Analysis and Selection of Optimized Machine Learning Techniques for Software Bug Prediction

By

Noor ul Ain

Supervisor

Brig (Retd) Associate Professor Dr. Fahim Arif

A thesis submitted to the Department of Computer Software Engineering, Military College of Signals (MCS), National University of Sciences and Technology, Islamabad, Pakistan, in partial fulfillment of the requirement for the degree of MS in Software Engineering

January 2023

Analysis and Selection of Optimized Machine Learning Techniques for Software Bug Prediction

By

Noor ul Ain

00000318578

Supervisor

Brig (Retd) Associate Professor Dr. Fahim Arif

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Science in Software Engineering MSSE

In

Department of Computer Software Engineering, Military College of Signals (MCS), National University of Sciences and Technology, Islamabad, Pakistan.

(January 2023)

Thesis Acceptance Certificate

Certified that final copy of MS/MPhil thesis entitled **"Analysis and Selection of Optimized Machine Learning Techniques for Software Bug Prediction"** written by **Noor ul Ain**, (Registration No. **00000318578**), of Department of Computer Software Engineering Military College of Signals (MCS), has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: ____________________

Name of Supervisor: **Brig (Retd) Associate Professor Dr. Fahim Arif**

Date: ____________________

Signature(HoD): ____________________

Date: ____________________

Signature (Dean/Prinicpal): ____________________

Date:____________________

Declaration

I, *Noor ul Ain* declare that this thesis titled "Analysis and Selection of Optimized Machine Learning Technique for Software Bug Prediction” and the work presented in it is my own and has been generated by me as a result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a Master of Science degree at NUST
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at NUST or any other institution, this has been clearly stated
3. Where I have consulted the published work of others, this is always clearly attributed
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work
5. I have acknowledged all main sources of help
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself

Noor ul Ain,
NUST00000318578 MSSE26

Copyright Notice

I thank Almighty Allah, The Most Gracious and The Most Merciful.

This thesis is dedicated to the apple of my eye, *my beloved son, Adan*, whose arrival in the world urged me to complete my thesis.

Abstract

Software Bug Prediction is an active research area and is being widely explored with the help of Machine Learning. Since bug prediction is now considered as an important measure of SDLC, we need to have optimized techniques for making predictive models. Presently transfer learning and ensemble learning approaches are being researched much. However, previous studies are not sufficient in this regard. So in this paper a framework is created by using multiple techniques to explore their effectiveness when combined in one model. The techniques involved feature selection which is used to reduce the dimensionality and redundancy of features and select only the relevant ones; transfer learning is used to train and test the model on different datasets to analyze how much of the learning is passed to other dataset; and ensemble method is utilized to explore the increase in performance upon combining multiple classifiers in a model. Four NASA and four Promise datasets are used in the study, the results of which show an increase in the performance of the model by providing better AUC-ROC values when different classifiers were combined in the model. Thus revealing that use of amalgam of techniques such as used in this study, feature selection, transfer learning and ensemble methods prove helpful in optimizing the software bug prediction models and provide high performing, useful end model.

Keywords: *Software bug prediction, Transfer learning, Ensemble learning method, Feature selection, Machine Learning*

Acknowledgments

Quite some time has passed since I began my degree, numerous things happened, various new places visited, survived a pandemic and life has changed altogether. However, it is time I finally close this chapter. I would like to thank my parents, my parents-in-law and my sisters for their constant support. I am grateful to my husband for his emotional support and also his never-ending nudging to complete it. A special thanks to my mother for keep asking me the thesis progress updates every fifteen minutes. I am indebted to my supervisor Dr. Fahim Arif for his politeness and firm belief in me and my committee member, Dr. Saddaf Rubab, for always being there for guidance. Lastly, I want to thank my institution, MCS, for providing me the opportunity to learn and grow.

Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

SBP	Software Bug Prediction
SDP	Software Bug Prediction
CPDP	Cross-Project Defect Prediction
ML	Machine Learning
RF	Random Forest
KNN	K-Nearest Neighbors
NB	Naive Bayes
DT	Decision Tree
SVM	Support Vector Machine
NB	Naive Bayes

Chapter 1

Introduction

1.1 Introduction

In this era, trend is shifting towards automating the process and procedures. For this reason, software development industry has thrived and more emphasis is being placed on quality software production. Bug prediction techniques are help developers concentrate on parts of code/software that are particularly prone to have bugs. This helps to efficiently allocate resources in testing and fixing of error prone modules and hence producing high quality product at lower cost. Technically bug/defect predictor is a model of machine learning which is applied on historical software metrics to predict defects in software modules. This efficiency of model is based on the quality of provided training data and the classification technique used. This chapter will provide a walk through the importance and need of Software Bug Prediction (SBP) in todays time and technology and how it can impact the success of a software and software based products. The problem statement is also discussed in this chapter, for which the main contributions and methodology used in this research project is elaborated while the thesis outline is described at the end of chapter.

1.2 Software Bug Prediction – the need of time/Importance of SBP

Each day in the world of Information Technology (IT) brings new changes; new software releases, new version of applications or languages, or entirely new techniques and programs. The world is rapidly shifting towards software-based products thus drastically increasing our reliance on software. Software houses and companies work hard to meet the need and develop high-end software products. However, often a software crisis or failure occurs due to increased complexity, short time to market and high customer demands consuming resources like time and budget. This situation gives rise to the idea of successful, error-free software putting emphasis on quality software production. Many models, principles and techniques are followed to achieve this notion such as small iterations, documentations, user interaction and well-organized process; still some inevitable bugs occur causing a great distress to the software users and owners.

Testing the whole software system completely and thoroughly is practically not possible especially with the limited testing resources [1]. The unexpected behavior of a system against the provided requirements show the presence of bug, and by promptly identifying them, developers can efficiently allocate testing resources and enhance a system's architectural design by determining the high-risk system components. [2]. The figure below represents the bug prediction phase and detection phase in a conventional style.

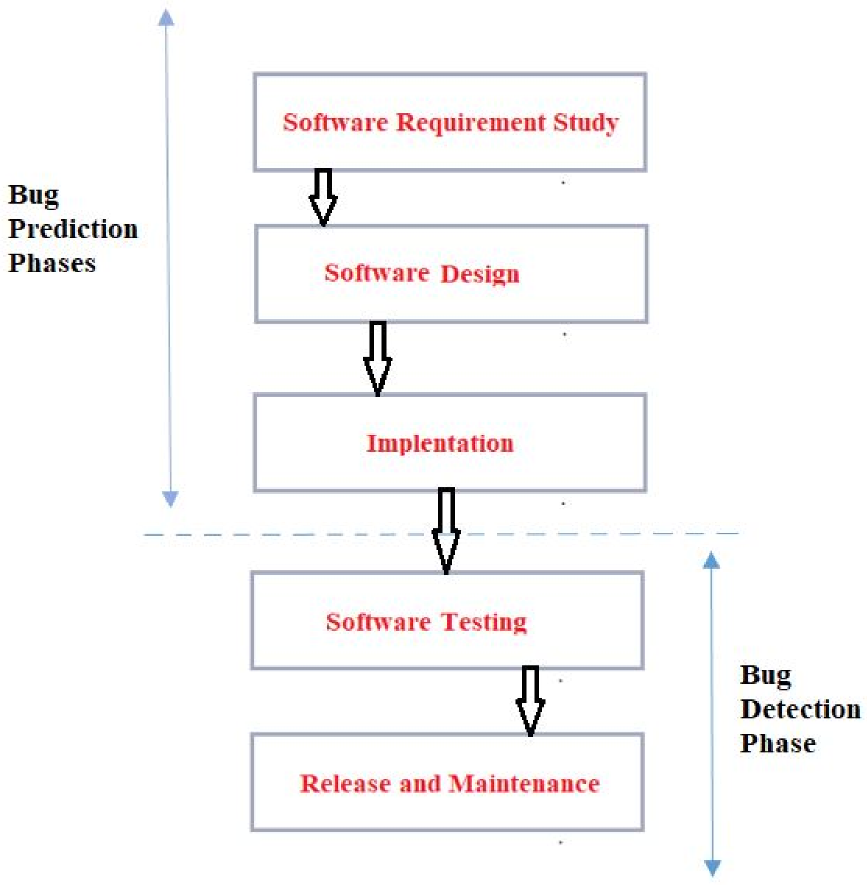


Figure 1.1: Phases of Bug Prediction and Detection

In order to mitigate these defects an analysis of predicting them before they are born is necessary. This inspires the birth of Software Bug Prediction methods, which can forecast the software bugs in initial stages of development, increasing the efficiency and performance of the final product. These methods are used to help developers concentrate on parts of code/software that are particularly prone to have bugs. It also helps in efficient allocation of resources in testing and fixing of error prone modules hence producing high quality product at reduced cost.

For this purpose, machine-learning models have proved to be very effective in achieving required results. These models are applied on historical software metrics data to predict defects in software modules. They help narrowing down and reducing the testing hardships of faulty modules by identifying such modules of software system, whose chances of being fault-prone are higher [3]. Yet the efficiency of model depends on certain factors like dataset being used, its quality, used features and the choice of machine learning classification technique used. One of the key factor for success criteria of bug prediction methods involves the prediction of the correct occurrence of bug and understanding the SDLC process flow. Any stage of SDLC can make use of Machine Learning algorithms for bug prediction, be it identifying problems, planning and design, development, testing phase, deployment and maintenance irrespective of the type of SDLC model employed [4].

Prediction models serve a great advantage in development environments as once incorporated, they can give feedback to the developers while they are in the development process. This might give rise to the notion that models are 100 % accurate, but in actual expecting 100% accuracy of prediction is unreasonable.Additionally, these models function differently depending on the dataset, which frequently results in conflicting fault predictions in a software project [3]. These models are constructed in ways to achieve excellence, yet false prediction are unavoidable. The false predictions are differentiated in two main categories, one where clean code gets classified as defected while the other one in which a defected code gets classified as clean code [5]. During such a situation trusting a model becomes difficult therefore, we need to obtain the best model that compensates for fake/false predictions [1]. Efforts have been made to examine the accuracy and complexity of models, although there are no standard benchmarks for comparing models. This brings about the compelling utilization of ensemble method for software bug prediction, as it uses various methods for the provided dataset to give better prediction results. According to observations, various methods have resulted in varied levels of prediction performance, but none of them have consistently delivered the most accurate predictions across various datasets. In this regard, there was a lot of theoretical and empirical evidence in favor of using the ensemble method to get better results for fault prediction. The ensemble method promises to improve fault prediction by reducing the shortcomings of individual methods. [3].

1.3 Problem Statement

The world is rapidly shifting towards software-based products, increasing our reliance on software drastically. Software development industry follows a lot of models, principles and techniques to achieve error-free software still some inevitable bugs occur.

- To mitigate the software bugs many ML techniques are devised by researchers but no optimized technique is found among all of them.
- Machine learning based software bug prediction methods require certain measures and parameters to train the models but how we can optimize and generalize the performance of the model is an open research area.

1.4 Research Questions

This research is focused on creating an optimized model to help predict bugs in software and answers following research questions:

- What ML techniques are widely used to develop software bug prediction models?
- What impact does Ensemble Learning technique has, on the efficiency of a model in comparison to the individual classifiers?
- What is the effect of cross project bug prediction on a model?

1.5 Main Contributions

A novel machine learning approach is devised after conducting extensive research on the developed models and approaches and evaluated the results using comparison. In order to optimize the model we used the ensemble-learning method to create the model that will help in the cross project bug prediction. Cleaned datasets were chosen for the research and feature selection technique was carefully applied on them to curate the best features for our model. In Chapter 3, we have discussed the detail of our implementation and Chapter 4 will walk us through the results and experiments.

1.6 Methodology

The research aims to develop a novel model for software bug prediction based on machine learning techniques and method. Four major steps are involved in this research: Cleaning the dataset followed by Feature Selection, leading towards training the source dataset with individual classifiers and then with ensemble learning method to draw a comparison between both the performances and finally testing the performance of trained model on the target dataset. A general Architecture Diagram in Figure 1.2 explains the overall project. The detailed framework architecture is shown in Chapter 3.

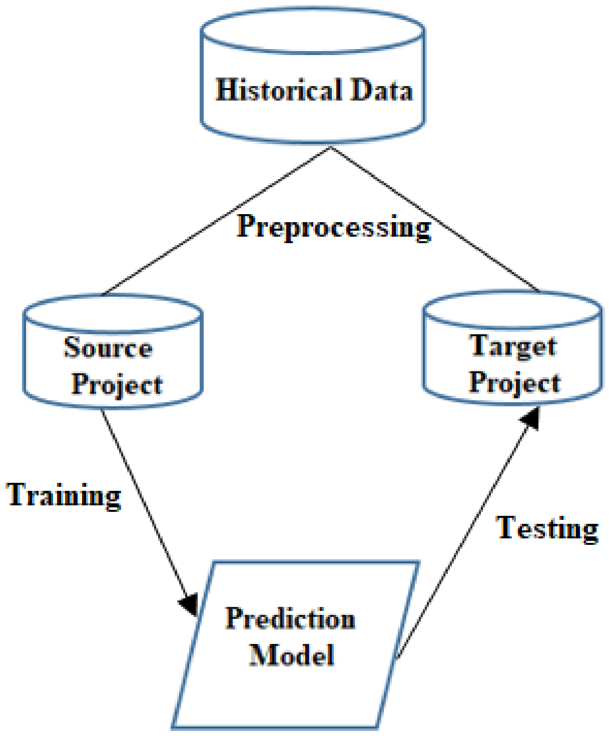


Figure 1.2: General Structure of a Machine Learning Model

1.7 Thesis Outline

The research for such a system is a multi-step process and below is the breakdown of this research.

Chapter 1 Introduction: An overview of software, need of software defect prediction in software industry concluding with research framework and questions.

Chapter 2 Literature Review: Provides a complete journey of exploration in the discipline of software bug prediction. A review of datasets, preprocessing of data, selection of features and approaches used for it and classification techniques of software defect prediction. (Problem Statement & Research Objectives).

Chapter 3 Technical Approach: How the software defect prediction will use the learning insight of machine learning algorithms for the system. The theoretical description of our model. The detail of implementation of the system using WEKA has been presented in this chapter following the complete details.

Chapter 4 Experimentation and Results: has been presented in this chapter and the results are discussed.

Chapter 5 Conclusion and Future Work: The said section gives a recap of the entire work done along with providing future direction for research.

Chapter 2

Literature Review

2.1 Introduction

This chapter discusses these previous approaches used for software bug prediction, machine learning and prediction techniques. Cross-Project Bug Prediction has, lately, been considered as a very sought-after trend in the industry of software bug prediction. There is numerous work done in the past about bug prediction and the likes. The quantity of source code errors has been predicted using a variety of metrics. The frequency of source code errors has been predicted using a variety of metrics. The majority of the previous research on software fault prediction is restricted to using comparison methods for analysis of each machine learning technique. Some of them employed only a few techniques and offered the contrast between them and others suggested methods as an extension of prior work. The literature survey also reveals that the use of combination of techniques gives better results.

2.2 Software Bug Prediction (SBP) Approaches

There are various approaches to create software bug prediction models mainly depending on factors like the required output, availability of datasets, features in a dataset, ML classifiers etc. The previous models often ignored some of the above-mentioned factors, which made them less effective. Later on, with the rapid growth of complexity of a software, the area of software bug prediction gained much popularity and turned into a sought-after research area in software engineering field. Many researchers are

attracted towards this field proposing a variety of techniques, framework and models for bug prediction. Additionally, there are researchers working on enhancing the models and techniques already in use. The field of software bug prediction still suffers from a lot of ambiguity, despite numerous efforts. Although numerous models and frameworks have been proposed, each method has its own limitations. Bugs are detected by a variety of machine learning algorithms, and datasets are made accessible to the public so that researchers can carry out their experiments without worrying about data. For machine learning techniques to be useful in bug prediction, it is necessary to examine the experimental evidence gathered from previous studies [6].
The figure 2.1 shows what normally is included in literature review in the area of SBP. Often times there are surveys or reviews conducted, discussions of previously used techniques, their goods and bads, the latest trends and famous topics as all this is very much required for a researcher to conduct a relevant and fruitful research project.

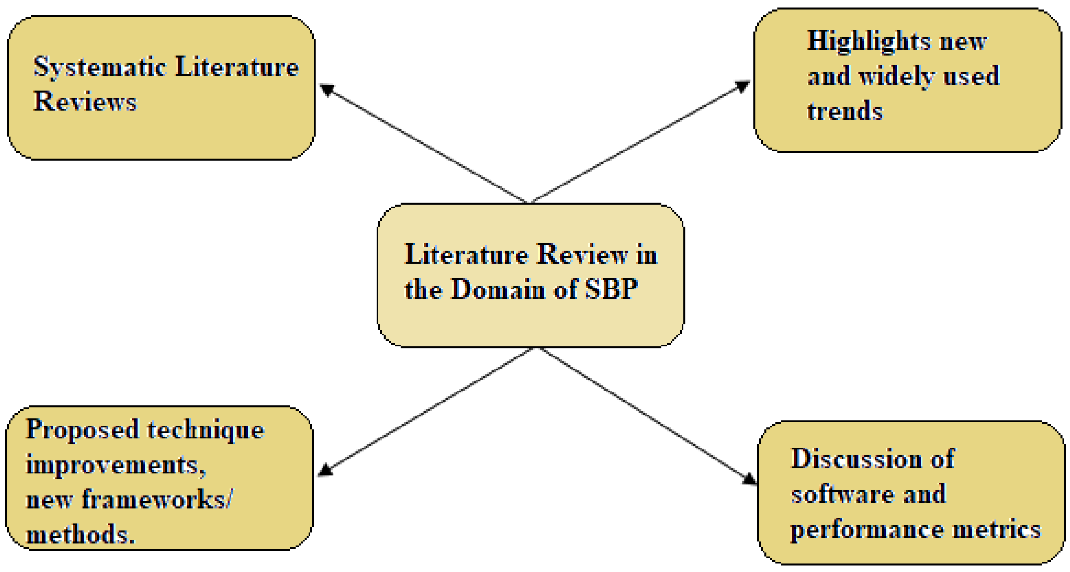

Systematic Literature Reviews
Highlights new and widely used trends
Literature Review in the Domain of SBP
Proposed technique improvements, new frameworks/ methods.
Discussion of software and performance metrics

Figure 2.1: Literature Review in the domain of Software Bug Prediction

The following is a list of related works in SBP field displayed in table 2.1 below. The table shows the year and aim of study, the methodology used to conduct it and the advantages and disadvantages of the used method. The table gives a quick idea to the reader about the mentioned study and its usefulness.

Table 2.1: SBP Approaches

Study	Goal	Dataset	Methodology	Advantages	Disadvantages
[1] (2016)	Bug proneness index prediction	Eclipse JDT Core	Linear Regression Model using Marginal R square values	Proposed an approach to select the minimal number of best performing software metrics for fault prediction	The study was performed over only one software system
[2](2015)	Bug prediction	15 software bug dataset from PROMISE data repository.	NaiveBayes, MLP, SVM, AdaBoost, Bagging, Decision Tree, Random Forest, J48, KNN, RBF and K-means	Carried out a comparative performance analysis of multiple machine learning techniques for software bug prediction.	No comparison with previous studies has been provided. Evaluation of the results not thorough
[3](2017)	Number of faults in software module	11 software fault datasets from PROMISE data repository	Ensemble method using linear regression, multilayer perceptron, genetic programming, zero-inflated Poisson regression, and negative binomial regression.	Heterogeneous ensemble method using a linear combination rule and a non-linear combination rule based approaches for the ensemble. Improved results compared to the single fault prediction technique.	
[6] (2020)	Analysis of software bug prediction using machine learning techniques	Exploring different datasets in 31 research studies	Machine Learning Techniques	Conducted a review to analyze and evaluate the performance of software bug prediction model using machine learning techniques	The study can be used as reference but did not provide any redeem of the stated problems

Continued on next page

Table 2.1 – *Continued from previous page*

Study	Goal	Dataset	Methodology	Advantages	Disadvantages
[7](2019)	Software Bug Detection	PROMISE software defect dataset repository KC1, CM1, PC3, PC4	Genetic Algorithm and Deep Neural Network (DNN) classification techniques using MATLAB tool and compared it with existing techniques such as NB classifier, SVM, Decision Tree, KNN, etc.	Found that the proposed hybrid approach performs better that existing classification schemes and facilitates feature optimization reducing the computational time.	Only few datasets have been used for empirical investigations
[8](2015)	Defect prediction in multiple phases across SDLC	Input data taken from 45 real time projects	Neural Network	Proposed a probabilistic defect prediction approach that provides a defect range (min, max, and mean) on a graphical user interface for easy access by project managers.	Only theoretical validation presented. Lacks generalization
[9](2020)	Investigation of Machine learning techniques in Software bug prediction field	Exploring multiple datasets in 165 research studies	Machine Learning Methods	A review investigation has been conducted over the ML technique in SDP to highlight the recent and widely used activities in this domain.	The study lacks details

Continued on next page

Table 2.1 – *Continued from previous page*

Study	Goal	Dataset	Methodology	Advantages	Disadvantages
[10](2021)	Systematic literature review done in defect prediction area	Inspection of mulitple datasets in 46 research studies	Machine Learning Techniques	Conducted a review to provide latest trends and advances in ensemble learning approach concluding that they perform significantly better than individual classifiers.	Only theoretical validation presented.
[11](2020)	Defect prediction using generative model	NASA dataset	Stack Sparse Auto-Encoder (SSAE) and Deep Belief Network (DBN)	Generative deep learning models were used to manage data unbalancing.	Validation of proposed approach is needed through more case studies
[12](2020)	Improving accuracy in software defect prediction PROMISE dataset (CM1, JM1, KC1, KC2, and PC1)	Bayes network, Random forest, SVM, and the Deep Learning	Investigated the effect of deep learning as a Neural Model and compared it with other techniques which improved fault prediction performance significantly	Dataset contains unbalanced classes. Lacks preprocessing of dataset	
[5](2020)	Fault identification and recovery	3 NASA datasets: JM1, CM1, PC1	Decision Tree (DT), k-nearest neighbors (KNN), Logistics Regression (LR), Naïve Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM)	Proposed automated fault recovery inside software through a predictive model.	The proposed methods needs proper validation and thorough evaluation.

Continued on next page

Table 2.1 – *Continued from previous page*

Study	Goal	Dataset	Methodology	Advantages	Disadvantages
[13](2012)	Hybrid feature selection approach for improved fault prediction	KC1 data set (Promise software engineering repository)	Naïve Bayes (NB), J48, Radial Basis Function Network (RN)	Utilized a hybrid feature selection method (Filter-based and wrapper FS method) to examine the effect of a smaller feature set on the learning algorithm's performance	The research only made use of one fault prediction dataset.
[14](2019)	Investigating the effect of feature selection and handling imbalanced data in defect prediction	NASA MDP datasets: KC1, KC3, MC2, MW1, PC4 and PC5	MLP, Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), K Nearest Neighbor (kNN), Bayes Net (BN)	Proposed framework with ensemble learning techniques and used multiple search methods for feature subset selection.	No comparison with previous studies has been provided.
[15](2020)	Tackles Fault prone module classification problem	NASA MDP Dataset	k-nearest neighbor (KNN), random forest (RF), SVM	Hybrid Particle Swarm Optimization-Modified Genetic Algorithm (PSO-MGA) is used for feature selection and bagging techniques for increasing classification accuracy.	More datasets of different domains are required to prove the usefulness of proposed approach.
[16](2019)	Investigation on feature selection techniques	NASA and PROMISE datasets	Feature subset selection and three feature-ranking approaches. Classifiers: K-Nearest Neighbors (KNN) and Naive Bayes (NB)	Found that selection of representative feature subset or setting a reasonable proportion of selected features improves the performance of Cross-Project Defect Prediction (CPDP).	Empirical study not exhaustive. No comparison with other feature selection techniques has been provided

Continued on next page

Table 2.1 – *Continued from previous page*

Study	Goal	Dataset	Methodology	Advantages	Disadvantages
[17](2017)	Software Defect Detection	PC1, PC2, PC3 and PC4 from PROMISE repository	Random Forest as classifier and Bat-based search Algorithm (BA) for the feature selection	Proposed an approach to have high performance of defect prediction model and applied three-feature selection metaheuristic algorithm to get the most effective features.	No concrete results have been presented to prove the effectiveness of proposed approach
[18](2019)	Addressing the issue of class disparity	NASA Dataset: CM1, MW1, PC1, PC3, and PC4.	SMOTE, Adaboost, Bagging, Naïve Bayes	To diminish the impact of class irregularity issue and misclassification, SMOTE and ensemble technique is applied	The case study needs more information to get validation for the proposed method
[19](2015)	Analysis of issues in software defect prediction		Discussion of problems	Analyzed a total of six major defect prediction issues.	No specifics is provided by the study regarding the problem's solution.

Although individual classifiers have given good performance in predicting number of defects but scholars are trying to devise hybrid frameworks to further improve defect prediction accuracy.

2.3 Machine Learning Techniques

From the literature review we explored that various frameworks and methods have been proposed to perform bug prediction by combining data preprocessing, FS, data sampling, and machine learning classifiers in a systematic manner to build models. The review of past work has given us some insight about the widely and commonly used measures in this domain which are shown in the table 2.2 below. Researchers have preferred them in order to ease the evaluation of the performance of their work. Apart from the mentioned approaches, different kinds of preprocessing methods are utilized in the cleaning of data and feature selection or feature ranking methods are also utilized to reduce the dimensionality of dataset. After data cleaning, different classifiers are applied on the dataset, either individual classifiers or ensemble methods, to train the model.

Table 2.2: Widely used tools in SBP

Dataset	NASA, PROMISE, SOFTLAB, Relink, AEEEM
Performance Measures	AUC, Accuracy, MCC, F-measure, Recall and Precision
Tool used	WEKA, Python, MATLAB, sklearn, LIBSVM, KEEL

2.4 Summary

This chapter sheds light on the work done previously, in the domain of software bug prediction, the approaches, methods, advantages and limitations of their work is shown. The widely used dataset, tools and approaches for creating a prediction model has also been considered.

‘

Chapter 3

Proposed Approach

3.1 Introduction

This chapter gives details regarding introduction to proposed solution, its theoretical concept and what best ways should be opted to utilize this technique. The framework of the architecture of the research project is also discussed both, diagrammatically and theoretically. It discusses the technique utilized and its implementation details. There is a detailed analysis of the used data set, its availability, features, relevancy of features and metrics. The machine learning classifier, tools, and performance metrics used in this study are also explained. This part will be concluded with proposition of optimized machine learning technique that will help predict software bugs with greater accuracy.

3.2 Theoretical Concept of the Proposed System

As the undertaken research project involves model with optimized technique for software bug prediction, it requires clean dataset with relevant features, efficient classifier and valid training and testing of model. For this purpose, two distinct methods are brought together to build the model more robust and novel among the previously built models seen in the literature survey. These two methods are discussed as follows:

- Transfer Learning employing cross project defect prediction
- Ensemble based method

3.2.1 Transfer Learning

The technique of transfer learning is unique in its form as it helps to save knowledge gathered from solving one problem and applies it to a related but different problem. We can also say that a model created for one task can be used as the basis for another through transfer learning. For example, the knowledge acquired to identify light vehicles can be applied to identify heavy vehicles as well. Thus, this technique serves as a great way to solve research problems in machine learning domain.
In this study the focus is on the cross-company bug prediction situation where source data and target data belongs to various companies/projects. [20]. The model is built using one project considered as source project and employed for prediction on another project called target project [16]. The set of features present in both of the projects are kept same but we employ feature selection approach to reduce irrelevant features.

3.2.2 Ensemble Technique

The Ensemble learning technique utilizes multiple classifiers in building a model for classification in order to improve the overall performance and efficiency of bug prediction. It also improves the capability of generalization of the model and decreases the problem of class imbalance. Different data is trained with different classifier so that each classifier generates its own classification error. However, not all classifiers produce the same set of corresponding errors. The ensemble learning methods can reduce biased learning caused by class imbalance classification by combining these classifiers through certain mechanisms. In this scenario, the ensemble learning techniques of boosting (Bst) and bagging (Bag) are widely used [21].

3.3 Framework Architecture

To make our software prediction model we opted transfer learning while training our model where cross project defect prediction is utilized. Our experiment is based on two phases where our main objective is to:

1. Analyze different techniques and
2. Chose optimal measures for defect prediction

For this purpose, the Figure 3.1 shows the framework architecture. It discusses the major aspects involved in creating the model. The chosen datasets are passed through feature selection process to extract the best and most relevant features as high dimensionality of dataset often ruin the results. From the newly curated datasets, we choose source project in order to train it on our classifier and then test the model with a different target project. The performance results of the model will reveal the most suitable classifiers and techniques for this research.

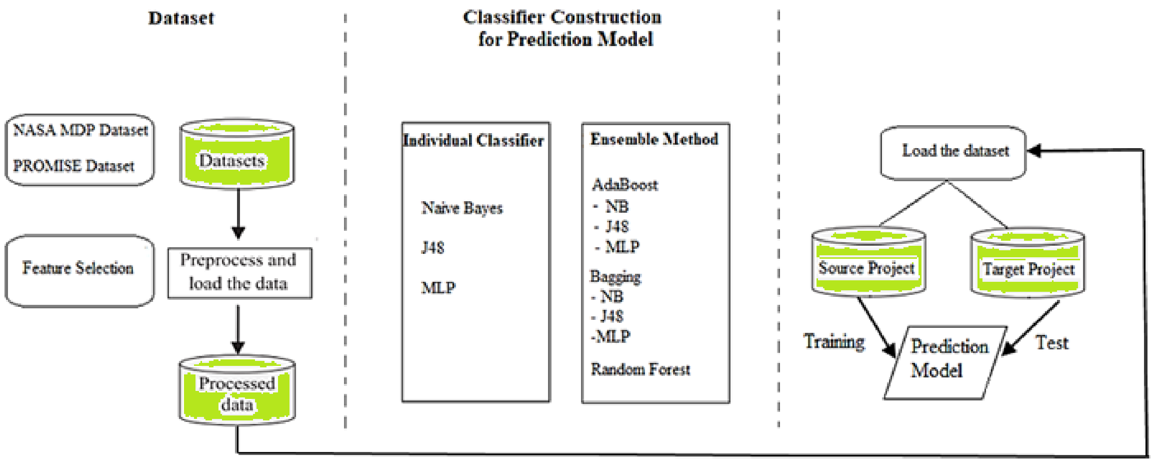


Figure 3.1: Framework Architecture

3.4 Data Sets

The performance of proposed framework is assessed on four NASA benchmark datasets and four datasets from Promise Repository. These datasets are publicly available and consist of historical data of software modules. Many studies have utilized these datasets in their researches and this is the primary reason of our interest in them as it will be easier to compare our results with them. They include several features and a known output class that determines the defectiveness of an instance. Based on data available for other features, this output class is predicted by the prediction model. The datasets have many projects with various attributes, sizes, and defective rates that helps to check the generality of research [21].

3.4.1 NASA MDP Dataset

From the NASA MDP Dataset, which is freely accessible on PROMISE Software Engineering Repository, we selected CM1, MW1, PC1, and PC2. The features shown in Table 3.1 belong to this dataset. McCabe and Halstead feature extractors of source code provide data from software for storage management that is used for receiving and processing ground data. In an effort to objectively characterize code features associated with software quality, these features were defined in the 1970s. [15].

Table 3.1: Features in NASA Dataset

ID	Feature Name	*ID*	Feature Name
1.	LOC_BLANK	20.	HALSTEAD_EFFORT
2.	BRANCH_COUNT	21.	HALSTEAD_ERROR_EST
3.	CALL_PAIRS	22.	HALSTEAD_LENGTH
4.	LOC_CODE_AND_COMMENT	23.	HALSTEAD_LEVEL
5.	LOC_COMMENTS	24.	HALSTEAD_PROG_TIME
6.	CONDITION_COUNT	25.	HALSTEAD_VOLUME
7.	CYCLOMATIC_COMPLEXITY	26.	MAINTENANCE_SEVERITY
8.	CYCLOMATIC_DENSITY	27.	MODIFIED_CONDITION_COUNT
9.	DECISION_COUNT	28.	MULTIPLE_CONDITION_COUNT
10.	DECISION_DENSITY	29.	NODE_COUNT
11.	DESIGN_COMPLEXITY	30.	NORMAL_CYCLOMATIC_COMPLEXITY
12.	DESIGN_DENSITY	31.	NUM_OPERANDS
13.	EDGE_COUNT	32.	NUM_OPERATORS
14.	ESSENTIAL_COMPLEXITY	33.	NUM_UNIQUE_OPERANDS
15.	ESSENTIAL_DENSITY	34.	NUM_UNIQUE_OPERATORS
16.	LOC_EXECUTABLE	35.	NUMBER_OF_LINES
17.	PARAMETER_COUNT	36.	PERCENT_COMMENTS
18.	HALSTEAD_CONTENT	37.	LOC_TOTAL

3.4.2 Promise Dataset

The data in Promise dataset refers to open-source Java systems from which we chose ant-1.7, camel-1.6, ivy-2.0 and xalan-2.4. The features present in them are shown in table 3.2. The table shows all of the twenty features present in the dataset. The first column displays the feature ID while the second and third column shows the feature name and detail respectively. These IDs are used in another table to show the selected features which are used in this study.

Table 3.2: Features in PROMISE Dataset

ID	Feature Name	Feature Detail
1.	wmc	Weighted methods per class
2.	dit	Depth of inheritance tree
3.	noc	Number of children
4.	cbo	Coupling between object classes
5.	rfc	Response for a class
6.	lcom	Lack of cohesion in methods
7.	ca	Afferent couplings
8.	ce	Efferent couplings
9.	npm	Number of public methods
10.	lcom3	Lack of cohesion in methods, different from LCOM
11.	loc	Lines of code
12.	dam	Data access metric
13.	moa	Measure of aggregation
14.	mfa	Measure of functional abstraction
15.	cam	Cohesion among methods of class
16.	ic	Inheritance coupling
17.	cbm	Coupling between methods
18.	amc	Average method complexity
19.	max_cc	Maximum McCabe's cyclomatic complexity
20.	avg_cc	Average McCabe's cyclomatic complexity

3.4.3 Preprocessing

First step in the proposed framework is data preprocessing. [22] provides NASA datasets in two versions. The version of the dataset known as DS' includes instances that are inconsistent and duplicated, whereas the version known as DS' is a cleaner version..

Originally, these datasets were available at NASA website; however, they are removed from this source. Backup of 12 cleaned NASA datasets are available at [23]. We have taken 4 cleaned and widely used datasets from the available datasets at [23] which include CM1, MW1, PC1, PC2. Previous studies have already discussed and used these cleaned versions of datasets in their experiments. The other four datasets have been taken from PROMISE repository available at [24]. They contain 20 Object Oriented metrics as independent features and defect-proneness of class as dependent variable. The criteria of cleaning as stated in [22] is shown in Table 3.3.

Table 3.3: Cleaning Criteria of NASA Dataset

Sr. No	Category of Data Quality	Description
1.	Identical cases	For this case various instances have similar values for all features together with class label
2.	Inconsistent cases	In this situation there are two or more instances which have same values for all features except for class label.
3.	Cases with missing values	This case refers to instances which hold one or more missing observations.
4.	Cases having conflicting feature values	An instance has two or more metric values that go against some referential integrity constraint in this case.
5.	Cases with implausible values	This case refers to instances that defy some integrity constraint. For example, value of LOC being 1.1

3.4.4 Feature selection

In recent years, feature selection technique is vastly implemented in building software defect prediction models as the high-dimensionality of features can disturb the performance of model therefore we select such features that appear relevant and similar to the class. This way the performance of model can be boosted and made reliable.

For our study, we have employed feature subset selection using CFS (correlation-based feature selection). It evaluates the redundancy between different features and analyze their individual predictive ability. In order to get the optimal subset of features, Best-first search is applied. Features with more relevancy to class are preferred over the features irrelevant to other features. While using the datasets in our prediction model we only use the shared features of each source and target dataset as the feature numbers are not the same in all datasets.
The features that are selected from each dataset are listed in Table 3.4. The table represents dataset name, ID of selected feature and number of total selected features. The ID list in table 3.1and 3.2 shows the ID of selected feature for NASA and PROMISE datasets respectively.

Table 3.4: Features in NASA Dataset and PROMISE Dataset after feature selection

	Dataset Name	Selected Feature ID	No. of Selected Feature
NASA	CM1	5, 16, 18, 34, 36	5
	MW1	1, 3, 5, 13, 18, 27, 29, 35, 37	9
	PC1	1, 4, 5, 8, 16, 17, 18, 30	8
	PC2	5, 17, 29	3
PROMISE	Ant-1.7	4, 5, 6, 11, 15, 18, 19	7
	Xalan-2.4	4, 5, 6, 10, 11, 17, 19	7
	Camel-1.6	2, 3, 4, 6, 7, 9, 15, 16, 17, 18, 20	11
	Ivy-2.0	1, 4, 5, 8, 9, 11, 13, 18	8

3.5 Machine Learning Classifier

A classifier is an algorithm, or the set of rules used to categorize or classify data whereas a model is the end result or the outcome of the classifier's machine learning. The training of the model is carried out using classifier, which also helps in the classification of data. In the scenario of this study, this step consists of choosing individual classifiers that were mostly used in artificial intelligence and the integration of well-known algorithm to form an ensemble-learning model.

In the first step, we chose three individual classifiers i.e., Naïve Bayes, Decision Tree (J48) and Multi-Layer Perceptron (MLP). These are said to be frequently used classifiers and give efficient performance in the prediction of defects [2, 9]. In the second step, we propose ensemble-learning method with transfer learning where we train the base classifier with multiple classifiers used in the first step to create a model. For ensemble method we used Random Forest, and the meta classifiers included Bagging and AdaBoost.
The source dataset after passing through preprocessing and feature selection phase was trained using these individual classifiers and then with ensemble method. Trained model was then tested on target dataset to see which classifier achieved better accuracy values. The Individual Classifiers used in this study are descried below:

3.5.1 Naïve Bayes (NB)

The Naïve Bayes classifier lies on the Bayes rule of conditional probability. It considers all the attributes important and independent thus analyzes each attribute individually [2]. It is the most simple yet strong Bayesian Network model and can achieve great accuracy level.

3.5.2 J48

This technique is based on Quinlan's C4.5 decision tree algorithm which is said to be implemented in Weka in java and works on the algorithm developed by Ross Quinlan known as ID3 and has more features that can tackle the issues which were not dealt by ID3 [25].

3.5.3 Multi Layer Perceptron (MLP)

MLP is based on neural network and works on back propagation in order to train classifier. The Input layer and output layer consists of various hidden layers that include sigmoid function. The hidden layers contain biased neuron except output layer, which is nonlinear function of the input [25].

3.5.4 Bagging

The Bagging (Bootstrap Aggregating) constructs every ensemble member by taking different datasets and then the average is combined to make predictions. In bagging, the results of combined model give better performance than one single model. Bagging-based methods use bootstrapped replicas of training data in building different classifiers [21].

3.5.5 Boosting

Boosting is another ensemble method and Adaboost is considered a well-known algorithm of Boosting family. During each round of iteration, a new model is created while multiple iterations are performed with different example weight. Since the weight of incorrectly classified classes can increase, this overfitting will be given a greater weight in the subsequent iteration. In this manner the series of the classifier work well together, and voting brings them all together. [2].

3.5.6 Random Forest

Random Forest which is considered as an ensemble classifier approach. The algorithm of decision tree chooses the features randomly as the process of pruning is not carried out at each node of the tree. RF is considered as a fast classifiers and the one that can handle large number of input features/attributes [2].

3.6 Performance Metrics

The efficiency of prediction model is evaluated using certain evaluation criteria and are generated through confusion matrix, which includes Accuracy, Recall, Precision, and Area Under Receiver Operating Characteristics Curve (AUC-ROC), F-measure etc. However in this study AUC-ROC, one of the most widely used performance metric [9], is used. The effectiveness of machine learning models can be measured using these metrics [26].

3.6.1 Confusion Matrix

A particular table known as the confusion matrix is used to evaluate the performance of machine learning algorithms. Every column of the matrix depicts the instance belonging to the predicted class whereas every row shows the actual class instance or vice versa. By reporting the number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN), this matrix provides a concise representation of the testing algorithm's output [4].

Prediction outcome

		p	**n**	**total**
actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

3.6.2 Accuracy

The ratio of the correctly predicted instances to the total number of instances by the classifier is called accuracy. It measures the hit and miss of the classifier.

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

3.6.3 Precision

The proportion of correctly predicted positive instances by the classifier out of all the positively classified instances is labeled as precision.

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

3.6.4 Recall

The ratio of correctly classified positive instances to the actual number of positive instances for a particular class. It is also referred to as true positive rate or sensitivity and it counts the number of hits of the classifier for the class.

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

3.6.5 AUC-ROC

The probability curve provides a visual representation of the trade-off between the true positive rate and the false positive rate. This curve is called as ROC (Receiver Operating Characteristic). The classifier's ability to differentiate between positive and negative classes is measured by the Area Under Curve (AUC).

$$AUC = \frac{1 + TP - FP}{2}$$

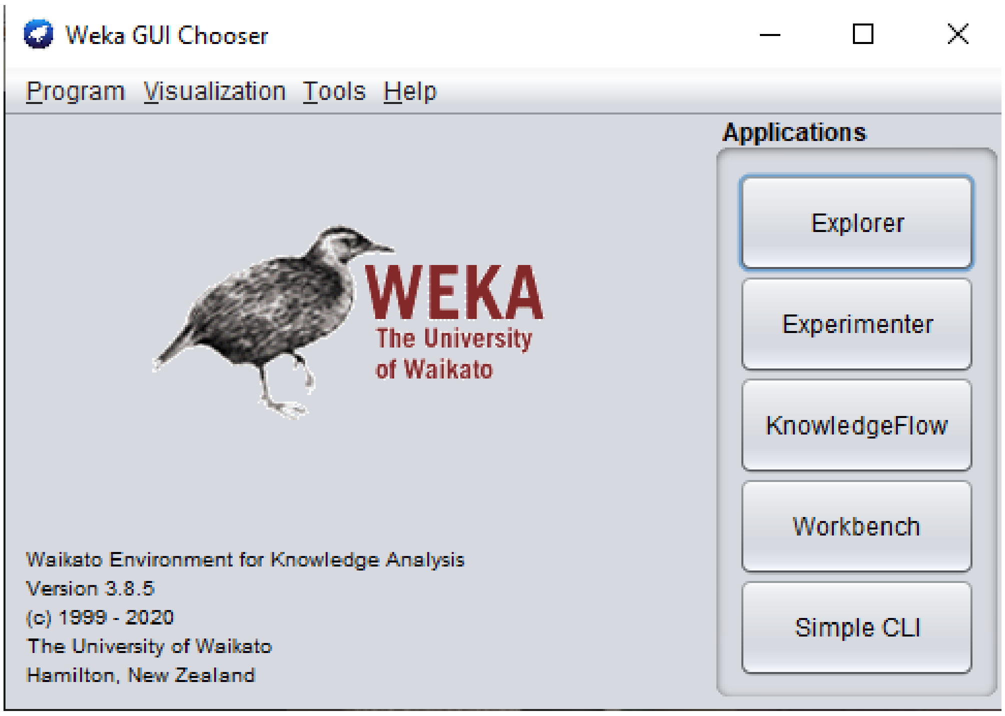
Since accuracy is regarded as a poor performance metric for imbalanced defect datasets, we use the most widely used metric to estimate the performance of each classifier, the area under the ROC curve - AUC as the performance metric in our study.

3.7 Tools

In this study experiments are performed using Weka, an open source data mining tool under GNU (General Public License), developed in Java language at University of Waikato, New Zealand. As a collection of machine learning algorithms with a variety of tools for data preparation, classification, regression, clustering, association rules mining, and visualization, this tool has been widely used in data mining studies.[27].

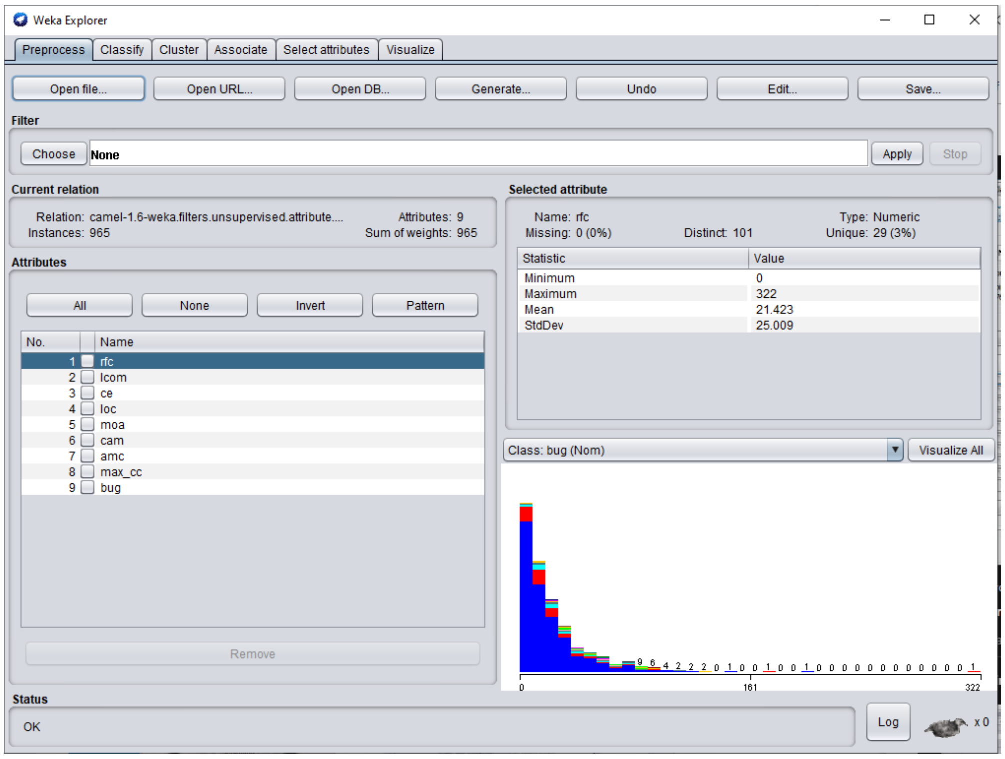
Weka is preferred usually due to the ease it provides because of its graphical user interface. It contains numerous algorithms from which any of choice can be selected, their parameters can be tuned and finally run on desired dataset. One dataset can have different models applied on it and the output that meets the requirement can be chosen. Most of the functions are found in-built therefore the researcher does not have to worry about learning languages and focus on his work alone.

Weka requires the input in the formatting of Attribute-Relational File Format while the filename should have the extension of .arff. The results/output achieved are easily readable and can also be visualized. Thus, the machine learning models can easily be developed quickly with the use of Weka. The figure 3.2 and 3.3 shows the interface of Weka.



Waikato Environment for Knowledge Analysis
Version 3.8.5
(c) 1999 - 2020
The University of Waikato
Hamilton, New Zealand

Figure 3.2: Weka Graphical User Interface



Filter

None

Current relation

Relation: camel-1.6-weka.filters.unsupervised.attribute.... Attributes: 9
Instances: 965 Sum of weights: 965

Attributes

No.	Name
1	rfc
2	lcom
3	ce
4	loc
5	moa
6	cam
7	amc
8	max_cc
9	bug

Selected attribute

Name: rfc Type: Numeric
Missing: 0 (0%) Distinct: 101 Unique: 29 (3%)

Statistic	Value
Minimum	0
Maximum	322
Mean	21.423
StdDev	25.009

Class: bug (Nom)

Status

OK

Figure 3.3: Weka Explorer

3.8 Summary

This section concludes and briefly highlights the major parts of the chapter. The chapter includes the theoretical approach we have used to make this model. We have discussed the creation of model that involves preprocessing, feature subset selection, transfer learning and ensemble learning method. We mentioned the data sets, all of their features and then after carrying out feature selection, the chosen features are stated. We discussed the classifiers used in the making of model and the performance metrics which will used to calculate the efficiency of model. In short we have reviewed the technical approach to make the software defect prediction model.

Chapter 4

Results and Discussion

4.1 Introduction

In this chapter the discussion about results given by the model that is created in chapter 3. All the implementation details are provided in the previous chapter. This chapter reveals the results of the proposed model and explains that which technique performed well on the dataset. It also describes the optimization of the techniques used in order to gain better performance and hence better result. The evaluation is done in the form of comparison between the results achieved by both the methods (individual and ensemble). This chapter concludes with the analysis of the achieved results which is shown in the form of table and also demonstrated on line chart and box plot graphs.

4.2 Results Overview

Table 3.1 shows all features present in CM1, MW1, PC1, PC2 dataset while Table 3.2 shows all features present in ant-1.7, camel-1.6, ivy-2.0, xalan-2.4. As the datasets had high dimensionality which could effect the end results so we conducted feature subset selection. Therefore, the Table 3.4 contains the selected features from NASA and PROMISE datasets. These features will provide us the basis to conduct transfer learning across projects.

The result tables below discusses the results for individual base classifiers and the ensemble based method where the defect prediction carried from the source project to the target project is represented by '$Source \rightarrow Target$'. For instance, '$CM2 \rightarrow PC1$' shows that CM2 is considered as the source project while PC1 is taken as the target project. The results of both the individual and ensemble based method for each dataset are shown in one table as it will be easier for comparison.

Since accuracy sometimes give biased results so we took AUC values as the performance metric instead of accuracy. The values marked bold are the highest ones among all the classifiers.

4.2.1 Results Discussion of NASA Dataset

Performance results of NASA datasets are revealed in the Table 4.1. We can easily evaluate the comparison of performance between the individual classifiers and the ensemble method. Apart from a few cases we have noticed that when we trained our model with source dataset and changed the target dataset to test the model, our results with ensemble method are significantly improved. However in two cases the individual classifier outperformed the ensemble method.

- When CM1 is kept as source dataset and target dataset chosen to be are MW1, PC1 and PC2, we see that the model performed well with Bagged NB and twice with Random Forest respectively. These results with PC2 as target dataset were quite compromising with individual classifiers but they got much better with ensemble method.

- With MW1 as source dataset and CM1, PC1, PC2 as target dataset, we see that the model performed well with NB as individual classifier, and twice with Random Forest respectively. Here NB gave better results to $MW1 \rightarrow CM1$, altough the results with RF and Bagged MLP are almost near.

- PC1 being the source dataset and CM1, MW1, PC2 as target dataset, Bagged J48, NB and Random Forest gave good results respectively. Here again NB as individual classifier outperformed when MW1 was used for training the model.

- When PC2 is kept as source dataset and target dataset chosen to be are CM1, MW1 and PC1, we see that the model performed well with Boosted J48 and twice with Bagged MLP respectively.

4.2.2 Results Discussion of PROMISE Dataset

To analyze the performance results of PROMISE datasets we refer the table 4.2. Here again we can see the comparison of performance between the individual classifiers and the ensemble method where we observe the model showed good results with ensemble method but deviation is also seen three times where individual classifier outperformed. Rest we see ensemble method taking the charge. To go into detail explanation we get:

- Ant 1.7 is taken as source dataset and target dataset chosen to be are Camel 1.6, Xalan 2.4, Ivy 2.0 where the highest values came from NB, J48 and with Random Forest respectively. Here we notice that when model is trained on Ant 1.7 dataset, it performed well with individual classifiers twice and once with ensemble Random Forest. This reveals that with some more feature tweaking and parameter tuning we can get better result with ensemble as well.

- With Xalan 2.4 as source dataset and Camel 1.6, Ant 1.7 and Ivy 2.0 as target dataset, we see that the model performed well with MLP as individual classifier, and twice with Bagged MLP respectively.

- Ivy 2.0 being the source dataset and Ant 1.7, Xalan 2.4, Camel 1.6 as target dataset showed that Bagged NB, Random Forest and Bagged MLP outperformed all other classifiers respectively.

- When Camel 1.6 is kept as source dataset and target dataset are Ant 1.7, Xalan 24 and Ivy 2.0, we see that the model performed well with Bagged NB all the times.

Table 4.1: Results of NASA Dataset

Sr. No	*Source → Target*	**Individual Classifiers**				**Ensemble Method**					
		NB	**J48**	**MLP**	**RF**	**Adaboost**			**Bagging**		
1.	$CM1 \rightarrow MW1$	0.708	0.500	0.704	0.748	0.564	0.653	0.643	0.775	0.618	0.730
2.	$CM1 \rightarrow PC1$	0.757	0.539	0.750	0.797	0.708	0.651	0.713	0.690	0.584	0.746
3.	$CM1 \rightarrow PC2$	0.210	0.491	0.145	0.855	0.486	0.827	0.350	0.404	0.584	0.849
4.	$MW1 \rightarrow CM1$	0.729	0.446	0.627	0.709	0.694	0.648	0.691	0.725	0.648	0.728
5.	$MW1 \rightarrow PC1$	0.767	0.485	0.536	0.796	0.594	0.668	0.686	0.763	0.673	0.697
6.	$MW1 \rightarrow PC2$	0.786	0.552	0.740	0.798	0.737	0.674	0.579	0.763	0.683	0.690
7.	$PC1 \rightarrow CM1$	0.674	0.580	0.702	0.695	0.583	0.699	0.664	0.689	0.729	0.694
8.	$PC1 \rightarrow MW1$	0.755	0.681	0.652	0.745	0.563	0.498	0.624	0.748	0.663	0.725
9.	$PC1 \rightarrow PC2$	0.813	0.399	0.762	0.835	0.768	0.681	0.735	0.793	0.725	0.766
10.	$PC2 \rightarrow CM1$	0.636	0.500	0.601	0.743	0.496	0.754	0.614	0.695	0.664	0.748
11.	$PC2 \rightarrow MW1$	0.559	0.500	0.503	0.652	0.500	0.570	0.530	0.565	0.591	0.762
12.	$PC2 \rightarrow PC1$	0.756	0.500	0.748	0.763	0.508	0.724	0.570	0.767	0.721	0.799

Table 4.2: Results of Promise Dataset

Sr. No	*Source* → *Target*	**Individual Classifiers**				**Ensemble Method**					
		NB	**J48**	**MLP**	**RF**	**Adaboost**			**Bagging**		
1.	$Ant-1.7 \to Camel-1.6$	0.901	0.814	0.492	0.588	0.554	0.562	0.567	0.599	0.590	0.606
2.	$Ant-1.7 \to Xalan-2.4$	0.695	0.820	0.754	0.752	0.661	0.807	0.717	0.725	0.739	0.777
3.	$Ant-1.7 \to Ivy-2.0$	0.750	0.740	0.647	0.798	0.740	0.797	0.762	0.784	0.794	0.789
4.	$Xalan-2.4 \to Camel-1.6$	0.628	0.653	0.705	0.583	0.507	0.563	0.588	0.576	0.603	0.619
5.	$Xalan-2.4 \to Ant-1.7$	0.799	0.762	0.812	0.800	0.604	0.683	0.738	0.782	0.772	0.813
6.	$Xalan-2.4 \to Ivy-2.0$	0.810	0.748	0.725	0.705	0.620	0.644	0.776	0.811	0.779	0.829
7.	$Camel-1.6 \to Ant-1.7$	0.725	0.626	0.734	0.652	0.667	0.594	0.656	0.736	0.645	0.688
8.	$Camel-1.6 \to Xalan-2.4$	0.698	0.603	0.700	0.649	0.620	0.577	0.638	0.708	0.642	0.663
9.	$Camel-1.6 \to Ivy-2.0$	0.722	0.634	0.689	0.718	0.670	0.658	0.655	0.746	0.649	0.692
10.	$Ivy-2.0 \to Ant-1.7$	0.769	0.737	0.712	0.776	0.702	0.707	0.669	0.777	0.746	0.768
11.	$Ivy-2.0 \to Xalan-2.4$	0.815	0.801	0.864	0.985	0.822	0.953	0.924	0.810	0.952	0.905
12.	$Ivy-2.0 \to Camel-1.6$	0.602	0.545	0.584	0.557	0.568	0.498	0.524	0.618	0.595	0.605

4.3 Result Analysis

Performance of machine learning model is dependent on so many factors. Our goal here is to check if optimizing the model using multiple classifiers works well. A comparative analysis of individual classifiers and ensemble based method shows that generally the use of multiple classifiers as in ensemble methods performed better as compared to the individual ones across all the datasets. However, our results indicate that minor deviations are there which are actually reveal different dimensions of a prediction model.

The varying nature of work of the selected classifiers were taken into consideration as Naive Bayes works on probability, J48 which is a trimmed version of C4.5 decision tree [28], MLP which is an artificial neural network, Random Forest that consists of decisions trees while using bagging and feature randomness during classification, Bagging and Adaboost are the ensemble learning methods. So these all classifiers perform different yet are strong to give us good insight of our model performance. Although with our currently used datasets, J48 and Adaboost has not performed very well whereas Bagged NB, Bagged MLP and Random Forest has given good results.

The Line chart plots, attained by all classifiers, of ROC-AUC scores are shown in the Figure 4.1 and 4.2. The x-axis shows the AUC values while the y-axis displays the iterations carried out by different datasets. The graph reveals the change in AUC values when a classifier is used to train each dataset and demonstrates their trends. For instance, Random Forest and bagging MLP classifiers performed well on multiple NASA datasets as shown in Figure 4.1 , while Bagging MLP and Naive Bayes classifiers achieved the highest ROC-AUC scores for Promise datasets as displayed in Figure 4.2.

Also, it makes it abundantly clear that there is no one predominant classifier, which may be due to the nature of datasets. At some point one classifier performed better while in some places other outperformed it. This may be due to the nature of classifier used to train or test a particular dataset or even the nature of dataset that was trained on the classifier and also the dataset that was tested with it. The variations are there, however, Random Forest and Bagged NB outperformed the most number of times.

In order to further review our results and outcome, we performed box-plot examination of the results obtained. The Box-plot review is considered as a non-parametric test which presents variation in the samples without doing any self assumption related to the statistical distribution of the data (Benjamini, 1988). Figure 4.3 and 4.4 demonstrate the results got from the analysis of box-plot for NASA and Promise datasets respectively. The diagrams of Box-plot yields the minimum, maximum, first quartile and third quartile values of a sample whereas the median value is shown by the center of the box-plot. The box-plot for NASA dataset as show in Fig. 4.3 reveals much variations as all the boxes are of varying heights, their medians are different too. and box-plot for Promise dataset as shown in Fig. 4.4 has some stable values.

4.4 Summary

The findings we get show that there was vulnerability in the performance of classifiers, as certain classifiers exhibited good performance in some datasets however compromised in others. Therefore, our outcomes suggest that detection of software defects should make use of ensembles as predictive models. Our results also indicate that minor deviations are there which can be corrected with few more optimizations such as tuning of parameters, class balancing and feature selection using different techniques.

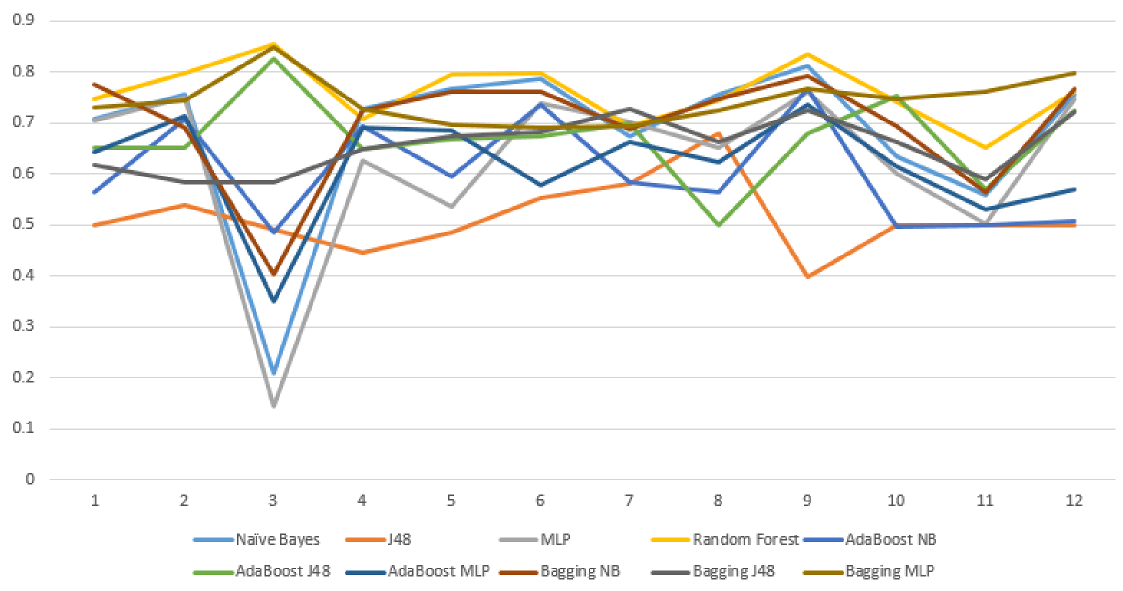


Figure 4.1: Line Chart for NASA Dataset

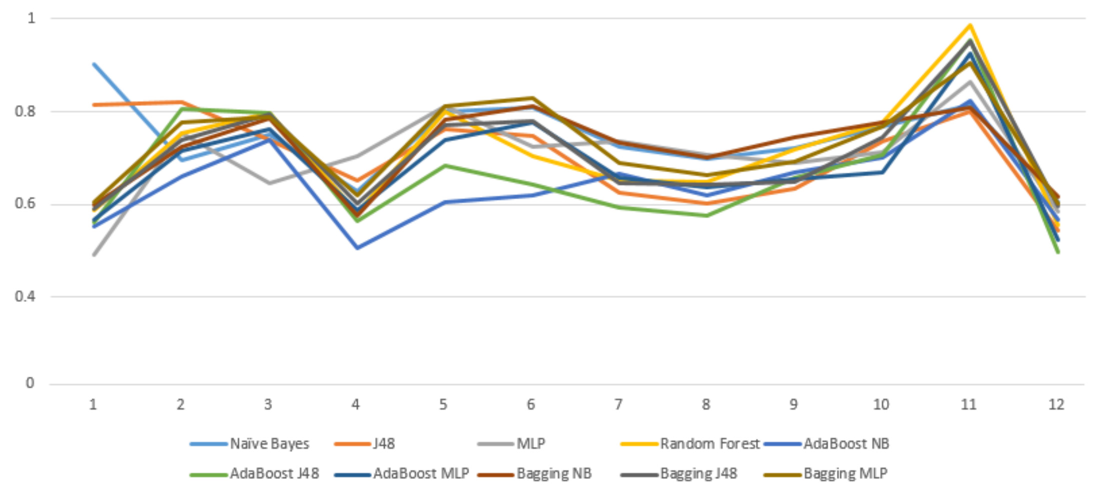


Figure 4.2: Line Chart for Promise Dataset

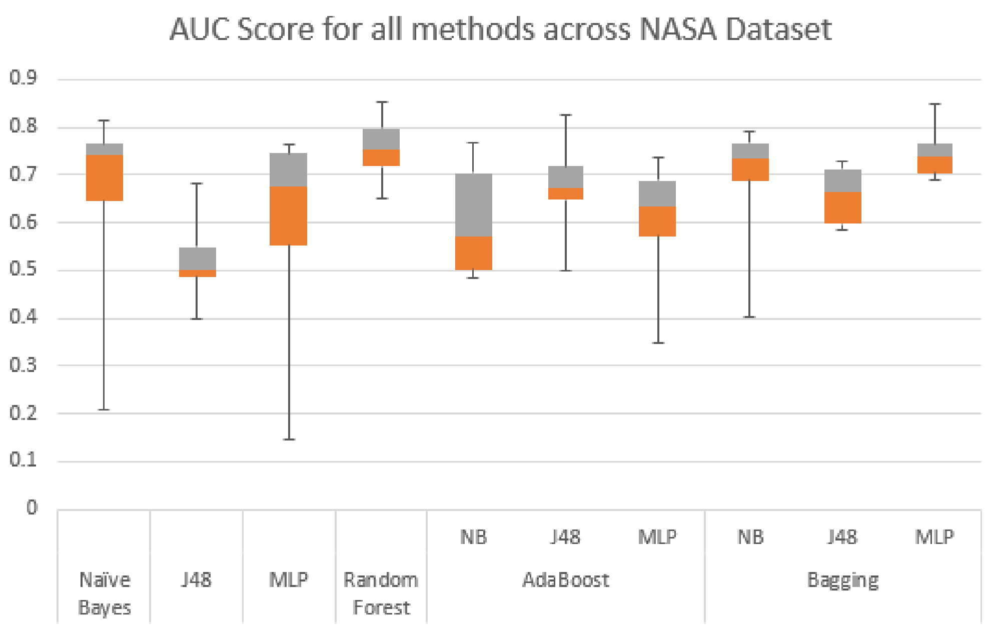


Figure 4.3: Boxplot for NASA Dataset

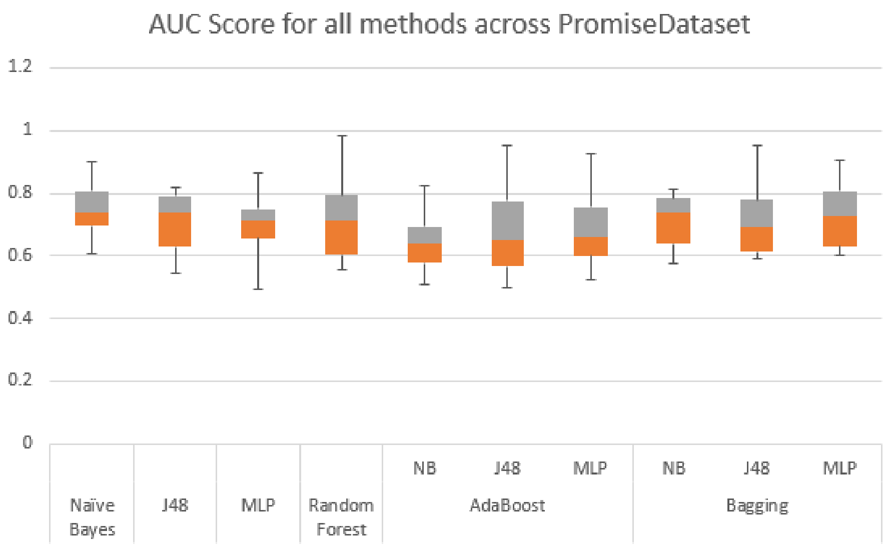


Figure 4.4: Boxplot for Promise Dataset

Chapter 5

Conclusion and Future Work

An overview of the whole study is provided in this chapter. It also describes the conclusion and gives an insight to future research work.

The major objective of this thesis is to prepare a machine learning model that will be used for software bug prediction in the development of a software. The main task these models perform include the prediction of bugs when the software happens to be in its initial stages rather than in the testing phase where ruling out bugs becomes very expensive and often results in chaos.
To address this objective the state-of-the-art traditional methods and machine learning algorithms were studied to discover out the best suitable results for the particular problem, out of which an amalgam of Transfer Learning and Ensemble Learning Method was selected.

5.1 Conclusion

The framework comprised of feature selection, transfer learning and classification through individual classifiers and then through ensemble methods, so that we can compare both ways. Four cleaned NASA datasets and four Promise datasets are taken to carry out the experiment. The afore-mentioned dataset was chosen for further comparisons due to the fact that the majority of related works utilized these datasets to evaluate the performance of their software bug prediction techniques.
For feature subset selection we used Best-First search while using CFS (correlation-based feature selection). It helped in analyzing the redundancy between different features and

evaluate the predictive ability of each individual feature. The model was developed in WEKA due to its flexibility, inbuilt functions and various supporting libraries to complete this task.

The findings show that the use of multiple classifier mostly gives good results than using individual classifier. So we compared the results by using individual NB, J48, MLP and ensemble methods that included Random Forest, Adaboost and Bagging where the latter two served as meta classifier with NB, J48 and MLP as their base classifier. ROC-AUC curve was used as the performance metric.

Evaluation is carried out by comparing the performance score of all classifiers used within the framework to see the high performing classifier. The results reflected better performance of ensemble methods as compared to all classifiers.

One of the major observations made during the development and analysis is that in order to get the best of results we need to have a fine tuned data set, which is itself a very tedious and important job to do. In creating the software bug prediction model using ML, one must know the core task is to prepare the dataset with good features for our model to be trained and optimized on and choosing excellent classifiers to increase the predictive ability of the model.

5.2 Future Work

Future work will involve investigation of other approaches of Machine Learning for the undertaken model. This endeavor will lead to better results in terms of generating defect prediction. Future research can be performed in several directions:

1. Parameters and their optimization have a huge influence on the performing ability of a model. Parameter tuning can be performed to check any improvement in performance.

2. Various combinations of Feature Selection and data sampling techniques can be investigated.

3. Datasets from different domains can be used in transfer learning to analyze the efficiency of model.

References

[1] Shruthi Puranik, Pranav Deshpande, and K. Chandrasekaran. A novel machine learning approach for bug prediction. *Procedia Computer Science*, 93:924–930, 2016. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2016.07.271. URL https://www.sciencedirect.com/science/article/pii/S1877050916315174. Proceedings of the 6th International Conference on Advances in Computing and Communications.

[2] Saiqa Aleem, Luiz Capretz, and Faheem Ahmed. Benchmarking machine learning techniques for software defect detection. *International Journal of Software Engineering & Applications*, 6:11–23, 05 2015. doi: 10.5121/ijsea.2015.6302.

[3] Santosh Rathore and Sandeep Kumar. Towards an ensemble based system for predicting the number of software faults. *Expert Systems with Applications*, 82, 04 2017. doi: 10.1016/j.eswa.2017.04.014.

[4] Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan, and Fatima Alsarayrah. Software bug prediction using machine learning approach. *International Journal of Advanced Computer Science and Applications*, 9, 01 2018. doi: 10.14569/IJACSA.2018.090212.

[5] Md. Razu Ahmed, Md. Asraf Ali, Nasim Ahmed, Md Fahad Zamal, and F.M. Shamrat. The impact of software fault prediction in real-world application: An automated approach for software engineering. 01 2020. doi: 10.1145/3379247.3379278.

[6] Syahana Nur'Ain Saharudin, Koh Tieng Wei, and Kew Si Na. Machine learning techniques for software bug prediction: A systematic review. *Journal of Computer*

Science, 16(11):1558–1569, Nov 2020. doi: 10.3844/jcssp.2020.1558.1569. URL https://thescipub.com/abstract/jcssp.2020.1558.1569.

[7] C. Manjula and Lilly Florence. Deep neural network based hybrid approach for software defect prediction using software metrics. *Cluster Computing*, 22, 07 2019. doi: 10.1007/s10586-018-1696-z.

[8] Lal M. Vashisht, V. and G. Sureshchandar. A framework for software defect prediction using neural networks. *Journal of Software Engineering and Applications*, 8:384–394, 2015. doi: http://dx.doi.org/10.4236/jsea.2015.88038.

[9] Md Fahimuzzman Sohan, Md Alamgir Kabir, Mostafijur Rahman, Touhid Bhuiyan, M. Ismail Jabiullah, and Amarachukwu Felix. *Prevalence of Machine Learning Techniques in Software Defect Prediction*, pages 257–269. 07 2020. ISBN 978-3-030-52855-3. doi: 10.1007/978-3-030-52856-0_20.

[10] Faseeha Matloob, Taher Ghazal, Nasser Taleb, Shabib Aftab, Munir Ahmad, Muhammad Khan, Sagheer Abbas, and Tariq Soomro. Software defect prediction using ensemble learning: A systematic literature review. *IEEE Access*, 9: 98754–98771, 07 2021. doi: 10.1109/ACCESS.2021.3095559.

[11] Ahmad Hassanpour, Pourya Farzi, Ali Tehrani, and Akbari Reza. Software defect prediction based on deep learning models: Performance study, 04 2020.

[12] Waleed Albattah Mohammad Nazrul Islam Khan Rehan Ullah Khan, Saleh Albahli. Software defect prediction via deep learning. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 9(5), March 2020. doi: 10.35940/ijitee.D1858.039520.

[13] Devi Akalya, K. Kannammal, and Surendiran B. A hybrid feature selection model for software fault prediction. *International Journal on Computational Science & Applications (IJCSA)*, 2:25–35, 04 2012.

[14] Faseeha Matloob, Shabib Aftab, and Ahmed Iqbal. A framework for software defect prediction using feature selection and ensemble learning techniques. *International Journal of Modern Education and Computer Science*, 11:14–20, 12 2019. doi: 10.5815/ijmecs.2019.12.02.

[15] Manu Banga and Abhay Bansal. Proposed software faults detection using hybrid approach. *Security and Privacy*, 01 2020. doi: 10.1002/spy2.103.

[16] Qiao Yu, Junyan Qian, Shujuan Jiang, Zhenhua Wu, and Gongjie Zhang. An empirical study on the effectiveness of feature selection for cross-project defect prediction. *IEEE Access*, PP:1–1, 01 2019. doi: 10.1109/ACCESS.2019.2895614.

[17] Dyana Ibrahim, Ghnemat Rawan, and Amjad Hudaib. Software defect prediction using feature selection and random forest algorithm. 10 2017. doi: 10.1109/ICTCS.2017.39.

[18] Aries Saifudin, Harco Leslie Hendric Spits Warnars, Benfano Soewito, F.L. Gaol, Edi Abdurachman, and Yaya Heryadi. Tackling imbalanced class on cross-project defect prediction using ensemble smote. volume 662, page 062011, 11 2019. doi: 10.1088/1757-899X/662/6/062011.

[19] Ishani Arora, Vivek Tetarwal, and Anju Saha. Open issues in software defect prediction. *Procedia Computer Science*, 46:906–912, 12 2015. doi: 10.1016/j.procs.2015.02.161.

[20] Ying Ma, Guangchun Luo, Xue Zeng, and Aiguo Chen. Transfer learning for cross-company software defect prediction. *Information and Software Technology*, 54:248–256, 03 2012. doi: 10.1016/j.infsof.2011.09.007.

[21] Shaojian Qiu, Lu Lu, Siyu Jiang, and Yang Guo. An investigation of imbalanced ensemble learning methods for cross-project defect prediction. *International Journal of Pattern Recognition and Artificial Intelligence*, 33, 01 2019. doi: 10.1142/S0218001419590377.

[22] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. Data quality: Some comments on the nasa software defect datasets. *Software Engineering, IEEE Transactions on*, 39:1208–1215, 09 2013. doi: 10.1109/TSE.2013.11.

[23] Martin Shepperd, Qinbao SOng, Zhongbin Sun, and Carolyn Mair. Nasa mdp software defects data sets, Mar 2018. URL `https://figshare.com/collections/NASA_MDP_Software_Defects_Data_Sets/4054940/1`.

[24] Deepti Aggarwal. Software Defect Prediction Dataset. 1 2021. doi: 10.6084/m9.figshare.13536506.v1. URL https://figshare.com/articles/dataset/Software_Defect_Prediction_Dataset/13536506.

[25] Wasiur Rhmann, Babita Pandey, Gufran Ansari, and Devendra Pandey. Software fault prediction based on change metrics using hybrid algorithms: An empirical study. *Journal of King Saud University - Computer and Information Sciences*, 32, 03 2019. doi: 10.1016/j.jksuci.2019.03.006.

[26] Ernest Ampomah, Zhiguang Qin, and Gabriel Nyame. Evaluation of tree-based ensemble machine learning models in predicting stock price direction of movement. *Information*, 11:332, 06 2020. doi: 10.3390/info11060332.

[27] Morgan Kaufmann. *Data Mining: Practical Machine Learning Tools and Techniques*, chapter The WEKA Workbench. Fourth edition edition, 2016.

[28] N.Sarav anaN and V.Gaya thri. Performance and classification evaluation of j48 algorithm and kendall's based j48 algorithm (knj48). *International Journal of Computer Trends and Technology*, 59:73–80, 05 2018. doi: 10.14445/22312803/IJCTT-V59P112.