20]	0.11.2	It is a data visualization package that based on matplotlib package. It is used to draw attractive graphics of data and result.
Deployment Tools		
Flask Server[21]	1.1.2	It is a web server application framework which is implemented by python that is used to deployed and access developed model And it is easier for web developer to implement application.

5.2. Dataset Description

For this research, datasets of AEEEM projects are used that are gathered from different version of software system like eclipse and apache which are publically available online and was collected by (Ambros & Lanza, 2010). These project datasets were developed by java object oriented programming and have various software modules and software metrics that are based on class level projects and evolution version of a system. In AEEEM datasets, five software projects exist in which each software projects dataset has totally 62 attributes. That means, one dependent attribute and 61 independent software metrics attributes which are prepared from combination of 17-metrics from Source code metrics of CK (Chidamber and Kemerer) and OO (Object Oriented) metrics, 5-metrics from previous defects, 5-metrics from Entropy of change metrics, 17-metrics from Churn of Source code metrics and 17-metrics from Entropy of Source code metrics.

AEEEM datasets have selected because of these datasets has a class imbalance and high feature dimensionality problem. The following table represents the name of features, feature categories and it description.

Table 5.2: Dataset Attributes Description

Software Metrics Categories	**No.**	**Software Metrics(Features)**	**Description**
	1.	ck_oo_numberOfPrivateMethods	Number of private methods of source code

[20] https://seaborn.pydata.org/
[21] https://realpython.com/tutorials/flask

Source Code Metrics(CK_OO metrics)	2.	ck_oo_numberOfPublicAttributes	Number of public attributes of source code
	3.	ck_oo_noc	number of children of source code
	4.	ck_oo_wmc	weight method count of source code
	5.	ck_oo_fanOut	Number of other classes referenced by the class of source code
	6.	ck_oo_numberOfLinesOfCode	number of line code of source code
	7.	ck_oo_numberOfAttributesInherited	Number of attributes inherited of source code
	8.	ck_oo_numberOfMethods	Number of methods of source code
	9.	ck_oo_dit	depth of inheritance tree of source code
	10.	ck_oo_fanIn	Number of other classes that reference the class of source code
	11.	ck_oo_lcom	lack of cohesion in methods of source code
	12.	ck_oo_rfc	response of class of source code
	13.	ck_oo_cbo	coupling between objects of source code
	14.	ck_oo_numberOfAttributes	Number of attributes of source code
	15.	ck_oo_numberOfPrivateAttributes	Number of private attributes of source code

	16.	ck_oo_numberOfMethodsInherited	Number of methods inherited of source code
	17.	ck_oo_numberOfPublicMethods	Number of public methods of source code
Churn of Source Code Metrics(WCHU metrics)	1.	WCHU_numberOfPublicAttributes	Churn of Number of public attributes
	2.	WCHU_numberOfAttributes	Churn of Number of attributes
	3.	WCHU_fanIn	Churn of Number of other classes that reference the class
	4.	WCHU_numberOfPrivateMethods	Churn of Number of private methods
	5.	WCHU_numberOfMethods	Churn of Number of methods
	6.	WCHU_numberOfPrivateAttributes	Churn of Number of private attributes
	7.	WCHU_noc	Churn of number of children
	8.	WCHU_wmc	Churn of weight method count
	9.	WCHU_dit	Churn of depth of inheritance tree
	10.	WCHU_numberOfAttributesInherited	Churn of Number of attributes inherited
	11.	WCHU_fanOut	Churn of Number of other classes referenced by the class
	12.	WCHU_lcom	Churn of lack of cohesion in methods
	13.	WCHU_rfc	Churn of response of class

	14.	WCHU_numberOfPublicMethods	Churn of Number of public methods
	15.	WCHU_cbo	Churn of coupling between objects
	16.	WCHU_numberOfMethodsInherited	Churn of Number of methods inherited
	17.	WCHU_numberOfLinesOfCode	Churn of number of line code
Entropy of Source Code Metrics(LDHH metrics)	1.	LDHH_lcom	Entropy of lack of cohesion in methods metrics
	2.	LDHH_fanIn	Entropy of Number of other classes that reference the class
	3.	LDHH_numberOfPublicMethods	Entropy of Number of public methods
	4.	LDHH_numberOfPrivateAttributes	Entropy of Number of private attributes of
	5.	LDHH_numberOfPublicAttributes	Entropy of Number of public attributes
	6.	LDHH_numberOfPrivateMethods	Entropy of Number of private methods
	7.	LDHH_numberOfAttributesInherited	Entropy of Number of attributes inherited
	8.	LDHH_noc	Entropy of number of children
	9.	LDHH_wmc	Entropy of weight method count
	10.	LDHH_numberOfAttributes	Entropy of Number of attributes

	11.	LDHH_numberOfLinesOfCode	Entropy of number of line code
	12.	LDHH_dit	Entropy of depth of inheritance tree
	13.	LDHH_fanOut	Entropy of Number of other classes referenced by the class
	14.	LDHH_numberOfMethodsInherited	Entropy of Number of methods inherited
	15.	LDHH_rfc	Entropy of response of class
	16.	LDHH_cbo	Entropy of coupling between objects
	17.	LDHH_numberOfMethods	Entropy of Number of methods
Previous Defects Metrics	1.	numberOfNonTrivialBugsFoundUntil:	non-trivial bugs (severity>trivial)
	2.	numberOfCriticalBugsFoundUntil:	critical bugs (critical or blocker severity)
	3.	numberOfHighPriorityBugsFoundUntil:	high priority bugs (priority and default)
	4.	numberOfMajorBugsFoundUntil:	major bugs (severity>major)
	5.	numberOfBugsFoundUntil:	All bugs
Entropy of Change Code Metrics	1.	CvsWEntropy	Weight History of Complexity Metric
	2.	CvsEntropy	History of Complexity Metric
	3.	CvsLinEntropy	Linearly Decayed History of Complexity Metric

	4.	CvsExpEntropy	Exponentially Decayed History of Complexity Metric
	5.	CvsLogEntropy	Logarithmically Decayed History of Complexity Metric
Target Attributes		Defects	This is an attributes of the class that used to identify the classes into defect and non-defect

5.3. Data Preprocessing Implementation

Under data preprocessing implementation, data cleaning and controlling missing value implementation, data normalizations implementation, feature selection implementation and data balancing implementation are included as it is proposed in chapter four. As the dataset is suitable and acceptable for machine learning algorithms, models are constructed using python programming language.

Before starting to do preprocessing stage necessary library and important datasets are imported. Therefore, we have five datasets of AEEEM projects and those are loaded sequentially based on the different time of execution using input array of the dataset.

```
import numpy as np
import pandas as pd
import re, os,time

#This is a method of Easily Reading the Files
path='../../../AEEEM_Data/'
print('AEEEM DATASET')
for i,file in enumerate(os.listdir(path)):
    print(str(i+1)+'. '+file.replace('.csv',''),end=',')
    data = pd.read_csv(path+file)#loading dataset
    print(len(data),'Instances')
opt = input('Input Number of File: '); opt = int(opt)-1
for i,file in enumerate(os.listdir(path)):
    if i != opt: continue
    print('\nDATASET:',file.replace('.csv','').upper(),'have selected','\n')
    data = pd.read_csv(path+file)#loading dataset
```

Figure 5.1: Implementation of loading different datasets in different execution

5.3.1. Data Cleaning implementation

A method of data cleaning is used to maintain dataset from error, noisy, incomplete, missing, garbage, corrupted and duplicate**.** However, the datasets don't have the problem of missed value, when it is checked through methods represented below and the data is clean it does not have a problem.

```
data.isnull()  data.isna().sum()  data.isna().any().sum()
```

Figure 5.2: Implementation of checking missing value

After datasets are checked for missing values, it is split into independent variables and dependent variables. The goal is to perform the next preprocessing activity and working with machine learning algorithms so as to implement the proposed architecture.

```
#Splitting Dataset into Independent Attribute and Target Attribute
Y=data['defects']# Target Attribute
X = data.drop(columns = 'defects')#Independant Attribute
```

Figure 5.3: Implementation of Splitting Dataset into Independent Attribute and Target Attribute

5.3.2. Data Normalization Implementation

Data Normalization is an extremely crucial part for model development of machine learning technique which transforms the values of variables that appeared in different magnitudes/scales into the same order magnitudes as all features have equal influence on the building model and also used to controlling outliers of the values. The dataset used in this research have the problem of different scales on the value of features and outliers value on some features as it has been discussed in chapter four. Therefore, Z-Score has been used as a data normalization technique, for this study to make the values of variables on the equal magnitudes/scales.

```
# Z-Score data normalization(Standardization)
from sklearn import preprocessing
# create a scaler object
standard_scaler = preprocessing.StandardScaler()
# fit and transform the data
X_standard = pd.DataFrame(standard_scaler.fit_transform(X), columns=X.columns)
X_standard
```

Figure 5.4: Implementation of Z-score(Standardization) data normalization

5.3.3. Feature Selection Implementations

The results of preprocessing activity that has been done at the first stage are further used as a dataset, for the next task of preprocessing such as feature selection and data balancing as to provide suitable and quality dataset for machine learning training.

Feature selection technique is a technique to analysis and selects most important attributes for machine learning algorithms. Those selected attributes are crucial to increase performance and produce effective model for prediction, reduce computational cost, decrease overfitting and increase interpretability of the models. This technique is used to solve the problem of high feature dimensionality of the datasets as previously discussed in chapter four. For this reason, three feature selection techniques such as Correlation Based Feature Selection (CFS), Sequential Forward Selection(SFS) and Correlation-Filter(CO) have been implemented based on their effectiveness.

Correlation Based Feature Selection (CFS)

It is one of filtering methods/approaches which evaluate and analysis the whole attribute subset based on correlation between independent variables and target variables. It is used to choose/select best attribute from full set of original features to improve performance of the machine learning algorithms. Python programming language which is manifested in the form of the following python code is implemented.

```
from CFS import cfs, merit_calculation #This Library/Module is used for Correlation Based Feature Selection
from sklearn.preprocessing import MinMaxScaler #This is used for Data Normalization
scaler = MinMaxScaler()
X_min_max=scaler.fit_transform(X)

# Using Best First Searching Method of CFS
Sel_feat = cfs(X_min_max,Y) # This Method is used to call or import cfs method from CFS Module
Sel_feat = Sel_feat[Sel_feat!=-1]
Sel_feat

# Print the names of the features selected
feature_names_sel = data.columns[Sel_feat]
feature_names_sel
```

Figure 5.5: Sample code of Correlation Based Feature Selection

This feature selection searches a number of best subset attributes based on correlation coefficient by using Best First Searching Method of correlation based feature selection. The full

implementation of correlation based feature selection is working with others sample code such as merit calculation implementation, symmetrical uncertainty calculation implementation, information gain implementation and conditional entropy calculation implementation and they are depicted on **Appendix C**.

Correlation-Filter(CO)

It is statistical based approach of univariate filter method which decides the merit of an attribute by calculating its Pearson correlation coefficient to the dependent variable. Therefore, it chooses the variables based on their rank results. It gives the relationship that each variable has with target as indicated in the preceding chapter. This feature selection implementation is represented as follows.

```
# Brute Force Method to find Correlation between features
def correlation(data, threshold=None):
    # Set of all names of correlated columns
    col_corr = set()
    corr_mat = data.corr()# Correlation Analysis
    for i in range(len(corr_mat.columns)):
        for j in range(i):
            if (abs(corr_mat.iloc[i,j]) > threshold):
                colname = corr_mat.columns[i]
                col_corr.add(colname)
    return col_corr

#This is used to identify and extract correlated features using threshold
correlated_features = correlation(data=X_standard, threshold=0.8)

#This is used to remove correlated attributes
X_Selected=X_standard.drop(correlated_features,axis=1)

#This is used to display selected attributes
X_Selected.columns
```

Figure 5.6: Implementation of Correlation Filter

In implementation of correlation filter we used Brute Force method by deciding the threshold point of limiting (threshold=**0.8**) to find and identify correlated attributes and select best attributes from original sets.

Sequential Forward Selection(SFS)

It is one of the wrapper methods which depend on machine learning classification algorithms. This technique selects best attributes from original sets by using selected ensemble machine learning algorithms that is used for model development such as RF, ET, GB, XGBoost, Bagging,

AdaBoost and Stacking. These feature selection methods is appropriate for this study as proposed in the previous chapter has been implemented for selecting features as indicated in the following figure.

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from matplotlib.ticker import MaxNLocator
from sklearn.model_selection import KFold,cross_val_score
from sklearn.ensemble import ExtraTreesClassifier

# This is splitting datasets into Kfold
kf = KFold(n_splits=10, shuffle=True, random_state=1)

# This is Single Machine Learning Algorthim working with Sequential Forward Selection
clf = ExtraTreesClassifier(n_estimators=100, n_jobs=-1)

# This is Building Sequential Forward Selection with ML algorithm and cross validation
max_k = X_standard.shape[1]
sfs_forward = SFS(clf,k_features=(1, max_k),forward=True,floating=False,scoring='accuracy',cv=kf,n_jobs=-1)

# Sequential Forward Selection is Fitting Feature and Target
sfs_forward = sfs_forward.fit(X_standard,Y)

#This is Print best selected features from Original set
sfs_forward.k_feature_names_
```

Figure 5.7: Sample code of Sequential Forward Selection

5.3.4. Data Balancing Implementation

Data balancing technique is a method used to balance or equalize class distribution of the datasets that appeared in different ratio. It makes all classes of the datasets to have equal influence on model training. As it has been discussed in chapter three and four, the datasets used have the problem of data imbalance. To solve that problem, SMOTE data balancing approach has been used. SMOTE data balancing approach is mostly applied and suggested in software defect prediction to improve performance of the prediction activity. For this study, SMOTE balancing approach is used to balance class distribution of the defect with non-defect class. Therefore, SMOTE balancing technique is applied and implemented to duplicate ratio of minority class(defect) to equalize with ratio of majority class(non-defect) as both classes have equal impact on machine learning model. This approach is performed after best set of attributes is selected by feature selection techniques and it is the last preprocessing stage that is performed

before training the machine learning algorithms. This implementation approach is represented as the following figure.

```
from collections import Counter
from imblearn.over_sampling import SMOTE # Data Balancing Library

# Generating the Data to Balance
smote=SMOTE(sampling_strategy =1,random_state=1)
X_smote,Y_smote=smote.fit_resample(X_selected,Y)

# Display Equal ratio of the Class
Counter(Y_smote)
```

Figure 5.8: Implementation of SMOTE Data balancing technique

5.4. Machine Learning Used for Model Building Implementation

To build effective ensemble based machine learning model for software defect prediction by using the proposed algorithms such as Adaptive Boosting, Gradient Boosting, Extra Gradient Boosting, Random Forest, Extra Tree, Bagging and Stacking algorithms, important steps such as preprocessing, feature selection and data balancing activities have been done. Furthermore, training ensemble machine learning models with base classifier and evaluation of the developed models has been taken place by importing important libraries for all of these tasks as indicated below.

```
import numpy as np
import pandas as pd
import re, os,time
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler,LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier,BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier,AdaBoostClassifier,StackingClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV,cross_val_score,cross_val_predict,KFold
from sklearn import metrics
from sklearn.metrics import *
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from CFS import cfs, merit_calculation
from CFS_ForwardSearch import CFS_FS
from matplotlib.ticker import MaxNLocator
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
from imblearn.over_sampling import SMOTE
```

Figure 5.9: Importing necessary Library for model development

After important libraries are loaded for the preprocessing, feature selections and data balancing tasks thereby completing the required tasks, the proposed ensemble machine learning algorithms with base learners are trained by taking the final appropriate datasets that are produced by the last preprocessing activity for defect prediction models. The models built on the basis of the last appropriate datasets are presented hereafter.

Random Forest Implementation

This is one of supervised ensemble machine learning algorithms that works by Bagging approach and by holding a collection of decision tree structure algorithm on several sub-sample (bootstrap) of the datasets which then after uses average predication. RF chooses random attributes in order to construct bootstrap learner by using decision trees. The algorithm implementation instantiation and fitting the model to train our datasets by importing *RandomForestClassifier*() module from sklearn ensemble library is demonstrated in the following figure.

```
from sklearn.ensemble import RandomForestClassifier
#Define model
model=RandomForestClassifier(random_state=1)
#Fit to train model
model.fit(X_smote,Y_smote)
```

Figure 5.10: Sample code of Random Forest Classifier and It is Model Fitting

Extra Tree Implementation

Extra Tree is also one of supervised ensemble machine learning algorithms that work by Bagging approach. It uses randomization for the selection of features and cut-point while splitting a tree (extremely randomized tree). It is almost the same as Random Forest classifier, but with additional randomness. It uses decision tree algorithm on it base classifier. The algorithm implementation instantiation and fitting the model to train our datasets by importing *ExtraTreeClassifier*() module from sklearn ensemble library is demonstrated in the following figure.

```
from sklearn.ensemble import ExtraTreesClassifier
#Define model
model=ExtraTreesClassifier(random_state=1)
#Fit to train model
model.fit(X_smote,Y_smote)
```

Figure 5.11: Sample code of Extra Tree Classifier and It is Model Fitting

Gradient Boosting Implementation:

Gradient Boosting works by boosting approach that aggregating previous predict errors with every subsequent numerous weak learner to provide strong classifier and it is decision tree based algorithms. Gradient boosting utilizes gradient to fit new base learner and it can be construct by utilizing a diversity of loss function. The algorithm implementation instantiation and fitting the model to train our datasets by importing *GradientBoostingClassifier*() module from sklearn ensemble library is presented in the following figure.

```
from sklearn.ensemble import GradientBoostingClassifier
#Define model
model=GradientBoostingClassifier(random_state=1)
#Fit to train model
model.fit(X_smote,Y_smote)
```

Figure 5.12: Sample code of Gradient Boosting Classifier and It is Model Fitting

XGBoosting Implementation:

XGBoosting works also by boosting approach and it is a gradient boosting tree ensemble that employed the second-order differentials(derivatives) of the loss function to get the optimal base learner more efficiently and accurately. XGBoosting uses the second-order derivatives gradients and decision trees based algorithms. The algorithm implementation instantiation and fitting the model to train our datasets by importing *GradientBoostingClassifier*() module from sklearn ensemble library is presented in the following figure.

```
from xgboost import XGBClassifier
#Define model
model=XGBClassifier(objective ='binary:logistic',use_label_encoder=False,random_state=1)
#Fit to train model
model.fit(X_smote,Y_smote)
```

Figure 5.13: Sample code of XGBoosting Classifier and It is Model Fitting

Bagging Meta-Estimator Implementation

Bagging Meta-Estimator is one of supervised ensemble machine learning algorithm that carrying out a bagging approach procedure to enhance accuracy and stability of machine learning classifier by aggregating prediction output of several weak learners. It is best with unstable machine learning algorithms (Example: Decision Tree) that means when small changes happen on training set, the consequence of that changes are huge on predictions output. Therefore, it uses user-specified classification algorithms for base estimator. For this study, we try to use different base classifiers such as SVM, NB, DT and KNN based on their characteristics and effectiveness are used as it is proposed in chapter four. Therefore, when they are compared by their characteristics and result benefits; the DT algorithm is best due to its unstable property. And also other DT based ensembles machine learning algorithms such as RF, ET, GB and XGBoost are used on base classifier of bagging meta estimator due to their unstable

characteristics on their base classifier. Bagging meta-estimator uses DT, RF, ET, GB and XGBoost on it as a base classifier. Sample code implementation of bagging meta-estimator is represented as the following with ET base classifier.

```
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
#Define model
model= BaggingClassifier(ExtraTreesClassifier(random_state=1),random_state=1,n_jobs=-1)
#Fit to train model
model.fit(X_smote,Y_smote)
```

Figure 5.14: Sample code of Bagging Classifier by using ET on Base Learner and It is Model Fitting

Adaptive Boosting Implementation:

Adaptive Boosting, is one of supervised ensemble machine learning algorithm that works by boosting approach which construct an ensemble by doing several iterations every time with different observation (instance) weights and change slightly to achieve more correct result to the errors returned by learners(classifiers) from previous occurring iterations.

This algorithm uses different user-specified base classifier algorithms. Therefore, for this study, different base classifier such as SVM, NB, DT and KNN are used as proposed in chapter four. When we compare them by their characteristics and result benefits, the DT algorithm is the best among them. And also other DT based ensembles machine learning algorithms such as RF, ET, GB and XGBoost are used on base classifier of adaptive boosting algorithm. Adaptive boosting algorithm uses RF, ET, GB and XGBoost on it as a base classifier. Sample code implementation of adaptive boosting algorithm is represented as the following with ET base classifier.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import ExtraTreesClassifier
#Define model
model= AdaBoostClassifier(ExtraTreesClassifier(),random_state=1)
#Fit to train model
model.fit(X_smote,Y_smote)
```

Figure 5.15: Sample code of Adaptive Boosting Classifier by using ET on Base Learner and It is Model Fitting

Stacking Implementation:

Stacking is included under heterogeneous ensembles learning algorithm that combines several different machine learning algorithms through meta learning algorithm. In this algorithm different base weak learner is executed on the whole datasets and the output of that base weak learner is inputted to final estimator and train for prediction. For this study, different base classifier such as SVM, NB, DT and KNN are used as per the proposal in chapter four. When these classifiers are compared by their characteristics and result benefits, the DT algorithm is the best from them. And also other DT based ensemble machine learning algorithms such as DT, RF, ET, GB and XGBoost are there but for stacking ensemble we used only three base learners such as ET, RF and DT because of computational performance of stacking is very high with all these five algorithms (DT, RF, ET, GB and XGBoost) on base learners due to the property of stacking is heterogeneous. So that stacking algorithm uses DT, RF and ET on it is base weak learning and default algorithm on it is final meta learner (i.e. Logistic Regression). Sample code implementation of stacking algorithm is represented as indicated in the following figure with base weak classifier like ET, RF and DT, and default final estimator.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
#Define model
estimators = [('rf', RandomForestClassifier(random_state=1)),
              ('et',ExtraTreesClassifier(random_state=1)),
              ('dt',DecisionTreeClassifier(random_state=1))]
model= StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
#Fit to train model
model.fit(X_smote,Y_smote)
```

Figure 5.16: Sample code of Stacking Classifier with default Final Estimator and It is Model Fitting

5.5. Model Evaluation Procedure Implementation

After the steps of the first preprocessing and the second preprocessing like feature selection and data balancing, appropriate datasets are obtained for machine learning training and model construction. Based on what has been trained, several models of software defect prediction based on proposed ensemble machine learning algorithms have been developed. To use these developed models for software module classifications as either defective or non-defective, evaluation is needed to determine the performance of every developed models as important and basic task. Model evaluation is the last process and it is an important work of assessing performance of the models on the whole datasets. In this study, we have five software defect datasets of AEEEM projects where each software defect dataset should be evaluated based on all developed models.

To evaluate performance of all developed models, ten-fold cross validation data splitting technique has been applied as a result of which the performance of all developed models is evaluated using evaluation metrics provided by confusion matrix. The metrics include accuracy, recall, precision, F-measure, AUC (Area Under ROC curve) as it is discussed in chapter three and four. In 10-fold cross validation, cross validation is applied on the whole datasets where 9-fold of them is used for training and 1-fold for testing. k-fold cross validation is implemented by instantiating *cross_val_score()* method with different parameters such as classifier(estimator), independent variable, dependent variable and number of folds, scoring metrics and n_jobs. *cross_val_predict()* method has also been used with different parameters like classifier(estimator), independent variable, dependent variable and number of folds for cross validation prediction on testing folds dataset. Both these methods are imported from sklearn library. *confusion_matrix()* method for other metrics calculations such as accuracy, recall, precision, F-measure, AUC (Area Under ROC curve) is implemented and results are represented in the form of text classification result by using *classification_report()* method. All used methods are imported from sklearn package library and their implementation is represented in the figure below.

```
from sklearn.model_selection import KFold,cross_val_score

#This is used to splitting dataset into 10-folds by using shuffle
kf = KFold(n_splits=10, shuffle=True, random_state=1)

#This is used to instantiate 10-fold cross validation with ML algorithm and Accuracy
cv_results_acc= cross_val_score(model,X_smote,Y_smote, cv=kf,scoring="accuracy",n_jobs=-1)

#This is used to Display Accuracy of 10-fold cv of training ML Model
cv_results_acc*100,'Mean=',cv_results_acc.mean()*100
```

Figure 5.17: Implementation of 10-fold Cross Validation for Model Evaluation by Accuracy

```
from sklearn.model_selection import KFold,cross_val_predict
from sklearn.metrics import confusion_matrix

kf = KFold(n_splits=10, shuffle=True, random_state=1)

#This is used to Predict Test Fold
y_pred=cross_val_predict(model,X_smote,Y_smote, cv=kf,n_jobs=-1)

#This is used to Display Confusion Matrix
conf_matrix= confusion_matrix(Y_smote,y_pred)
conf_matrix
```

```
#This is used to Display Classification Report
classif_report=classification_report(Y_smote,y_pred)
classif_report
```

Figure 5.18: Code of 10-fold CV Prediction, Confusion Matrix and Classification Report of a Models

CHAPTER SIX

6. RESULT AND DISCUSSION

In this chapter, we discuss the results obtained through experimenting the proposed system of software defect prediction supported by different evaluation methods. The chapter includes three parts. Firstly, elaborate results of data preprocessing. Secondly, elaborate performance and evaluation results of selected ensembles machine learning algorithms for model construction with effect of feature selection and data balancing by performance metrics on each five AEEEM datasets. Lastly, we compare the results of seven selected ensemble learning with combination of three feature selection and data balancing on five AEEEM software defect datasets and the best model is provided for deployment to classify or predict software modules into defect or non-defect. A detailed comparison of this research with the previous studies is also provided to assure that this study fills the gaps identified in the literature review, answers the research question, and achieves stated research specific objectives.

6.1. Data Preprocessing Results

The Datasets of AEEEM projects used in this research has totally 62 attributes. Among them one is dependent attribute and 61 independent software metrics attributes. Based on selected datasets, data preprocessing results are explained so as to recognize, understand and solve problems through different scales on features magnitude, class imbalance and high feature dimensionality.

6.1.1. Data Normalization Results

Data normalization technique applied on this research is z-score standardization that is used to transform features scales magnitude into similar order magnitude to obtain equal effect of features value on machine learning training thereby reducing outlier. Based on z-score normalization, the result of data normalization is represented in ranges of -3.00 up to 3.00 for all selected datasets features as it is indicated in the following figure.

ck_oo_numberOfPrivateMethods	LDHH_lcom	LDHH_fanIn	numberOfNonTrivialBugsFoundUntil:	WCHU_numberOfPublicAttributes	WCHU_numberOfAttributes	CvsWEntropy	L
0.005418	-0.405219	0.230487	-0.180406	-0.272427	0.456227	0.326057	
-0.384673	-0.167946	-0.453933	0.068324	-0.272427	-0.421286	-0.384903	
-0.384673	0.081728	2.413224	1.436342	1.227138	0.030503	0.316167	
-0.384673	-0.405219	1.705017	-0.056041	-0.272427	-0.421286	-0.422175	
-0.384673	1.245306	1.123314	0.068324	-0.272427	0.456227	0.648040	

LDHH_numberOfPublicMethods	WCHU_fanIn	LDHH_numberOfPrivateAttributes	...	numberOfBugsFoundUntil:	LDHH_fanOut	LDHH_numberOfMethodsInherited	LDHH_rfc	ck_oo_numberOfMethodsInherited
-0.377861	1.173946	0.339926	...	-0.201584	-0.200511	0.471580	-0.361954	-0.584948
0.169497	-0.512222	-0.343828	...	0.030434	-0.152250	0.666157	-0.097476	0.553101
0.478167	1.750793	0.379304	...	1.306535	2.185492	0.358906	1.205005	-0.205598
-0.377861	0.047985	-0.343828	...	-0.085575	-0.505457	-0.699054	-0.467824	0.325491
2.298322	2.915580	-0.028288	...	0.146444	1.834053	-0.026629	0.432402	-0.584948

ck_oo_numberOfPublicMethods	LDHH_cbo	WCHU_numberOfLinesOfCode	CvsExpEntropy	LDHH_numberOfMethods
0.280492	0.098782	0.164187	-0.649814	-0.413335
-0.283976	-0.439643	0.073806	-0.593161	-0.154212
0.139375	1.837935	1.056903	1.330638	0.138959
0.986077	-0.582529	-0.477997	1.909030	-0.413335
1.691662	1.634339	1.064831	0.585979	1.058544

Figure 6.1: Result of data normalization technique

The figure shown above represents the data normalization result of the datasets as data features become equal effects for all activities perform after this section.

6.1.2. Feature Selection Results

Feature selection techniques are employed in this research to choose best and appropriate attribute set for enhancing prediction power of classifier algorithms and minimize overfitting. In this study, three feature selection techniques such as Correlation Based Feature Selection (CFS), Correlation-Filter(CO) and Sequential Forward Selection(SFS) have been used. FS and CO are included under filtering methods which is independent from machine learning while SFS is included under wrapper methods which is based on machine learning classifier. For that reason, SFS is used with all selected ensemble learning algorithms also with selected base classifier in the study. For CO feature selection technique, Brute Force method with threshold point of **0.8** is used to extract unneeded features and to obtain best features (i.e. identify best

features by rank score of each features). Therefore, from all feature selection techniques, the output result of best selected features is displayed as in the following table.

Table 6.1: Feature Selection result for every selected ensemble learning algorithms in model building

Datasets	***Number of Selected Attributes by different Feature Selection Technique***								
	CFS	***CO***	***SFS***						
			RF	***ET***	***GB***	***XGBoost***	***BET***	***AET***	***SDF***
EQ	17	26	11	7	47	7	16	7	9
JDT	12	27	31	5	50	12	23	26	6
LC	7	34	9	15	19	9	10	6	23
ML	16	34	34	26	47	18	14	38	15
PDE	15	28	11	20	30	16	13	28	15
BET: Bagging with Extra tree base learner, AET: Adaptive boosting with Extra tree base learner, SDF: Stacking by default final meta learner with base weak learner such as ET, RF and DT.									

Table 6.1 shown above represents the number of attributes selected by different feature selection techniques from 61 original features for all selected ensemble learning algorithms as well as some feature selection techniques like SFS depend on selected ensemble learning with their properties. The effect of feature selection result is given to data balancing technique as input and then the last preprocessed data is given to selected ensemble learning for training purpose. The next section discusses about data balancing by taking output result of feature selection as input to produce the last data for training.

6.1.3. Data Balancing Results

The datasets used in this study is impacted by data imbalance problem. And this problem causes majority classes to dominate the minority class in machine leaning training. In other words, it obtains biased performance result which is dominated by single class. Dataset balancing technique is the method used to equalize the classes occurrence of the dataset. It balances classes ratio in the dataset that occurred in different quantities. For this study, we used SMOTE data balancing technique that performs oversampling approach to balance the classes by increasing minor class (defect). Therefore, SMOTE data balancing is applied for all AEEEM datasets such

as EQ, JDT, LC, ML and PDE as it is represented in the following figures that shows imbalanced data and balanced data for all datasets.

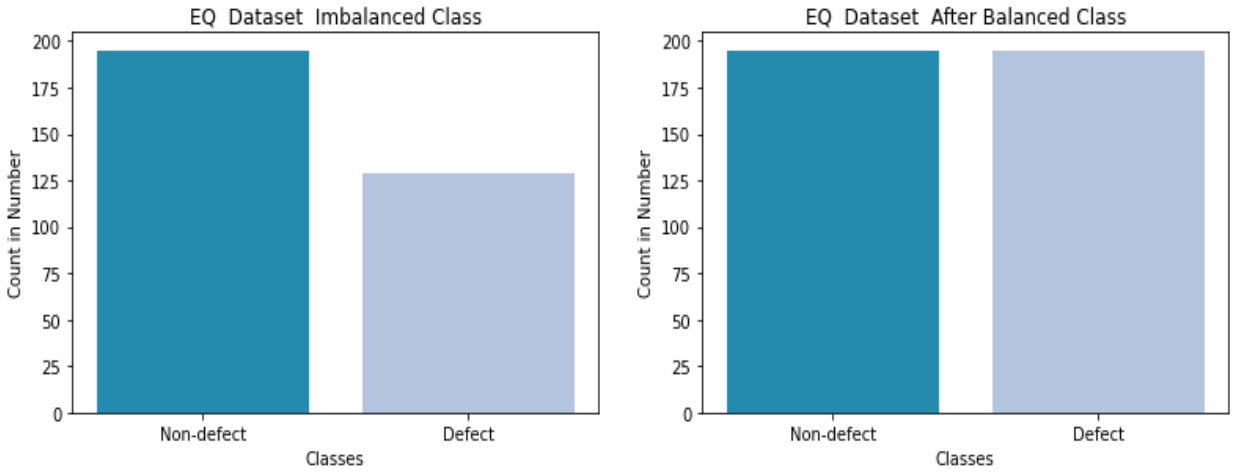

Figure 6.2: EQ dataset before balanced(at left side) and after balanced class(at right side)

Figure 6.2 shown above indicates that EQ dataset before balanced has held 324 software modules, from which 195 modules are non-defective and 129 modules are defective. After SMOTE data balancing technique is performed, the EQ dataset have changed to total number of 390 software modules, from which 195 modules are non-defective and 195 modules are defective.

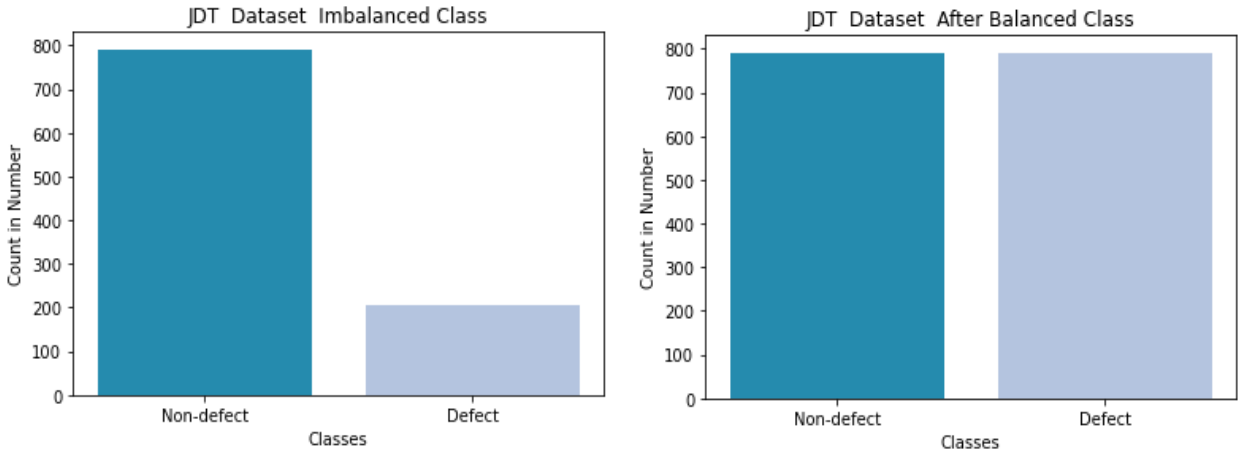

Figure 6.3: JDT dataset before balanced(at left side) and after balanced class(at right side)

Figure 6.3 shown above indicates that JDT dataset before balanced has held 997 software modules, from which 791 modules are non-defective and 206 modules are defective. After SMOTE data balancing technique is performed, the JDT dataset have changed to total number

of 1,582 software modules, from which 791 modules are non-defective and 791 modules are defective.

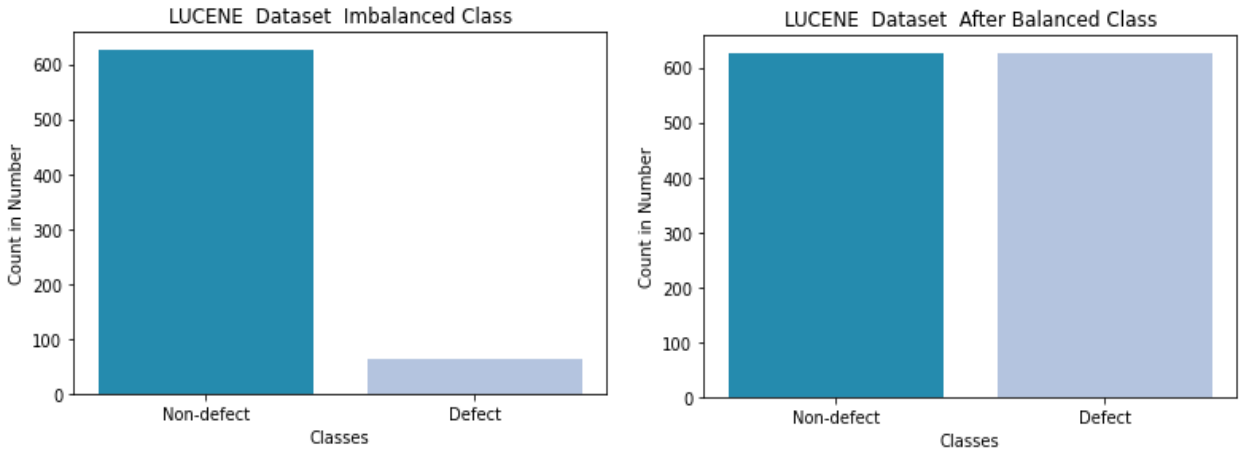

Figure 6.4: LC(Lucene) dataset before balanced(at left side) and after balanced class(at right side)

Figure 6.4 shown above indicates that LC(Lucene) dataset before balanced has held 691 software modules, from which 627 modules are non-defective and 64 modules are defective. After SMOTE data balancing technique is performed, the LC(Lucene) dataset have changed to total number of 1,254 software modules, from which 627 modules are non-defective and 627 modules are defective.

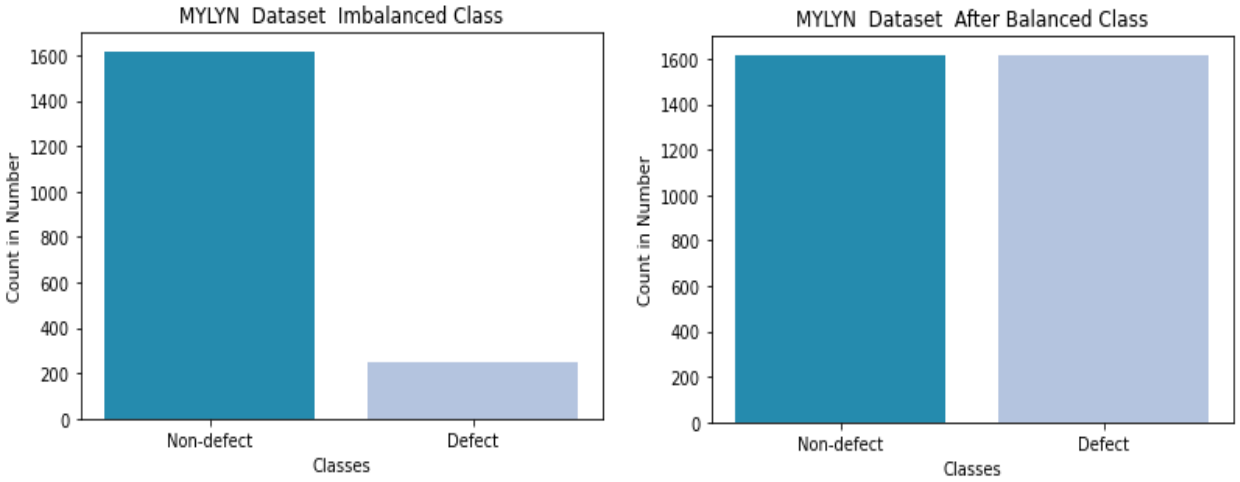

Figure 6.5: ML(Mylyn) dataset before balanced(at left side) and after balanced class(at right side)

Figure 6.5 shown above indicates that ML(Mylyn) dataset before balanced has held 1862 software modules, from which 1617 modules are non-defective and 245 modules are defective. After SMOTE data balancing technique is performed, the ML(Mylyn) dataset have changed to total number of 3,234 software modules, from which 1617 modules are non-defective and 1617 modules are defective.

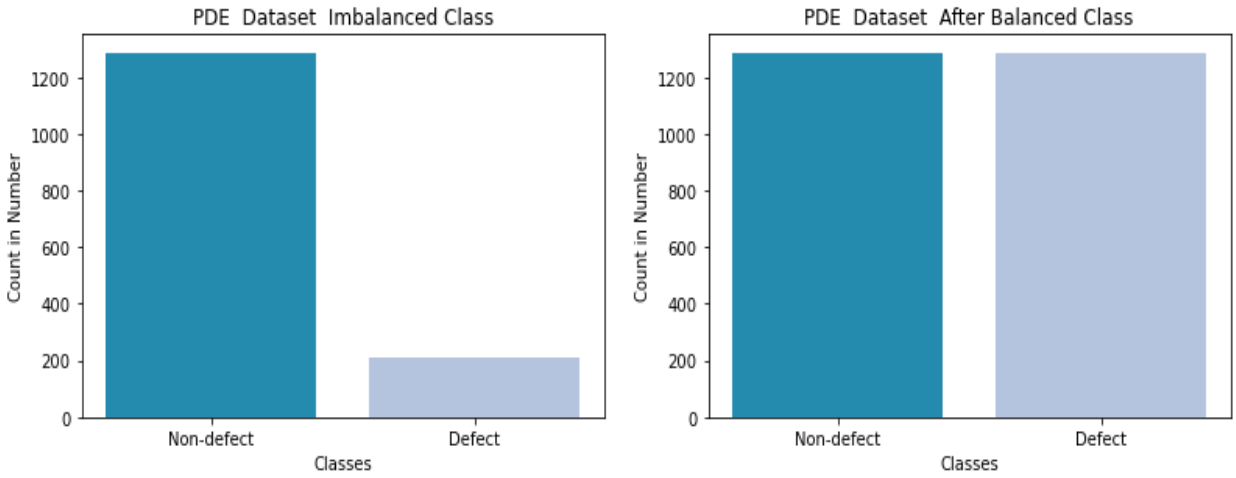

Figure 6.6: PDE dataset before balanced(at left side) and after balanced class(at right side)

Figure 6.6 shown above indicates that PDE dataset before balanced has held 1497 software modules, from which 1288 modules are non-defective and 209 modules are defective. After SMOTE data balancing technique is performed, the PDE dataset have changed to total number of 2,576 software modules, from which 1288 modules are non-defective and 1288 modules are defective.

The above mentioned data balancing results are provided the last suitable datasets for model building by selected ensemble machine learning algorithms and this technique's results are considered as last preprocessing. After this stage the dataset produced by this technique is used for training purpose by the selected ensemble learning algorithms such as AdaBoost, GB, XGBoost, RF, ET, Bagging and Stacking algorithms with different base learners. The performances of these selected ensemble learning techniques are discussed as the following sections.

6.2. Models Performance and Evaluation Results

In this study, when the first data preprocessing stage is completed, the appropriate data is provided for the next data preprocessing stages such as feature selection and data balancing technique. Then after the final best and suitable dataset is provided for model building training. Based on above procedures, the final provided datasets for training purpose is given for selected ensemble machine learning algorithms such as AdaBoost, GB, XGBoost, RF, ET, Bagging and Stacking algorithms with different base learners to construct software defect prediction models. In this study, each feature selection techniques such as CFS, CO and SFS used in the proposed system are firstly combined with SMOTE data balancing technique thereby combining the results of both techniques with each selected ensemble learning. Consequently, their performance is evaluated by performance metrics such as confusion matrix, accuracy, recall, precision, F-measure, AUC (Area Under ROC curve) with 10-fold cross validation for all used datasets. And then their performance results are compared each other and the best model is selected for deployment of software defect prediction. In the next section, the result of each ensemble learning algorithms is discussed with selected feature selection techniques.

6.2.1. Result of Ensemble Learning with feature selection

Under here, the results of each ensemble learning are discussed with selected feature selection techniques by accuracy metrics evaluation to identify effective and best performance feature selection technique for all datasets.

The experimental results of selected ensemble learning algorithms such as AdaBoost, GB, XGBoost, RF, ET, Bagging and Stacking algorithms with selected feature selection techniques such as CFS, CO and SFS are depicted as in the following tables.

Table 6.2: Accuracy result of Random Forest(RF) with different Feature Selection

Datasets	***CFS***	***CO***	***SFS by RF***
EQ	0.903	0.915	**0.944**
JDT	0.927	**0.961**	**0.961**
LC	0.918	**0.98**	**0.98**
ML	0.927	**0.976**	0.975
PDE	0.949	**0.971**	0.963

Table 6.2 shown above represents the accuracy result of Random Forest(RF) classifier with different feature selection such as CFS, CO and SFS. The result is represented for all datasets and the highest value is indicated by bold for each dataset. The performed experiment shown indicates that RF classifier is integrated with each feature selection techniques for all datasets. Based on the results, CO feature selection technique has given better performance on all of the datasets except on EQ. SFS feature selection technique has also shown good performance on three datasets such as EQ, JDT and LC.

Table 6.3: Accuracy result of Extra Tree(ET) with different Feature Selection

Datasets	***CFS***	***CO***	***SFS by ET***
EQ	0.89	0.931	**0.936**
JDT	0.937	**0.963**	**0.963**
LC	0.927	**0.992**	0.988
ML	0.93	**0.982**	**0.982**
PDE	0.952	**0.978**	0.974

Table 6.3 shown above represents the accuracy result of Extra Tree(ET) classifier with different feature selection such as CFS, CO and SFS. The result is represented for all datasets and the highest value is indicated by bold for each dataset. The performed experiment shown indicates that ET classifier is integrated with each feature selection techniques for all datasets. Based on the results, CO feature selection technique has shown better performance on all of the datasets except on EQ. Also SFS feature selection technique has also shown good performance on three datasets such as EQ, JDT and ML.

Table 6.4: Accuracy result of Gradient Boosting(GB) with different Feature Selection

Datasets	***CFS***	***CO***	***SFS by GB***
EQ	0.897	0.91	**0.923**
JDT	0.894	0.936	**0.951**
LC	0.887	**0.975**	0.961
ML	0.867	**0.935**	0.932
PDE	0.847	0.926	**0.932**

Table 6.4 shown above represents the accuracy result of Gradient Boosting(GB) classifier with different feature selection such as CFS, CO and SFS. The result is represented for all datasets and the highest value is indicated by bold for each dataset. The performed experiment shown indicates that GB classifier is integrated with each feature selection techniques for all datasets. Based on the results, SFS feature selection technique is obtained has shown better performance on three datasets such as EQ, JDT and PDE. CO feature selection technique has also obtained good performance on only two datasets namely, LC and ML.

Table 6.5: Accuracy result of Extreme Gradient Boosting(XGBoost) with different Feature Selection

Datasets	***CFS***	***CO***	***SFS by XGBoost***
EQ	0.905	0.908	**0.928**
JDT	0.924	**0.965**	0.963
LC	0.91	**0.982**	0.978
ML	0.921	**0.973**	0.972
PDE	0.929	**0.97**	**0.97**

Table 6.5 shown above represents the accuracy result of Extreme Gradient Boosting(XGBoost) classifier with different feature selection methods such as CFS, CO and SFS. The result is represented for all datasets and the highest value is indicated by bold for each dataset. The performed experiment shown indicates tthat XGBoost classifier is integrated with each feature selection techniques for all datasets. Based on the results, CO feature selection technique has obtained better performance on all of the datasets except on EQ. SFS feature selection technique has also given good performance on only two datasets such as EQ and PDE.

The above mentioned ensemble learning algorithms such as RF, ET, GB and XGBoost are used only decision tree(DT) algorithm in their base classifier. they are decision tree(DT) based ensemble learning algorithms. However, the ensemble machine learning algorithms used in this study is not only the mentioned above. Therefore, additionally others ensemble learning algorithms such as Bagging, AdaBoost and Stacking are used. And they all uses different base learning algorithms such as SVM, NB, DT and KNN. When the results of all base learning techniques mentioned above for integrating on base estimator of those ensemble learning are compared, the DT algorithm has shown better performance than others. Here, DT is used as a

base learner(estimator) for those ensembles learning as it can be seen from their experimental results which is depicted on **Appendix B**. However, DT algorithm is itself, is the only base learner of ensemble learning such as RF, ET, GB and XGBoost. Because of this ensemble learning schemes such as RF, ET, GB and XGBoost used only DT algorithm on their base learning. Hence, the experimental result of DT with RF, ET, GB and XGBoost has been compared including these ensemble learning with each other in which case ET has shown better in performance than DT, RF, ET, GB and XGBoost. ET has also been considered as base learning of Bagging and AdaBoost ensemble learning instead of DT default on base learner assuming that it is best and an improved algorithm. The experimental results of using ET as a base learning techniques is presented as an **Appendix B**. But for Stacking ensemble, only three base learners such as ET, RF and DT are used because of computational performance of stacking is very high with all these five algorithms (DT, RF, ET, GB and XGBoost) on base learners as stacking is heterogeneous. Besides, default algorithm for final meta estimator of stacking algorithm with those base learners is used. It is possible to refer the result of all base learning from **Appendix B** as mentioned before.

Tables 6.6, 6.7, 6.8 and 6.8 depict experimental results of ensemble learning such as Bagging, AdaBoost and Stacking with best and effective base learners combined with each selected feature selection techniques as we discussed earlier.

Table 6.6: Accuracy result of Bagging that uses ET on Base Learner with different Feature Selection

Datasets	***CFS***	***CO***	***SFS by Bagging of ET base learner***
EQ	0.887	0.913	**0.928**
JDT	0.934	**0.96**	0.958
LC	0.919	**0.99**	0.977
ML	0.926	**0.981**	0.975
PDE	0.946	**0.978**	0.969

Table 6.6 shown above represents the accuracy result of Bagging that uses ET classifier on base learner with different feature selection such as CFS, CO and SFS. The result is represented for all datasets and the highest value is indicated by bold for each dataset. The performed

experiment shows that Bagging by ET on base learner is integrated with each feature selection techniques for all datasets. Based on the results, CO feature selection technique has demonstrated better performance on all of the datasets except on EQ. On the other hand, SFS feature selection technique has proven good performance on only EQ dataset.

Table 6.7: Accuracy result of AdaBoost that uses ET on Base Learner with different Feature Selection

Datasets	***CFS***	***CO***	***SFS by AdaBoost of ET base learner***
EQ	0.882	0.918	**0.954**
JDT	0.93	0.965	**0.967**
LC	0.923	**0.991**	0.97
ML	0.931	**0.979**	0.973
PDE	0.953	0.981	**0.983**

Table 6.7 shown above represents the accuracy result of AdaBoost that uses ET classifier on base learner with different feature selection mechanisms such as CFS, CO and SFS. The result is represented for all datasets and the highest value is indicated by bold for each dataset. The performed experiment shows that AdaBoost by ET on base learner is integrated with each feature selection techniques for all datasets. Based on the results, SFS feature selection technique has demonstrated better performance on three datasets such as EQ, JDT and PDE. On the other hand, CO feature selection technique has confirmed good performance on only LC and ML datasets.

Table 6.8: Accuracy result of Stacking that uses ET, RF and DT on Base Learners, and uses default Final Estimator with different Feature Selection

Datasets	***CFS***	***CO***	***SFS by Stacking of ET, RF and DT on base learner and default final estimator***
EQ	0.903	0.918	**0.933**
JDT	0.933	**0.962**	0.958
LC	0.923	**0.989**	0.988
ML	0.93	**0.984**	0.977
PDE	0.953	**0.981**	0.966

Table 6.8 shown above represents the accuracy result of Stacking that uses ET, RF and DT classifiers on base learners, and uses default final estimator with different feature selection such as CFS, CO and SFS. The result is represented for all datasets and the highest value is indicated as bold for each dataset. The performed experiment shows that Stacking by ET, RF and DT on base learner as well as on default final estimator is integrated with each feature selection techniques for all datasets. Based on the results, CO feature selection technique assured better performance on all of the datasets except on EQ. SFS feature selection technique has also demonstrated good performance on only EQ dataset.

When the performance results of three feature selection techniques combined with seven ensemble machine learning algorithms on five datasets displayed on Tables 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 and 6.8 above are compared each other, the CO feature selection technique confirmed better performance than SFS and CFS on most datasets. SFS feature selection technique has also shown good performance on some dataset. However, experiments on CFS feature selection technique indicated lower performance on all datasets.

Analysis based on CO feature selection technique, is also compared on seven ensemble machine learning algorithms to select best performance ensemble learning for software defect prediction models. CO feature selection, analysis of the performance results of those seven ensemble learning algorithms with performance evaluation metrics such as accuracy, recall, precision, F-measure and AUC for all datasets are displayed as in the following table.

Table 6.9: Summary Results of All Ensembles Learning Algorithms

Datasets	***Performance Metrics***	***RF***	***ET***	***GB***	***XGBoost***	***BET***	***AET***	***SDF***
EQ	Accuracy	0.915	**0.931**	0.91	0.908	0.913	0.918	0.918
	AUC	0.979	**0.985**	0.951	0.969	0.98	0.983	0.982
	F1-Measure	0.915	**0.933**	0.909	0.906	0.916	0.92	0.92
	Precision	0.91	**0.916**	0.9	0.902	0.894	0.898	0.902
	Recall	0.924	**0.953**	0.921	0.918	0.943	0.947	0.944
JDT	Accuracy	0.961	**0.963**	0.936	**0.965**	0.96	**0.965**	0.962
	AUC	0.993	**0.996**	0.98	0.99	0.994	**0.996**	0.995
	F1-Measure	0.961	**0.964**	0.935	**0.965**	0.961	**0.965**	0.962

	Precision	0.962	0.955	0.948	**0.966**	0.952	0.957	0.961
	Recall	0.96	**0.974**	0.923	0.964	0.97	**0.974**	0.964
LC	Accuracy	0.98	**0.992**	0.975	0.982	0.99	0.991	0.989
	AUC	0.998	**1.0**	0.995	0.997	0.999	**1.0**	0.999
	F1-Measure	0.98	**0.992**	0.975	0.982	0.989	0.991	0.989
	Precision	0.981	**0.989**	0.97	0.98	**0.989**	0.987	**0.99**
	Recall	0.979	**0.995**	0.981	0.984	0.99	**0.995**	0.987
ML	Accuracy	0.976	**0.982**	0.935	0.973	0.981	0.979	**0.984**
	AUC	**0.996**	**0.996**	0.982	0.994	**0.997**	0.996	**0.997**
	F1-Measure	0.976	**0.982**	0.935	0.973	0.981	0.979	**0.984**
	Precision	**0.973**	**0.973**	0.925	0.973	0.97	**0.975**	**0.978**
	Recall	0.979	**0.991**	0.946	0.973	**0.991**	0.984	0.99
PDE	Accuracy	0.971	**0.978**	0.926	0.97	**0.978**	**0.981**	**0.981**
	AUC	0.996	**0.999**	0.974	0.993	0.998	**0.999**	**0.999**
	F1-Measure	0.971	**0.978**	0.925	0.969	**0.978**	**0.981**	**0.981**
	Precision	0.97	0.968	0.933	0.974	0.967	0.972	0.976
	Recall	0.973	**0.988**	0.919	0.965	**0.99**	**0.99**	0.985

Table 6.9 shown above represents the summary results of all ensemble machine learning algorithms used in this study with five performance evaluation metrics on each five datasets. The results are depicted for all datasets with respect to all ensemble learning used in the study by five performance evaluation metrics and the significant results are indicated as bold text for all dataset with respect to ensemble learning algorithms. when we compare the experimental results of seven ensemble machine learning algorithms such as AdaBoost, GB, XGBoost, RF, ET, Bagging and Stacking with five performance evaluation metrics on five datasets as depicted on table 6.9, the ET ensemble algorithm has shown better performance on most datasets and most performance evaluation metrics. AdaBoost and Stacking ensemble algorithms has also shown the second and the third level performance results respectively next to the ET algorithm. However, GB ensemble algorithm has proven lowest performance on all datasets with all performance evaluation metrics. Therefore, ET ensemble algorithm is selected for software defect prediction model building with CO feature Selection Technique for all datasets and the

selected features of CO feature selection technique is depicted on **Appendix A**. Hence, the next section is discussion about best performed algorithm namely ET by different performance evaluation metrics.

6.2.2. Results of best performance ensemble learning

By using performance results of ensemble learning indicated on Table 6.9, when we compare the experimental results of seven ensemble machine learning algorithms such as AdaBoost, GB, XGBoost, RF, ET, Bagging and Stacking with five performance evaluation metrics on five datasets, the ET ensemble algorithm is obtained better performance on most datasets and most performance evaluation metrics.

Confusion Matrix of best performance ensemble learning without Normalize and with Normalize:

Confusion matrix is a representation of matrix that used to calculate performance evaluation of classification algorithms from it. The following figures represents confusion matrix of all datasets.

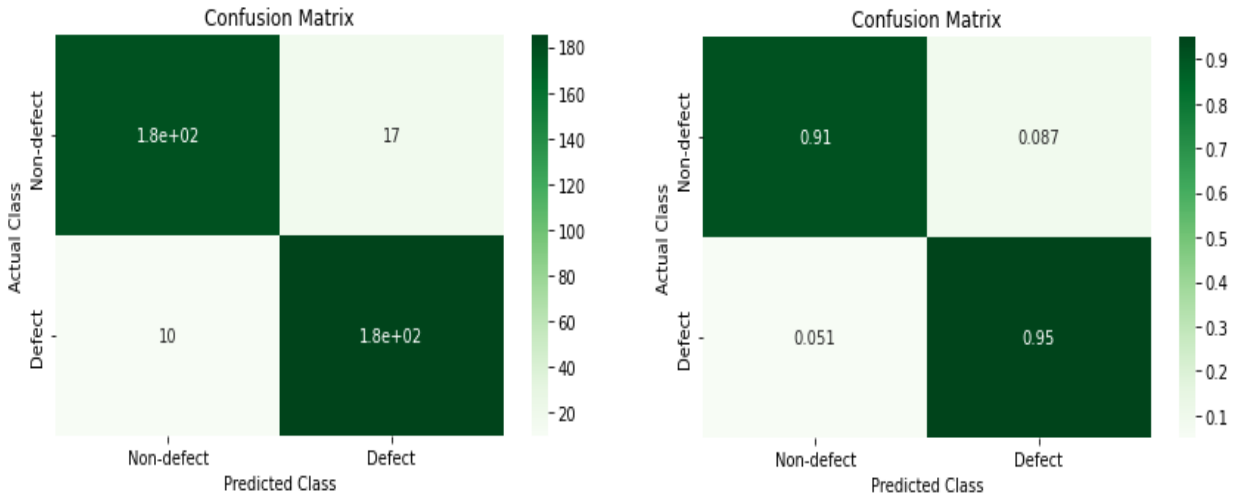

*Figure 6.7: Confusion Matrix of ET Ensemble Learning Algorithm for **EQ dataset** without normalize (at left side) and with normalize(at right side)*

Figure 6.7 represents confusion matrix of ET algorithm for **EQ dataset** that trained by 10-folds cross validation. As shown in the figure all classes of a dataset are trained in good manner that means the diagonal instances are higher than others. In diagonal TN (True Negative is both actual and predicted class is Non-defect) and TP (True Positive is both actual and predicted class

is Defect). The confusion matrix with normalize of ET algorithm classifies TN(Non-defect) class 91% and TP(Defect) class 95%.

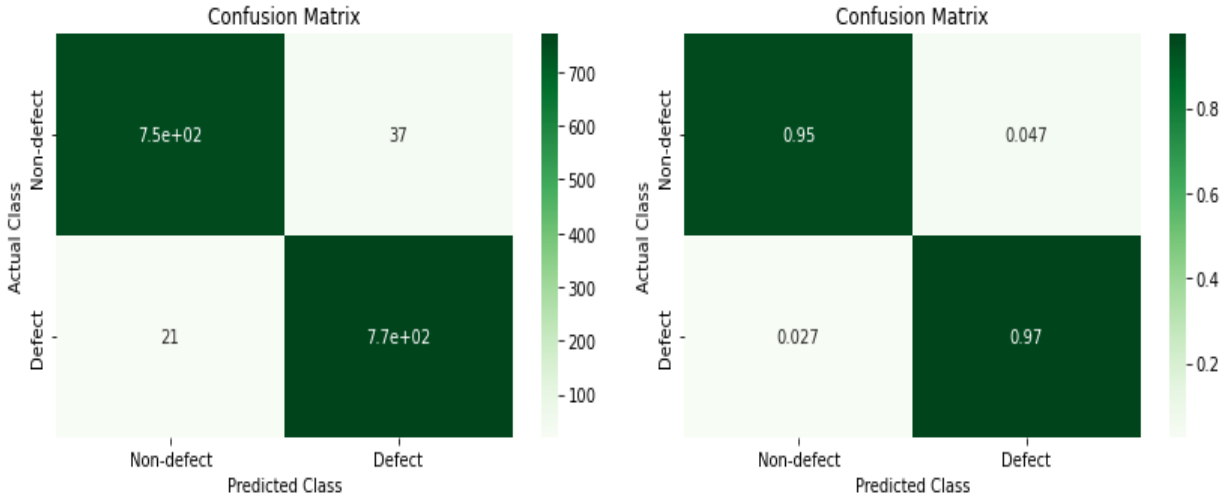

*Figure 6.8: Confusion Matrix of ET Ensemble Learning Algorithm for **JDT dataset** without normalize (at left side) and with normalize(at right side)*

Figure 6.8 represents confusion matrix of ET algorithm for **JDT dataset** that trained by 10-folds cross validation. As shown in the figure all classes of a dataset are trained in good manner that means the diagonal instances are higher than others. In diagonal TN (True Negative is both actual and predicted class is Non-defect) and TP (True Positive is both actual and predicted class is Defect). The confusion matrix with normalize of ET algorithm classifies TN(Non-defect) class 95% and TP(Defect) class 97%.

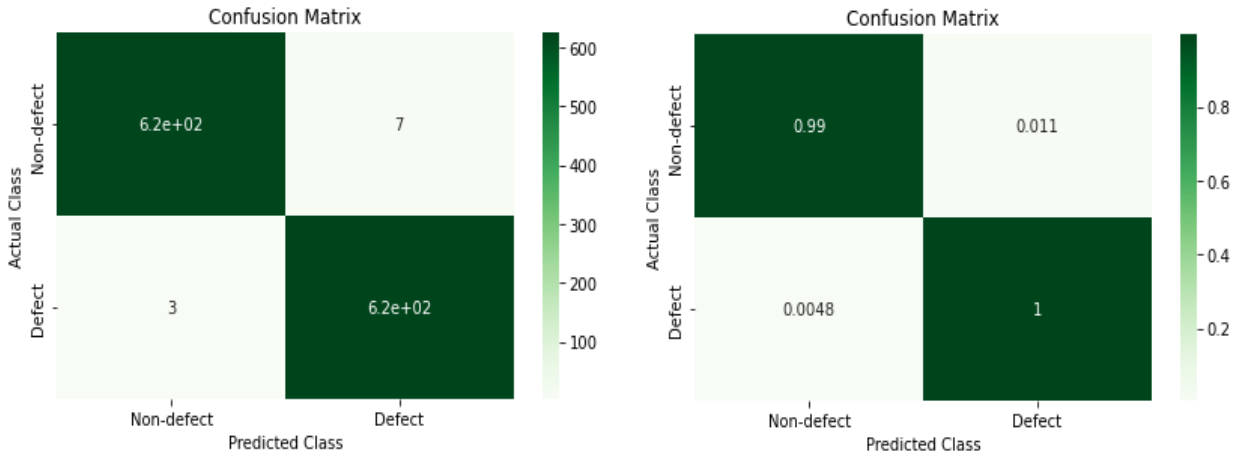

*Figure 6.9: Confusion Matrix of ET Ensemble Learning Algorithm for **LC(Lucene) dataset** without normalize (at left side) and with normalize(at right side)*

Figure 6.9 represents confusion matrix of ET algorithm for **LC(Lucene) dataset** that trained by 10-folds cross validation. As shown in the figure all classes of a dataset are trained in good manner that means the diagonal instances are higher than others. In diagonal TN (True Negative is both actual and predicted class is Non-defect) and TP (True Positive is both actual and predicted class is Defect). The confusion matrix with normalize of ET algorithm classifies TN(Non-defect) class 99% and TP(Defect) class 100%.

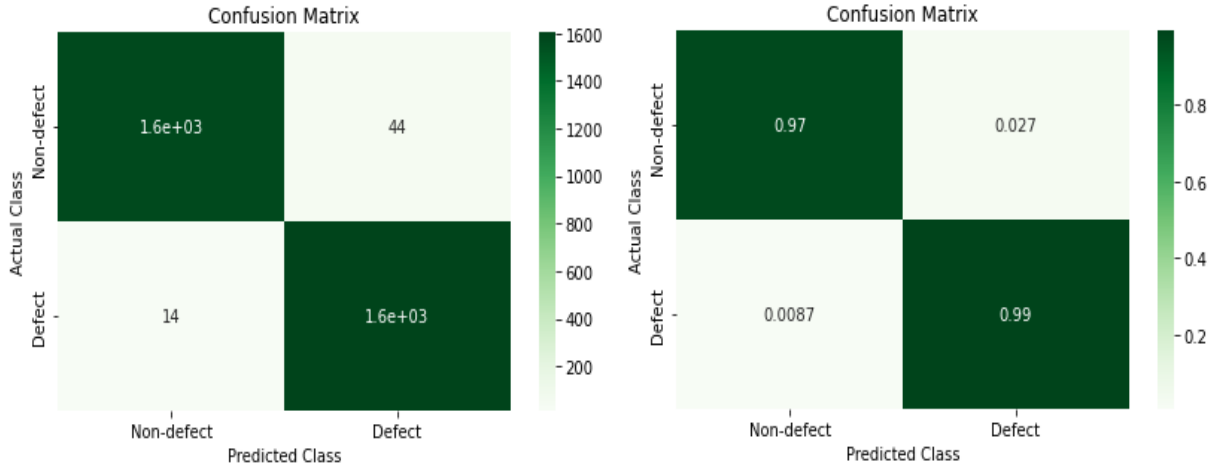

*Figure 6.10: Confusion Matrix of ET Ensemble Learning Algorithm for **ML(Mylyn) dataset** without normalize (at left side) and with normalize(at right side)*

Figure 6.10 represents confusion matrix of ET algorithm for **ML(Mylyn) dataset** that trained by 10-folds cross validation. As shown in the figure all classes of a dataset are trained in good manner that means the diagonal instances are higher than others. In diagonal TN (True Negative is both actual and predicted class is Non-defect) and TP (True Positive is both actual and predicted class is Defect). The confusion matrix with normalize of ET algorithm classifies TN(Non-defect) class 97% and TP(Defect) class 99%.

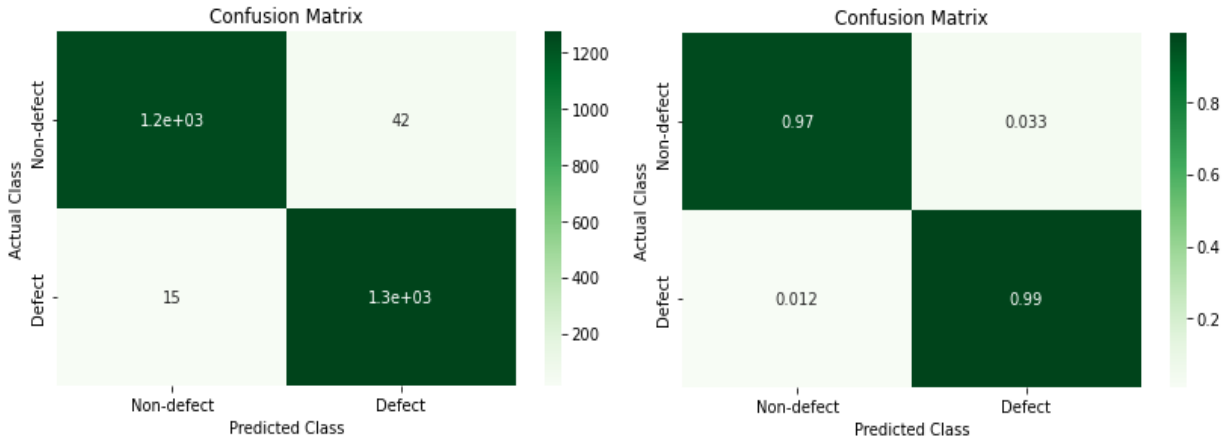

*Figure 6.11: Confusion Matrix of ET Ensemble Learning Algorithm for **PDE dataset** without normalize (at left side) and with normalize(at right side)*

Figure 6.11 represents confusion matrix of ET algorithm for **PDE dataset** that trained by 10-folds cross validation. As shown in the figure all classes of a dataset are trained in good manner that means the diagonal instances are higher than others. In diagonal TN (True Negative is both actual and predicted class is Non-defect) and TP (True Positive is both actual and predicted class is Defect). The confusion matrix with normalize of ET algorithm classifies TN(Non-defect) class 97% and TP(Defect) class 99%.

Classification Report of best performance ensemble learning:

Classification report is used to elaborate performance evaluation metrics such accuracy, precision, recall and f1-score (F-measure)

	precision	recall	f1-score	support
0	0.95	0.91	0.93	195
1	0.92	0.95	0.93	195
accuracy			0.93	390
macro avg	0.93	0.93	0.93	390
weighted avg	0.93	0.93	0.93	390

*Figure 6.12: Classification Report of ET Ensemble Learning Algorithm for **EQ dataset***

Figure 6.12 represents classification report of ET algorithm for **EQ dataset**. As shown in the figure ET algorithm produces, the precision of Non-defect (0) class is 95%, Defect (1) class is 92% and both classes in average 93%. The recall of Non-defect (0) class is 91%, Defect (1) class is 95% and both classes in average 93%. The F-Measure(f1-score) of Non-defect (0) class

is 93%, Defect (1) class is 93% and both classes in average 93%. The accuracy of the dataset is 93%. And also the EQ dataset of AUC metrics is 98.5%.

	precision	recall	f1-score	support
0	0.97	0.95	0.96	791
1	0.95	0.97	0.96	791
accuracy			0.96	1582
macro avg	0.96	0.96	0.96	1582
weighted avg	0.96	0.96	0.96	1582

*Figure 6.13: Classification Report of ET Ensemble Learning Algorithm for **JDT dataset***

Figure 6.13 represents classification report of ET algorithm for **JDT dataset**. As shown in the figure ET algorithm produces, the precision of Non-defect (0) class is 97%, Defect (1) class is 95% and both classes in average 96%. The recall of Non-defect (0) class is 95%, Defect (1) class is 97% and both classes in average 96%. The F-Measure(f1-score) of Non-defect (0) class is 96%, Defect (1) class is 96% and both classes in average 96%. The accuracy of the dataset is 96%. And also the JDT dataset of AUC metrics is 99.6%.

	precision	recall	f1-score	support
0	1.00	0.99	0.99	627
1	0.99	1.00	0.99	627
accuracy			0.99	1254
macro avg	0.99	0.99	0.99	1254
weighted avg	0.99	0.99	0.99	1254

*Figure 6.14: Classification Report of ET Ensemble Learning Algorithm for **LC(Lucene) dataset***

Figure 6.14 represents classification report of ET algorithm for **LC(Lucene) dataset**. As shown in the figure ET algorithm produces, the precision of Non-defect (0) class is 100%, Defect (1) class is 99% and both classes in average 99%. The recall of Non-defect (0) class is 99%, Defect (1) class is 100% and both classes in average 99%. The F-Measure(f1-score) of Non-defect (0) class is 99%, Defect (1) class is 99% and both classes in average 99%. The accuracy of the dataset is 99%. And also the LC(Lucene) dataset of AUC metrics is 100%.

	precision	recall	f1-score	support
0	0.99	0.97	0.98	1617
1	0.97	0.99	0.98	1617
accuracy			0.98	3234
macro avg	0.98	0.98	0.98	3234
weighted avg	0.98	0.98	0.98	3234

*Figure 6.15: Classification Report of ET Ensemble Learning Algorithm for **ML(Mylyn) dataset***

Figure 6.15 represents classification report of ET algorithm for **ML(Mylyn) dataset**. As shown in the figure ET algorithm produces, the precision of Non-defect (0) class is 99%, Defect (1) class is 97% and both classes in average 98%. The recall of Non-defect (0) class is 97%, Defect (1) class is 99% and both classes in average 98%. The F-Measure(f1-score) of Non-defect (0) class is 98%, Defect (1) class is 98% and both classes in average 98%. The accuracy of the dataset is 98%. And also the ML(Mylyn) dataset of AUC metrics is 99.6%.

	precision	recall	f1-score	support
0	0.99	0.97	0.98	1288
1	0.97	0.99	0.98	1288
accuracy			0.98	2576
macro avg	0.98	0.98	0.98	2576
weighted avg	0.98	0.98	0.98	2576

*Figure 6.16: Classification Report of ET Ensemble Learning Algorithm for **PDE dataset***

Figure 6.16 represents classification report of ET algorithm for **PDE dataset**. As shown in the figure ET algorithm produces, the precision of Non-defect (0) class is 99%, Defect (1) class is 97% and both classes in average 98%. The recall of Non-defect (0) class is 97%, Defect (1) class is 99% and both classes in average 98%. The F-Measure(f1-score) of Non-defect (0) class is 98%, Defect (1) class is 98% and both classes in average 98%. The accuracy of the dataset is 98%. And also the PDE dataset of AUC metrics is 99.9%.

Data integration of all datasets by best performance ensemble learning:

Data integration is the method of combining variety of a data from multiple sources to form big unit data for smart decision. In this study, we integrated 5 AEEEM datasets into single dataset with respecting software metrics of all datasets. Then, five AEEEM datasets namely EQ, JDT, LC, ML and PDE are integrated together and form new AEEEM dataset. This new AEEEM

dataset has 61 software metrics(features) that similar with all five previous AEEEM datasets. The new AEEEM dataset has totally 5371 software modules(instances) from which 4518 modules are non-defective (84.1 %) and 853 modules are defective (15.9 %).

For this new AEEEM dataset, we carry out best CO feature selection technique, SMOTE data balancing technique and best ET ensembles learning algorithm. Therefore, by CO feature selection technique 36 attributes/features are selected from original 61 attributes/features and the new AEEEM dataset has imbalanced class so that by SMOTE data balancing technique 4518 defective and 4518 non-defective modules are created to equalize the dataset classes. After CO feature selection and SMOTE balancing technique are carried out on the new AEEEM dataset and then the last appropriate dataset is produced to training with ET ensemble learning and the ET model performance results of integrated data are represented in percentage for all performance metrics such as Accuracy, AUC, F1-Measure, Precision, Recall are respectively 97.9%, 99.7%, 97.9%, 97.1% and 98.7%.

6.3. Comparison

Several researchers have done studies on software defect prediction using different methods and for different datasets with different problems to enhance reliability, testing efficiency and quality by reducing software development and maintenance cost. For this study, based on reviews made, problems such as high feature dimensionality and class imbalance are explored which are currently common in software defect prediction datasets. This study considered a research that has been carried out by (A. O. Balogun et al., 2020; Zhou et al., 2019) on AEEEM software defect datasets where the result of the performance of machine learning algorithms implemented is observed as very low. Here, in this study, software defect prediction models on AEEEM software defect datasets are experimented as a result of which their performance has been improved. This research applied three feature selection techniques for high feature dimensionality problem and SMOTE data balancing technique for class imbalance distribution problem. Seven effective ensemble learning algorithms are applied as proposed in chapter 4. Here, comparison with previous research work which is discussed in related works section of chapter two on software defect prediction has been made. A paper conducted on software defect prediction using new deep forest (**DPDF**) (Zhou et al., 2019) has shown low accuracy as a result of two main problems such as high feature dimensionality and class imbalance problem. As the comparison is indicated below in table 6.10, the implementation of ensemble technique (i.e. **ET**

ensemble learning with **CO** feature selection and **SMOTE** balancing method), on common five AEEEM software defect datasets has shown very high performance than that of previous study. The results are depicted for all datasets with respect to both approaches by five performance evaluation metrics and the significant results are indicated as bold text for all dataset with respect to both approaches.

Table 6.10: Comparison between Software defect prediction using new deep forest(DPDF) (Zhou et al., 2019) *and Our Best Performance Ensemble Learning (ET Approach)*

Datasets	***Performance Metrics***	***ET(Our Approach)***	***DPDF***
EQ	Accuracy	**0.931**	0.780
	AUC	**0.985**	0.85
	F1-Measure	**0.933**	0.75
	Precision	**0.916**	0.70
	Recall	**0.953**	0.81
JDT	Accuracy	**0.963**	0.850
	AUC	**0.996**	0.86
	F1-Measure	**0.964**	0.56
	Precision	**0.955**	0.72
	Recall	**0.974**	0.46
LC	Accuracy	**0.992**	0.930
	AUC	**1.0**	0.82
	F1-Measure	**0.992**	0.37
	Precision	**0.989**	0.81
	Recall	**0.995**	0.24
ML	Accuracy	**0.982**	0.870
	AUC	**0.996**	0.82
	F1-Measure	**0.982**	0.26
	Precision	**0.973**	0.47
	Recall	**0.991**	0.18
PDE	Accuracy	**0.978**	0.870
	AUC	**0.999**	0.77
	F1-Measure	**0.978**	0.31
	Precision	**0.968**	0.59
	Recall	**0.988**	0.21

Additionally, the combination of **CO** feature selection and **SMOTE** balancing method with **ET** ensemble learning approach are obtained good performance than paper conducted on software defect prediction using ensemble learning algorithms (Mehta, 2021) by solving data integration problem of the paper. The paper (Mehta, 2021) are discussed the performance of software defect prediction model by solving high feature dimensionality and class imbalance problem using respective techniques with using ensemble learning algorithms on the models. However, the performance results of (Mehta, 2021) paper approach/model is experimented and discussed on each four NASA datasets individually. This means the approach/ the model is not considering variety characteristics decision, training only on specific small sized dataset software modules, not using more efficient information and not using more powerful information. In this study, by applying data integration methods on four NASA datasets (Mehta, 2021) used in their paper with our approach namely **CO** feature selection and **SMOTE** balancing method with **ET** ensemble learning algorithm, our approach obtained good performance, improved variety characteristics decision, trained on huge sized dataset software modules, increased more efficient information and increased more powerful information than (Mehta, 2021) previous approach. As the comparison is indicated below in table 6.11, the implementation of ensemble technique (i.e. **ET** ensemble learning with **CO** feature selection and **SMOTE** balancing method), on four NASA software defect dataset by data integration has shown high performance than (Mehta, 2021) approach by average calculation from all four datasets. The results are depicted for NASA datasets with respect to both approaches by five performance evaluation metrics and the significant results are indicated as bold text for all dataset with respect to both approaches.

Table 6.11: Comparison between Software defect prediction using ensemble learning algorithms (Mehta, 2021) and Our Best Performance Ensemble Learning (ET Approach) respectively by average and data integration.

Dataset	**Performance Metrics**	***ET our approach (Data Integrated)***	***(Mehta, 2021) approach by average***
	Accuracy	**0.975**	0.957
4 NASA	Precision	**0.961**	0.96
	Recall	**0.99**	0.97
	F1-Measure	**0.975**	0.96

CHAPTER SEVEN

7. CONCLUSION, RECOMMENDATION AND FUTURE WORKS

Under this chapter, the conclusion, recommendation and futures works of this study is included. It identifies the essential ideas and future research direction for other researchers who interested in this domain.

7.1. Conclusion

The purpose of this study is to predict or classify software modules into defect and non-defect using ensemble machine learning algorithms for the software systems that are in any stage of software development life cycle and represent in quantitative form. This means before actual software testing process is beginning, the defect in any stage of software development life cycle can be detected thereby improve software quality, reducing development and maintenance costs. This approach that performs software testing activity at the end of software development life cycle will be exposed to high maintenance cost and budget risk if the fault is detected at the end stage.

In this study, software metrics that are used as input to ensemble machine learning algorithms are collected from two or more version evolving system of five AEEEM projects of software defect datasets concerning on different things such as time, version changes, file updates and previous defect. Five AEEEM datasets namely EQ, JDT, LC, ML and PDE are used for this study and each five datasets includes 62 software metrics of which 61 attributes are independent and one attribute is dependent. Those software metrics are 17-metrics from source code metrics, 17-metrics from churn of source code metrics, 17-metrics from entropy of source code metrics, 5-metrics from previous defects metrics, 5-metrics from entropy of change code metrics and one-metric is a target class which may hold two classes namely defect and non-defect. Here, the defect class percentage of EQ, JDT, LC, ML and PDE are respectively 39.8%, 20.7%, 9.3%, 13.2% and 13.96%.

Preprocessing methods such as data cleaning methods, missing values controlling methods and data normalization methods are performed to get suitable datasets for ensemble machine learning algorithms. However, the datasets don’t have missing values problem. Therefore, only

z-score data normalization method is performed to transform features value into the same order and reduce outlier problem. Suitable datasets prepared are given for the next preprocessing methods such as feature selection techniques and class balancing techniques to solve two main problems. These are high feature dimensionality and class imbalance problem that have been explored in literature review. Hence, to solve high feature dimensionality problem and select necessary set of attributes from original sets, three feature selection techniques from filter and wrapper methods that namely CFS, CO, and SFS are used for this study. To solve class imbalance problem of AEEEM datasets SMOTE data balancing technique are also performed. The appropriate datasets prepared are given to ensemble learning algorithms such as RF, ET, GB, XGBoost, AdaBoost, Bagging and Stacking with base classifier for training and model building.

The experiment is performed and its results are evaluated on the basis of 10- folds cross validation using performance metrics such as accuracy, recall, precision, F-measure and AUC. The performance results indicate that CO feature selection with ET ensemble learning algorithm is outperforming as compared to other models on all five datasets. For all five AEEEM datasets such as EQ, JDT, LC, ML and PDE the accuracy results are 93.1 %, 96.3 %, 99.2%, 98.2% and 97.8% respectively. Besides, other performance metrics on all datasets are showed more than 91%.

As well as when all five datasets are integrated together 5371 software modules are created that included 4518 modules are non-defective (84.1 %) and 853 modules are defective (15.9 %), and for this integrated dataset by carrying out CO feature selection and SMOTE data balancing technique with ET ensemble learning algorithm all performance metrics such as Accuracy, AUC, F1-Measure, Precision, Recall are respectively 97.9%, 99.7%, 97.9%, 97.1% and 98.7%. Therefore, ET with CO model is recommended for software defect prediction model that classify software modules into defect or non-defect. The recommended model deployment is implemented for end users.

7.2. Recommendation and Future Works

The objective of this study is to predict or classify software modules into defect and non-defect using ensemble machine learning algorithms. It carried out timely detection of defect-prone software modules before the actual testing process starts to enhance software quality, reducing development and maintenance costs instead of software testing perform at the end of software

development life cycle which may have high maintenance cost and budget risk if the fault is happen at the end stage. Therefore, this study simplifies and helps the work of software developers, testers and software companies by properly utilizing the resources and budget of the works and increase user satisfaction by delivering quality software for the market. Thereby, the researcher recommends software companies to develop software defect prediction models to detect the fault in time before the fault happen the risk on the budget.

This study proposed a solution for software defect prediction problems by using combination of feature selection techniques, data balancing technique and ensemble learning algorithms to construct software defect prediction models on currently existing small instances of a datasets and the result of these combination techniques is very good and it is a promising performances over existing studies on software defect prediction. However, it is a better to develop software defect prediction models by using deep learning algorithms with including more instances of datasets but currently there is no sufficient defective datasets on many software systems. Here, we recommend software companies as they prepare more defect datasets of software systems to improve software quality and reliability. In the future, we extend our study on all publically available datasets with other class balancing and feature selection techniques to further improve software defect prediction performance, and also to include software defect prediction model concerning cross/heterogeneous software defect projects.

* * *

*This research was funded by **Adama Science and Technology University** with a **grant number ASTU/SM-R/230/21**.*

REFERENCES

Abaei, G., & Selamat, A. (2014). A survey on software fault detection based on different prediction approaches. 79–95. https://doi.org/10.1007/s40595-013-0008-z

Adrion, W. R., Branstad, M. A., & Cherniavsky, J. C. (1982). Validation, Verification, and Testing of Computer Software. ACM Computing Surveys (CSUR), 14(2), 159–192. https://doi.org/10.1145/356876.356879

Ali, J., Khan, R., Ahmad, N., & Maqsood, I. (2012). Random Forests and Decision Trees. September.

Aljamaan, H. (2020). Software Defect Prediction using Tree-Based Ensembles.

Alsawalqah, H., Faris, H., B, I. A., & Alnemer, L. (2017). Hybrid SMOTE-Ensemble Approach. 1. https://doi.org/10.1007/978-3-319-57141-6

Alsawalqah, H., Hijazi, N., Eshtay, M., Faris, H., Al Radaideh, A., Aljarah, I., & Alshamaileh, Y. (2020). Software defect prediction using heterogeneous ensemble classification based on segmented patterns. Applied Sciences (Switzerland), 10(5). https://doi.org/10.3390/app10051745

Ambros, M. D., & Lanza, M. (2010). An extensive comparison of bug prediction approaches An Extensive Comparison of Bug Prediction Approaches. May. https://doi.org/10.1109/MSR.2010.5463279

Andina, D. (2010). Feature Selection Using Sequential Forward Selection and classification applying Artificial Metaplasticity Neural Network . December. https://doi.org/10.1109/IECON.2010.5675075

Balogun, A., Lafenwa-balogun, F. B., Mojeed, H., & Adeyemo, V. E. (2020). SMOTE-Based Homogeneous Ensemble Methods for Software Defect Prediction. September. https://doi.org/10.1007/978-3-030-58817-5

Balogun, A. O., Basri, S., Mahamad, S., Abdulkadir, S. J., Almomani, M. A., Adeyemo, V. E., Al-Tashi, Q., Mojeed, H. A., Imam, A. A., & Bajeh, A. O. (2020). Impact of Feature Selection Methods on the Predictive Performance of Software Defect Prediction Models : An Extensive Empirical Study. MDPI, Ml. https://doi.org/doi: 10.3390/sym12071147

Bbeiman, L. E. O. (1996). Bagging Predictors. 140, 123–140.

Breiman, L. E. O. (2001). Random Forests. 5–32.

Catal, C., Sevim, U., & Diri, B. (2009). Software Fault Prediction of Unlabeled Program Modules. May 2014.

Chawla, N. V, Bowyer, K. W., & Hall, L. O. (2002). SMOTE : Synthetic Minority Over-sampling Technique. 16, 321–357.

Chen, T., & Guestrin, C. (2016). XGBoost : A Scalable Tree Boosting System. 785–794.

Cunningham, P., & Delany, S. J. (n.d.). Feature Selection Tutorial with Python Examples.

Dhiauddin, M., Suffian, M., & Ibrahim, S. (2012). A Prediction Model for System Testing Defects using Regression Analysis. 2, 55–68. https://doi.org/10.7321/jscse.v2.n7.

Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. 81, 649–660. https://doi.org/10.1016/j.jss.2007.07.040

Esteves, G., Figueiredo, E., Viggiato, M., Ziviani, N., & Veloso, A. (2020). Understanding machine learning software defect predictions. Automated Software Engineering, 2(i). https://doi.org/10.1007/s10515-020-00277-4

Fenton, N. E., & Neil, M. (2000). Software Metrics : Roadmap Software Metrics : Roadmap. September. https://doi.org/10.1145/336512.336588

Freund, Y., & Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting *. 139, 119–139.

Freund, Y., Schapire, R. E., & Hill, M. (1996). Experiments with a New Boosting Algorithm.

Friedman, J. H. (2014). Greedy Function Approximation: A Gradient Boosting Machine. 29(5), 1189–1232.

Funda, G., Wolfinger, R., & Tan, P. (2017). Stacked Ensemble Models for Improved Prediction Accuracy. 1–19.

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. October 2005, 3–42. https://doi.org/10.1007/s10994-006-6226-1

Guyon, I. (2003). An Introduction to Variable and Feature Selection. 3, 1157–1182.

Hall, M. A. (1999). Correlation-based Feature Selection for Machine Learning. April.

Hassan, A. E. (2009). Predicting Faults Using the Complexity of Code Changes. 78–88.

Huda, S., Liu, K., Liu, S., Abdelrazek, M., Ibrahim, A., Al-dossari, H., & Ahmad, S. (2018). An ensemble oversampling model for class imbalance problem in software defect prediction. 3536(c). https://doi.org/10.1109/ACCESS.2018.2817572

Jaiswal, A., & Malhotra, R. (2016). Software Reliability Prediction Using Machine Learning

Techniques. Proceedings of Fifth International Conference on Soft Computing for Problem Solving, 436, 141–163. https://doi.org/10.1007/978-981-10-0448-3

Jiang, Y., Cukic, B., & Ma, Y. (2008). Techniques for evaluating fault prediction models. August, 561–595. https://doi.org/10.1007/s10664-008-9079-3

Johnson, A. M., & Malek, M. (1988). Survey of software tools for evaluating reliability, availability, and serviceability. ACM Computing Surveys (CSUR), 20(4), 227–269. https://doi.org/10.1145/50020.50062

Jovi, A., Brki, K., & Bogunovi, N. (n.d.). A review of feature selection methods with applications.

Koprinska, I., Poon, J., Clark, J., & Chan, J. (2007). Learning to classify e-mail. 177(October 2005), 2167–2187. https://doi.org/10.1016/j.ins.2006.12.005

LEVENDAL, Y. (1990). Reliability Analysis of Large Software Systems : Defect Data Modeling. IEEE Trans. Softw. Eng., I(February).

Li, Zhan, & Reformat, M. (2007). A practical method for the software fault-prediction. 2007 IEEE International Conference on Information Reuse and Integration, IEEE IRI-2007, 659–666. https://doi.org/10.1109/IRI.2007.4296695

Li, Zhiqiang, Jing, X. Y., & Zhu, X. (2018). Progress on approaches to software defect prediction. In IET Software (Vol. 12, Issue 3, pp. 161–175). Institution of Engineering and Technology. https://doi.org/10.1049/iet-sen.2017.0148

Liu, H. (2002). A Comparative Study on Feature Selection and Classification Methods Using Gene Expression Profiles and Proteomic Patterns. June 2014. https://doi.org/10.11234/gi1990.13.51

Maggo, S., & Gupta, C. (2014). A Machine Learning based Efficient Software Reusability Prediction Model for Java Based Object Oriented Software. International Journal of Information Technology and Computer Science, 6(2), 1–13. https://doi.org/10.5815/ijitcs.2014.02.01

Masutani, Y., Nemoto, M., Nomura, Y., & Hayashi, N. (2012). Machine Learning in Action. https://doi.org/10.4018/978-1-4666-0059-1.ch008

Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., Abbas, S., & Soomro, T. R. (2021). Software defect prediction using ensemble learning: A systematic literature review. In IEEE Access (Vol. 9, pp. 98754–98771). Institute of Electrical and Electronics

Engineers Inc. https://doi.org/10.1109/ACCESS.2021.3095559

Mehta, S. (2021). Improved prediction of software defects using ensemble machine learning techniques. Neural Computing and Applications, 3. https://doi.org/10.1007/s00521-021-05811-3

Mendes-Moreira, J., Soares, C., Jorge, A. M., & De Sousa, J. F. (2012). Ensemble approaches for regression: A survey. ACM Computing Surveys, 45(1). https://doi.org/10.1145/2379776.2379786

Menzies, T., Greenwald, J., & Frank, A. (2007). Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Transactions on Software Engineering, 33(1), 2–14.

Mishra, A., & Catal, C. (2018). Empirical analysis of change metrics for software fault prediction. March. https://doi.org/10.1016/j.compeleceng.2018.02.043

Moser, R. (2008). A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. 181–190.

Ni, C., Member, S., Liu, W., Chen, X., & Member, S. (2017). A Cluster Based Feature Selection Method for Cross-Project Software Defect Prediction. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 32(6):, November. https://doi.org/10.1007/s11390-017-1785-0

Pattnaik, S., & Pattanayak, B. K. (2016). A survey on machine learning techniques used for software quality prediction. International Journal of Reasoning-Based Intelligent Systems, 8(1–2), 3–14. https://doi.org/10.1504/IJRIS.2016.080058

Piao, Y., Piao, M., Park, K., & Ryu, K. H. (2012). An Ensemble Correlation-Based Gene Selection Algorithm for Cancer Classification with Gene Expression Data. February 2016. https://doi.org/10.1093/bioinformatics/bts602

Rathore, Santosh S., & Kumar, S. (2015). Predicting number of faults in software system using genetic programming. Procedia Computer Science, 62(Scse), 303–311. https://doi.org/10.1016/j.procs.2015.08.454

Rathore, Santosh S., & Kumar, S. (2017). An empirical study of some software fault prediction techniques for the number of faults prediction. Soft Computing, 21(24), 7417–7434. https://doi.org/10.1007/s00500-016-2284-x

Rathore, Santosh S., & Kumar, S. (2019). A study on software fault prediction techniques. Artificial Intelligence Review, 51(2), 255–327. https://doi.org/10.1007/s10462-017-9563-

5

Rathore, Santosh S, & Kumar, S. (2021). An empirical study of ensemble techniques for software fault prediction. 3615–3644.

Rathore, Santosh Singh, & Kuamr, S. (2016). Comparative analysis of neural network and genetic programming for number of software faults prediction. RAECE 2015 - Conference Proceedings, National Conference on Recent Advances in Electronics and Computer Engineering, 328–332. https://doi.org/10.1109/RAECE.2015.7510216

Rathore, Santosh Singh, & Kumar, S. (2017). Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems. Knowledge-Based Systems, 119, 232–256. https://doi.org/10.1016/j.knosys.2016.12.017

Rawat, M. S., & Dubey, S. K. (2012). Software Defect Prediction Models for Quality Improvement: A Literature Study. International Journal of Computer Science Issues, 9(5), 288–296.

Read, J., Bifet, A., Pfahringer, B., & Holmes, G. (2012). Batch-Incremental versus Instance-Incremental Learning in Dynamic and Evolving Data. 313–323.

Regolin, E. N., De Souza, G. A., Pozo, A. R. T., & Vergilio, S. R. (2003). Exploring machine learning techniques for software size estimation. Proceedings - International Conference of the Chilean Computer Science Society, SCCC, 2003-Janua, 130–136. https://doi.org/10.1109/SCCC.2003.1245453

Sah, S. (2020). Machine Learning: A Review of Learning Types. ResearchGate, July. https://doi.org/10.20944/preprints202007.0230.v1

Sandhu, P., Singh, S., & Budhija, N. (2011). Prediction of level of severity of faults in software systems using density based clustering. 2011 IEEE International Conference on …, 9, 87–93. http://www.ipcsit.com/vol9/17-A3016.pdf

Sathya, R., & Abraham, A. (2013). Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. International Journal of Advanced Research in Artificial Intelligence, 2(2). https://doi.org/10.14569/ijarai.2013.020206

Schapire, R. E. (2013). Explaining AdaBoost. In Empirical Inference; Springer: Berlin/Heidelberg, Germany,.

Seliya, N., & Khoshgoftaar, Æ. T. M. (2007). Software quality estimation with limited fault data : a semi-supervised learning perspective. 327–344. https://doi.org/10.1007/s11219-

007-9013-8

Shatnawi, R., & Li, W. (2008). The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. Journal of Systems and Software, 81(11), 1868–1882. https://doi.org/10.1016/j.jss.2007.12.794

Studies, P., Alabdulrahman, R., & Science, C. (2014). A Comparative Study of Ensemble Active Learning.

Sun, Z., Song, Q., & Zhu, X. (n.d.). Using Coding Based Ensemble Learning to Improve Software Defect Prediction. 1–12.

Tu, Y. (2019). Machine learning. In EEG Signal Processing and Feature Extraction. https://doi.org/10.1007/978-981-13-9113-2_15

Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., & Haesen, R. (2008). Mining software repositories for comprehensible software fault prediction models. Journal of Systems and Software, 81(5), 823–839. https://doi.org/10.1016/j.jss.2007.07.034

Wah, Y. B., Ibrahim, N., & Hamid, H. A. (2018). Feature selection methods : Case of filter and wrapper approaches for maximising classification accuracy. Pertanika J. Sci. & Technol. 26, May 2020.

Wan, Y., & Bi, H. (2019). What Major " Socio-Scientific Topics " Should the Science Curriculum Focused on ? A Delphi Study of the Expert Community in China.

Wang, F. (2016). An Empirical Study of Ranking-Oriented Cross-Project Software Defect Prediction. November 2017. https://doi.org/10.1142/S0218194016400155

Wen, J., Li, S., Lin, Z., Hu, Y., & Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. Information and Software Technology, 54(1), 41–59. https://doi.org/10.1016/j.infsof.2011.09.002

Xu, Z., Liu, J., Yang, Z., An, G., & Jia, X. (2016). The Impact of Feature Selection on Defect Prediction Performance : An Empirical Comparison. https://doi.org/10.1109/ISSRE.2016.13

Yuan, X., & Allen, E. B. (2000). An Application of Fuzzy Clustering to Softw are Qualit y Prediction. 561.

Z., D. (1995). Introduction to Machine Learning and Software Engineering. Machine Learning, Ml, 1–36.

Zhou, T., Sun, X., Xia, X., Li, B., & Chen, X. (2019). Improving defect prediction with deep

forest. Information and Software Technology, 114(June), 204–216. https://doi.org/10.1016/j.infsof.2019.07.003

APPENDIXES

Appendix A: Features selected by CO best feature selection technique for software defect prediction

Datasets	*Selected features by CO best feature selection technique for all Selected Ensemble Learning*
EQ	'ck_oo_numberOfPrivateMethods','LDHH_lcom','LDHH_fanIn','numberOfNonTrivialBugsFoundUntil:','WCHU_numberOfPublicAttributes','WCHU_numberOfAttributes','CvsWEntropy','LDHH_numberOfPublicMethods','LDHH_numberOfPrivateAttributes','ck_oo_numberOfPublicAttributes','ck_oo_noc','numberOfCriticalBugsFoundUntil:','WCHU_noc','LDHH_numberOfAttriutesInherited','ck_oo_numberOfAttributesInherited','ck_oo_dit', 'LDHH_noc', 'WCHU_dit','WCHU_numberOfAttributesInherited','ck_oo_numberOfAttri utes', 'numberOfHighPriorityBugsFoundUntil:','ck_oo_numberOfPrivate Attributes', 'WCHU_numberOfPublicMethods','WCHU_numberOfMethods Inherited', 'ck_oo_numberOfMethodsInherited','ck_oo_numberOfPublic Methods'
JDT	'ck_oo_numberOfPrivateMethods','LDHH_lcom','LDHH_fanIn','numberOfNonTrivialBugsFoundUntil:','WCHU_numberOfPublicAttributes','WCHU_numberOfAttributes', 'CvsWEntropy','LDHH_numberOfPrivateAttributes', 'WCHU_numberOfPrivateMethods','ck_oo_numberOfPublicAttributes','ck_oo_noc','numberOfCriticalBugsFoundUntil:','ck_oo_wmc','WCHU_numberOfPrivateAttributes', 'CvsLogEntropy', 'WCHU_noc','LDHH_numberOfAttribu tesInherited', 'ck_oo_fanOut', 'ck_oo_dit','LDHH_noc', 'WCHU_fanOut', 'ck_oo_numberOfAttributes','numberOfHighPriorityBugsFoundUntil:','ck_oo_numberOfPrivateAttributes', 'LDHH_dit','WCHU_numberOfMethods Inherited', 'ck_oo_numberOfMethodsInherited'
	'ck_oo_numberOfPrivateMethods', 'LDHH_lcom', 'LDHH_fanIn','number OfNonTrivialBugsFoundUntil:','WCHU_numberOfPublicAttributes','WCHU _numberOfAttributes', 'LDHH_numberOfPrivateAttributes','LDHH_number OfPublicAttributes', 'WCHU_numberOfPrivateMethods','WCHU_numberOf Methods', 'ck_oo_numberOfPublicAttributes', 'ck_oo_noc','numberOfCritical

LC	BugsFoundUntil:', 'LDHH_numberOfPrivateMethods','CvsLogEntropy', 'WCHU_noc', 'LDHH_numberOfAttributesInherited','ck_oo_fanOut', 'ck_oo_numberOfAttributesInherited','ck_oo_numberOfMethods', 'ck_oo _dit', 'WCHU_dit','WCHU_numberOfAttributesInherited', 'WCHU_fanOut', 'ck_oo_numberOfAttributes', 'numberOfHighPriorityBugsFoundUntil:','num berOfMajorBugsFoundUntil:', 'WCHU_numberOfPublicMethods', 'LDHH _dit','CvsLinEntropy', 'WCHU_numberOfMethodsInherited', 'LDHH_fan Out','ck_oo_numberOfMethodsInherited','ck_oo_numberOfPublicMethods'
ML	'ck_oo_numberOfPrivateMethods', 'LDHH_lcom', 'LDHH_fanIn', 'number OfNonTrivialBugsFoundUntil:','WCHU_numberOfPublicAttributes','WCHU _numberOfAttributes', 'CvsWEntropy','LDHH_numberOfPrivateAttributes', 'CvsEntropy','LDHH_numberOfPublicAttributes', 'WCHU_numberOfPrivate Methods','ck_oo_numberOfPublicAttributes', 'ck_oo_noc','numberOfCritical BugsFoundUntil:', 'ck_oo_wmc','LDHH_numberOfPrivateMethods','CvsLog Entropy', 'WCHU_noc','LDHH_numberOfAttributesInherited', 'ck_oo_fan Out','ck_oo_numberOfAttributesInherited', 'ck_oo_numberOfMethods', 'ck_oo_dit', 'ck_oo_fanIn', 'WCHU_dit', 'WCHU_numberOfAttributes Inherited','WCHU_fanOut','ck_oo_numberOfAttributes','ck_oo_numberOfPrivateAttr ibutes','numberOfMajorBugsFoundUntil:', 'LDHH_dit', 'WCHU_ numberOfMethodsInherited','LDHH_numberOfMethodsInherited','ck_oo_numberOf MethodsInherited'
PDE	'ck_oo_numberOfPrivateMethods', 'LDHH_lcom', 'LDHH_fanIn', 'number OfNonTrivialBugsFoundUntil:','WCHU_numberOfPublicAttributes','WCHU_numbe rOfAttributes', 'LDHH_numberOfPublicMethods','LDHH_number OfPrivateAttributes', 'WCHU_numberOfPrivateMethods','ck_oo_noc', 'num berOfCriticalBugsFoundUntil:', 'ck_oo_wmc', 'WCHU_numberOfPrivate Attributes', 'CvsLogEntropy','LDHH_numberOfAttributesInherited', 'ck_oo_ fanOut','ck_oo_numberOfAttributesInherited', 'ck_oo_dit', 'WCHU_dit', 'LDHH_numberOfAttributes', 'ck_oo_numberOfAttributes', 'numberOf HighPriorityBugsFoundUntil:','ck_oo_numberOfPrivateAttributes', 'number OfMajorBugsFoundUntil:','LDHH_dit', 'WCHU_numberOfMethodsInheri

	ted','ck_oo_numberOfMethodsInherited', 'ck_oo_numberOfPublicMethods'

Appendix B: Accuracy Result of Base Classifier Algorithms to select best Base Classifier for Bagging, Adaboost and Stacking Ensemble Learning

Datasets	*NB*	*KNN*	*SVM*	*DT*	*RF*	*ET*	*GB*	*XGBoost*
EQ	0.71	0.813	0.903	0.887	0.915	**0.931**	0.91	0.908
JDT	0.728	0.908	0.914	0.929	0.961	**0.963**	0.936	**0.965**
LC	0.718	0.924	0.923	0.955	0.98	**0.992**	0.975	0.982
ML	0.647	0.907	0.919	0.95	0.976	**0.982**	0.935	0.973
PDE	0.664	0.902	0.929	0.927	0.971	**0.978**	0.926	0.97

Appendix C: Sample Implementation of Correlation Based Feature Selection(CFS)

 # Implementation of Merit Calculation Function

```
import numpy as np
from mutual_information import su_calculation

def merit_calculation(X, y):

    n_samples, n_features = X.shape
    rff = 0
    rcf = 0
    for i in range(n_features):
        fi = X[:, i]
        rcf += su_calculation(fi, y)
        for j in range(n_features):
            if j > i:
                fj = X[:, j]
                rff += su_calculation(fi, fj)
    rff *= 2
    merits = rcf / np.sqrt(n_features + rff)
    return merits
```

 # Implementation of Correlation base feature selection(cfs) Function

```
def cfs(X, y):
    n_samples, n_features = X.shape
    F = []
    # M stores the merit values
    M = []
    while True:
        merit = -100000000000
        idx = -1
        for i in range(n_features):
            if i not in F:
                F.append(i)
                # calculate the merit of current selected features
                t = merit_calculation(X[:, F], y)
                if t > merit:
                    merit = t
                    idx = i
                F.pop()
        F.append(idx)
        M.append(merit)
        if len(M) > 5:
            if M[len(M)-1] <= M[len(M)-2]:
                if M[len(M)-2] <= M[len(M)-3]:
                    if M[len(M)-3] <= M[len(M)-4]:
                        if M[len(M)-4] <= M[len(M)-5]:
                            break
    return np.array(F)
```

 # Implementation of Information Gain, Conditional Entropy and Symmetrical Uncertainty Function

```
import entropy_estimators as ee

#This Function Calculates Information Gain
def information_gain(f1, f2):
    ig = ee.entropyd(f1) - conditional_entropy(f1, f2)
    return ig

#This Function Calculates Conditional Entropy
def conditional_entropy(f1, f2):
    ce = ee.entropyd(f1) - ee.midd(f1, f2)
    return ce

#This Function Calculates Symmetrical Uncertainty
def su_calculation(f1, f2):
    # calculate information gain of f1 and f2, t1 = ig(f1,f2)
    t1 = information_gain(f1, f2)
    # calculate entropy of f1, t2 = H(f1)
    t2 = ee.entropyd(f1)
    # calculate entropy of f2, t3 = H(f2)
    t3 = ee.entropyd(f2)
    # su(f1,f2) = 2*t1/(t2+t3)
    su = 2.0*t1/(t2+t3)

    return su
```