SOFTWARE DEFECTS PREDICTION ANALYSIS USING MACHINE LEARNING ALGORITHMS

BY

Md. Muftain Ahmed Joy
ID: 0242310005103021

This Report Presented in Partial Fulfillment of the Requirements for the Degree of Masters of Science in Computer Science and Engineering

Supervised By
Mr. Narayan Ranjan Chakraborty
Associate Professor and Associate Head
Department of CSE
Daffodil International University

Co-Supervised By
Mr. Md. Sadekur Rahman
Associate Professor
Department of CSE
Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY

DHAKA, BANGLADESH

JULY 2024

APPROVAL

This Thesis titled "**Software Defects Prediction Analysis Using Machine Learning Algorithms**", submitted by **Md. Muftain Ahmed Joy,** ID No: 0242310005103021 to the Department of Computer Science and Engineering, Daffodil International University, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of M.Sc. in Computer Science and Engineering (MSc.) and approved as to its style and contents. The presentation has been held on 06-07-2024.

BOARD OF EXAMINERS

Dr. Sheak Rashed Haider Noori, PhD — **Chairman**
Professor and Head
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Dr. Md. Zahid Hasan, PhD — **Internal Examiner**
Associate Professor
Department of CSE
Faculty of Science & Information Technology
Daffodil International University

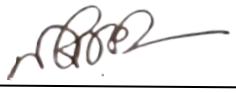
Dr. Arif Mahmud, PhD — **Internal Examiner**
Associate Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Dr. Md. Zulfiker Mahmud — **External Examiner**
Professor
Department of Computer Science and Engineering
Jagannath University

DECLARATION

I hereby declare that, this thesis has been done by me under the supervision of **Mr. Narayan Ranjan Chakraborty, Assistant Professor and Associate Head, Department of CSE** Daffodil International University. I also declare that neither this thesis nor any part of this thesis has been submitted elsewhere for award of any degree or diploma.

Supervised by:



Mr. Narayan Ranjan Chakraborty
Associate Professor and Associate Head
Department of CSE
Daffodil International University

Co-Supervised by:

Mr. Md. Sadekur Rahman
Assistant Professor
Department of CSE
Daffodil International University

Submitted by:

Md. Muftain Ahmed Joy
ID: 0242310005103021
Department of CSE
Daffodil International University

ACKNOWLEDGEMENT

First of all, I would like to express my gratefulness to the almighty Allah for blessing me to complete the final year thesis successfully.

I am really grateful and wish my profound indebtedness to **Narayan Ranjan Chakraborty**, Assistant Professor and Associate Head, Department of CSE Daffodil International University, Dhaka. Deep Knowledge & keen interest of my supervisor in the field of "Big data, Machine learning" to carry out this thesis. His endless patience ,scholarly guidance ,continual encouragement , constant and energetic supervision, constructive criticism , valuable advice ,reading many inferior draft and correcting them at all stage have made it possible to complete this thesis.

I would like to express my heartiest gratitude to **Dr. Sheak Rashed Haider Noori, Professor and Head, Department of CSE**, for his kind help to finish my thesis and also to other faculty members and the staff of CSE department of Daffodil International University.

Finally, I must acknowledge with due respect the constant support and passion of my parents.

ABSTRACT

Software defects are a significant concern in the software development industry, leading to increased costs, delays, and compromised quality. To address these challenges, our study focuses on utilizing machine learning algorithms for the early prediction of software defects. By analyzing historical defect data and relevant metrics, we aim to develop a predictive model that can identify potential defects before they occur in the final product. This proactive approach can help developers prioritize testing efforts, allocate resources efficiently, and improve overall software quality. In this research, we employ various machine learning algorithms, with a particular focus on the Random Forest algorithm, recognized for its effectiveness in classification tasks. Our model is trained and tested on multiple datasets, including those from NASA, to evaluate its performance in different scenarios. Key performance metrics such as accuracy, precision, recall, and F-measure are used to assess the model's effectiveness.

The findings of our study demonstrate that the Random Forest algorithm achieves superior performance in predicting software defects, with significant improvements in prediction accuracy and reliability. Our model not only identifies defect-prone areas in the code but also provides actionable insights for developers to address potential issues early in the development cycle. The proposed machine learning-based approach to defect prediction offers a robust tool for enhancing software quality assurance processes. By integrating this model into the software development lifecycle, organizations can reduce maintenance costs, accelerate development timelines, and deliver higher-quality software products.

Keywords: Software quality metrics, Software defects prediction, Machine learning algorithms, Code complexity, Historical defect data, Predictive models, Software reliability.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Introduction

High-quality software development significantly affects user satisfaction, operational efficiency, and corporate success. Software defects are inevitable but can lead to substantial financial and operational issues if not addressed promptly. This research employs machine learning to predict software defects early in the development process. This method promotes proactive quality assurance in software testing.

Software development is a complex process, including requirement analysis, design, coding, testing, and maintenance. Defects can arise at any stage, causing issues. Traditional defect detection relies on manual testing and code reviews, which, although effective, are time-consuming and costly. These methods often detect defects late in the development cycle, leading to expensive corrections and project delays.

A promising approach is the utilization of machine learning to improve the prediction of defects. By analyzing historical defect data and software metrics, machine learning algorithms can identify patterns. This predictive capability allows developers to focus on areas where errors are most likely to occur, optimizing resource usage and reducing development costs

.

I use machine learning techniques to examine how to predict software bugs. I test algorithms on several sets of data to find the best ones. Software measures like code complexity, lines of code, and defect records are used to train the models in the study.

Effective defect prediction has huge potential. Early defect detection prevents costly post-release corrections, saving money. It improves software dependability and performance, improving user happiness and market reception. Automating defect identification lets developers focus on innovation and feature creation rather than debugging and testing.

This research has the potential to change software development. Incorporating machine learning-based defect prediction into the development lifecycle can improve software development efficiency, cost, and reliability. This proactive strategy solves quality assurance

needs immediately and helps software projects succeed in the long run.

In conclusion, this machine learning-based software fault prediction research advances software engineering. I use data and predictive analytics to offer machine learning techniques that improve error detection, optimize development resources, and create high-quality software solutions.

1.2 Motivation

Software defects are a major concern in software development, leading to increased costs and time delays. Early detection of these defects can save significant resources and improve software quality. Machine learning offers a promising solution by automating the defect detection process. Our motivation is to create a model that can accurately predict defects, enabling developers to focus on preventive measures rather than reactive ones. By improving defect prediction, we aim to enhance the overall efficiency and reliability of software development processes.

1.3 Research Objective

The main goal of this research are as follows:

1. Identify which machine learning techniques work best for defect prediction.
2. Evaluate the performance of these algorithms based on various metrics, such as F-measure, accuracy, precision, and recall.
3. Recommend machine learning models that assist developers in early defect detection, ultimately lowering testing and maintenance expenses.

1.4 Research Questions

To guide our research, we focus on the following questions:

1. Which machine learning algorithms are most effective for predicting software defects?
2. What data preprocessing methods are required to enhance the accuracy of defect prediction?
3. How can we achieve the highest possible precision and recall in defect detection?
4. What are the challenges in generalizing the model across different software projects?

1.5 Report Layout

The report is divided into six chapters, each of which covers different aspects of my research:

Chapter 1 introduces the research topic, outlines the importance of software defect prediction, and presents the research questions, motivation, and objectives. Chapter 2 reviews existing work on software defect prediction, discussing various approaches, limitations, results, and methods used by other researchers. It also addresses the scope and challenges of my research. Chapter 3 details the methodology used in my research, including data collection procedures, data statistics, classifiers used, and implementation requirements. Chapter 4 shows the findings of my research, demonstrating the performance of several classifiers and the efficacy of my method of research. Chapter 5 discusses the significance and sustainability of my research, as well as its potential implications for the software development industry. Chapter 6 discusses the research findings and conclusions, as well as suggestions for future work. It also discusses the limitations of this particular study.

CHAPTER 2
BACKGROUND

2.1 Introduction

In the field of software development, ensuring high-quality software is crucial as it directly impacts user satisfaction and operational efficiency. However, predicting software defects early in the development cycle remains a significant challenge. To address this, we utilize machine learning algorithms to enhance defect prediction. Specifically, our approach involves applying various machine learning models to historical defect data to identify patterns and predict future defects. This method aims to assist developers in pinpointing potential issues early, thereby reducing the time and cost associated with extensive testing and maintenance.

2.2 Related Works

A number of research studies have helped us figure out how to predict software bugs. Here, we look at the different ways that experts have looked into to predict software bugs.

A model for predicting software bugs that uses several machine learning methods is described in the literature. The researchers evaluated eight algorithms on a NASA dataset that was open to the public. The algorithms included artificial neural networks, random forest, and SMOreg. The results show that the SMOreg classifier did the best overall, with low mistake rates in a number of statistical measures. Using a percentage split testing method also showed that the random tree strategy could work well. The study paper [1] demonstrates that using a variety of machine learning algorithms can reliably anticipate future software defects. This is critical for both increasing software quality and minimizing maintenance costs.

Machine learning methods can predict software bugs, as observed by Burcu Yalciner et al. [2]. Four NASA datasets were used to evaluate seven machine learning methods and find the best one for predicting defects. To determine how effectively the algorithms function, the study uses metrics like as accuracy, precision, recall, and the F-measure. The results indicate how effective Random Forests are at anticipating faults. The study's purpose was to help software businesses detect flaws early in the development process, saving money on testing and maintenance.

The study's authors [3], who conducted their research at the University of Maryland, used data from eight medium-sized information management systems that were constructed utilizing a sequential life cycle model and C++. Their findings demonstrate that metrics can reliably forecast a class's fault-proneness, particularly when the class is new or has undergone significant modifications. Interestingly, the Number of Children (NOC) metric indicated that having more children was associated with fewer errors, potentially due to extensive reuse. However, Lack of Cohesion in Methods (LCOM) was deemed unimportant as its definition rendered it non-variable. According to this study, these metrics can play a crucial role in identifying defects in object-oriented software development early on by serving as quality indicators.

The paper "Software Defect Identification Using Machine Learning Techniques" by Evren Ceylan et al.[4] develops a model using PCA for dimensionality reduction and machine learning methods like Decision Tree, MLP, and RBF. Using datasets from three major Turkish software companies, the study finds that the enhanced model significantly improves defect prediction accuracy, especially for major-priority defects, with RBF showing the best performance.

Using four NASA datasets, this study paper compares the performance of support vector machines (SVM) to eight statistical and machine learning models to see how well it can predict software modules that are prone to defects. The study paper[5] shows that across all datasets, SVM delivers high levels of performance. Overall, the results point to the potential benefit and applicability of SVM in software quality prediction frameworks, especially in lowering the likelihood of undetected defective modules. The report ends with a discussion of future work, including more research, the incorporation of other software measurements, and the use of SVM to predict other quality traits.

A methodology that might be adjusted for varying fault detection rates and overall accuracy was proposed by Lan Guo et al. [6]. The study assesses prediction accuracy using 10-fold cross-validation and compares random forests to other statistical techniques such as discriminant analysis and logistic regression.

The study report identifies Random Forest, Naive Bayes, and Support Vector Machines as the most widely used machine learning approaches for predicting software issues. Correlation-based feature selection is the most prevalent technique for reducing the number of measures. The NASA dataset is commonly utilized. Success is commonly assessed using metrics such as F-measure, AUC, recall, accuracy, and precision. Machine learning algorithms frequently yield precise predictions, with accuracy levels typically ranging from 75% to 85% and average AUC values falling between 0.7 and 0.83 [7].

Ruchika Malhotra investigated numerous methods for predicting software bugs, including C4.5, Naive Bayes, Multilayer Perceptrons and Random Forest. These approaches can handle big datasets and noisy data, but they are not always applicable because to the requirement to choose features and the occasional difficulties in interpretation. The review overall suggests that machine learning could potentially aid in predicting software bugs, yet more research is necessary, particularly with industrial datasets. This review was discussed in her paper [8].

In their study, the authors discussed class imbalance learning approaches in predicting software issues, highlighting that AdaBoost.NC exhibited superior performance compared to techniques such as Naive Bayes and Random Forest, particularly in terms of balance, AUC. These findings were detailed in the study [9].

Cagatay Catal et al. evaluated the effectiveness of class imbalance learning algorithms, finding that the AdaBoost.NC algorithm outperformed Naive Bayes and Random Forest in parameters like as balance, G-mean, and AUC. The authors propose a dynamic iteration of AdaBoost.NC that adapts its settings during training to enhance performance automatically. This approach demonstrates potential for achieving superior or comparable results without requiring manual parameter tuning, underscoring its relevance in addressing imbalanced data in software failure prediction [10].

The association between the intrinsic characteristics of open-source software systems and their fault-proneness was described in this study paper. The results [11] demonstrate that the fault prediction models' accuracy was not significantly affected by the selection of merely nine internal attributes. This implies that accurate predictions can be made with a reduced set of these properties. After examining four different categorization techniques, the study concludes that Random Forests performs the best overall. The findings demonstrate that

software quality assurance teams can save time and costs by concentrating on a smaller set of internal traits while still being very proficient in identifying which classes are most likely to have issues.

Ahmed Iqbal et al. assessed the prediction potential of several machine learning classification strategies on twelve NASA datasets. Random Forest, Naive Bayes, Support Vector Machine, and Multilayer Perceptron are a few of the strategies being tested. The findings show that differences in class size have no impact on accuracy or ROC assessments. However, when evaluating performance, accuracy, memory, the F-measure, and the Matthews Correlation Coefficient are superior. The outcomes can serve as a benchmark for contrasting several novel techniques for software bug prediction [12].

The effectiveness of Artificial Neural Networks (ANN) and Support Vector Machines (SVM) in predicting software errors were assessed using publicly available datasets. The authors [13] compared the two approaches based on accuracy, recall, and specificity. The results showed that while SVM excelled in recall, ANN was more precise and specific. They concluded that the choice of method should depend on the project's requirements and the importance of each evaluation criterion .

The Weighted Least Squares Twin Support Vector Machine (WLSTSVM) predicts software defects by considering the costs of misclassifying software components as faulty or functional. The authors [14] analyzed eight datasets from the PROMISE repository to compare WLSTSVM's efficacy with other techniques. The results show that WLSTSVM outperforms the competition across various performance metrics.

In a comparison of ensemble approaches and supervised machine learning for proactive software issue prediction, the authors used ten NASA datasets to evaluate the Random Forest (RF), Decision Tree (DS) & other models [15]. The study finds that RF consistently outperforms the other models in most cases. Additionally, the study discusses SMOTE oversampling as a potential solution for addressing class imbalance in datasets.

An empirical study was conducted to examine the efficacy of several data sampling approaches in predicting cross-project defects (CPDP). The objective was to evaluate the potential improvement of CPDP models and the resolution of the class mismatch problem by

using random data sampling. The study [16] used ensemble learning techniques, notably Random Forest and Gradient Boosting, along with twelve publicly available object-oriented project datasets. The results suggest that CPDP models utilizing random undersampling or no data sampling are much less predictive compared to those employing the SMOTE oversampling method. Furthermore, a notable statistical conclusion was found: after collecting data samples, the results of CPDP were similar to those of flaw prediction inside the project .

A feature weighting transfer naive Bayes (FWTNB) method and a minority over-sampling methodology known as TOMO is proposed to address the challenges of feature relevance and class mismatch in cross-project defect prediction (CPDP) [17]. The suggested TOMOFWTNB technique solves feature importance and class imbalance in CPDP by combining TOMO and FWTNB. Experiments on 11 public datasets show that TOMO outperforms traditional oversampling methods, FWTNB outperforms transfer naive Bayes, and TOMOFWTNB outperforms other cutting-edge CPDP methods in terms of MCC and G-Measure.

The findings of another paper demonstrate that while the classifiers produce comparable overall predictions, they identify distinct kinds of errors. Furthermore, there are variations in the predictability of the classifiers. Some predictors are more trustworthy than others. Based on the results, it is more likely to uncover more problems to use many classifiers than to use only one. The research underlines how crucial it is to look at the precise flaws that each classifier predicts, rather than merely looking at broad performance measures [18] .

The publication [19] provided instructions on how to develop a software fault proneness prediction program using an evolutionary method, specifically the genetic algorithm. Utilizing software data from source code files, the program determines the proportion of incorrect classes to send into the genetic process. Next, the optimal collection of software metrics that may be used to distinguish between modules that are broken and those that are not is found using the genetic algorithm. Use case, class, and sequence diagrams are examples of UML diagrams that illustrate how a program is designed. The approach can be used to forecast parts that are likely to have issues, according to experiments conducted with an open-source software project.

The performance of models that employ static code measurements to forecast defects was examined in the study paper. Examining five NASA datasets, the authors find that a large number of the modules are rather small. Based on the size of the modules, they separate the data into subsets and find that the subsets with larger modules perform better at predicting problems [20].

Six metrics are used to assess object-oriented design (OOD) and are based on Bunge's ontology and Weyuker's software measure properties. The metrics assess several crucial aspects of OOD, including coherence, inheritance, coupling, and class complexity. The authors [21] obtained real-world examples of these measures using data from two business OO projects, and they then solicited managers' interpretations of the data. Metrics can be used by managers and designers to assess the quality of an OO design, identify potential problems, schedule testing, and maintain the design. The metrics demonstrate how measurement may be leveraged to improve the OO software development process.

A new method was proposed for assessing software faults by analyzing prior bug reports[22]. It looks for duplicate defects, uses machine learning to estimate the duration of bug fixes, and evaluates a software component's risk value depending on the relevance of the bugs. The method was applied mostly to the "Networking: HTTP" component of the Mozilla Core dataset. The total risk ratings ranged from 27.4% to 84%. The results revealed a strong link with the risk values produced from the predicted problem fix time, but not with the risk values derived from the recurring bug records. The recommended approach can assist software finders in ranking software components that pose a risk.

The ability of three Weka data mining algorithms such as Reckontree, Simple Cart, and RandomTree to classify a collection of Indian news stories into groups was assessed. The collection contains 649 news stories divided into 7 categories. The outcomes demonstrated that the RandomTree method outperforms the other two. Every time, it accurately categorized all three sets of news articles. The RandomTree algorithm outperforms REPTree and Simple Cart for this categorization task, as demonstrated by the study [23].

Stefan P. Niculescu's demonstration focused on genetic algorithms and artificial neural networks, as well as their prospective applications in quantitative structure-activity relationship (QSAR) research. It discusses the general advantages and disadvantages of

neural network modeling, the importance of model validation, and the connections between neural networks, statistics, and expert systems. The most widely used neural network techniques for QSAR are then covered in the study, including Kohonen self-organizing maps, probabilistic, backpropagation, and Bayesian regularized neural networks. It displays the positive and negative aspects of each. It also goes over how model variable selection and naming in QSAR can be done using genetic algorithms. The study concludes by discussing the software tools that are available for implementing these techniques [24].

The effectiveness of the Random Forest (RF) method in predicting software classes that are likely to have faults using object-oriented (OO) metrics is examined in the study article [25]. The results demonstrate how well the RF technique predicts which classes in the open-source program JEdit will be incorrect. Its area under the ROC curve (AUC), recall, accuracy, and F-measure are all high. An overview of the RF algorithm and the research methodology for the empirical review are also included in the publication. Although the results are promising, the authors note that additional research of this kind is necessary to confirm that flaws may be identified using the RF model.

A comprehensive summary of various machine learning methodologies was provided in the study paper [26]. This paper explains numerous algorithms, including decision trees, Naive Bayes, support vector machines, principal component analysis, boosting, and neural networks. It also explores practical applications. The document offers pseudocode for numerous algorithms to demonstrate their use.

The potential application of Support Vector Machines (SVM) for identifying software classes prone to issues based on object-oriented (OO) metrics was examined in the research paper. This study used the publicly accessible NASA KC1 dataset to show how the SVM algorithm can effectively diagnose fault proneness. The statistics show that the SVM model beats specific OO metrics in terms of accuracy, sensitivity, and precision when identifying classes that are prone to generating errors. That one also assesses the predictive capabilities of the SVM model using ROC curves [27].

Machine learning for regression, forecasting, and prognostics use a variety of particular performance metrics. The authors of the paper[28] provided a summary of these requirements. The study examined many methods of defining measurements in the past and identified their

limitations. The article introduced a novel two-tier methodology that relies on the key elements of metrics: point distance, normalization, and aggregation. The typology consists of around 40 commonly used primary metrics, each of which may be defined using a basic mathematical formula.

Another study paper investigates two software complexity measurements: coupling between objects and weighted methods per class. Using data from the NASA KC1 library, the study explored if they follow a power-law distribution. The results reveal that neither CBO nor WMC exhibit scale-free features, contrary to findings in prior studies on open-source software. The authors argue that this discrepancy may be due to the relatively small dataset and call for further research to better understand the global aspects of software systems like "small world" and "scale-free" properties [29].

N. Gayatri et al. [30] created a new feature selection approach using Decision Tree Induction to identify critical software measures for forecasting challenges. This approach chooses qualities that show up in the decision rules provided by classifiers such as J48, CART, and BFTree. The 18 different models are trained with this new collection of features, and the results are compared with those obtained by using SVM and RELIEF to choose features. From a ROC, MAE, and RMSE point of view, the suggested technique does better than the other two for most classifiers. The success of categorization is substantially greater with the new collection of features than with the entire set of 94 metrics. It has been proved that this technique is easier to understand and interpret than the others.

2.3 Scope of the problem

This research aims to develop a model that predicts software defects using machine learning algorithms, providing developers with an early-warning tool. However, this task is challenging due to the complexity and imbalance of defect data, where different defects often exhibit overlapping characteristics. Additionally, the variability in coding standards and practices across projects makes it difficult to generalize the model. Despite these challenges, our initial experiments with a Random Forest classifier show promising results, indicating high accuracy and the potential for effective defect prediction.

2.4 Challenges

The primary challenge in developing our software defect prediction model is achieving high precision and accuracy. I need a large, high-quality dataset with comprehensive defect information if I want to get the improved performance of this model. One challenge is the variance in coding standards and procedures among many software projects, which influences model generalization. Furthermore, challenge is differentiating between similar types of defects given overlapping characteristics. This model needs to adapt to both common and unique defect patterns to be effective. Nonetheless, my preliminary results are encouraging, suggesting that with further refinement, this model can significantly aid in early defect detection.

CHAPTER 3
RESEARCH METHODOLOGY

3.1 Introduction

This section describes the research methods used in my machine learning-based software defect prediction study. For this research, it offers thorough data and practical strategies. This involves using my methodologies precisely, which adds to the validity of the information I gather and the research I do. I will show you how to efficiently classify and forecast defects in software using these methods. Here, the research techniques will be discussed.

3.2 Research Subject and Instrumentation

This research aims to develop a more reliable approach to predicting defects in software systems. To achieve this, I analyzed various software project datasets to identify patterns that could indicate the presence of defects. The software industry faces significant challenges in maintaining high-quality code, particularly in large and complex projects. Developers and project managers often struggle to detect potential bugs at the earlier stages in the development cycle, which can lead to increased costs and time delays. Accurately predicting defects can help in prioritizing testing efforts, improving software quality, and reducing maintenance costs. Using historical data, I applied machine learning algorithms, concentrating on categorization models to forecast the probability of faults. These models have the ability to remove unnecessary features and find important relationships between the characteristics of the code and the incidence of defects. The aim of this research is to evaluate the effectiveness of several machine learning approaches for defect prediction and to compare them with traditional techniques to identify the best approach.

3.3 Data Collection Procedure

Collecting relevant and high-quality data is essential for this study. I utilized open-source software project datasets, which include various metrics and attributes related to software development, such as lines of code, complexity measures, and historical defect data. I gathered approximately 4500 records from several projects, categorized into defective and non-defective instances. All collected data were formatted consistently.

3.3.1 Datasets

The datasets used in this research can be obtained from the public PROMISE repository and are presented in Table 3.2. The number of instances in each dataset varies. Data set KC1 contains the most cases overall. In terms of overall number of occurrences, CM1 contains the least amount of data.

Various forms of data were utilized to demonstrate how data size influences accuracy. Table 3.2 provides a thorough description of each dataset, including the language, attributes, number of occurrences, proportion of defective modules, and description. Each dataset contains the same 22 attributes. Table 3.1 defines the attributes.

TABLE 3.1: ATTRIBUTE DEFINITION

Type	**Metric**	**Definition**
McCabe	loc	Line count of code
	v(g)	Cyclomatic complexity
	ev(g)	Essential complexity
	iv(g)	Design complexity
Derived Halstead	n	Total operators + operands
	v	Volume
	l	Program length
	d	Difficulty
	i	Intelligence
	e	Effort
	b	Number of delivered bugs
	t	Time estimator
Basic Halstead	lOCode	Line count
	lOComment	Count of lines of comments
	lOBlank	Count of blank lines
	lOCodeAndComment	Count of lines containing both code and comments.
	uniq Op	Count of unique operators
	uniq Opnd	Count of unique operands
	total Op	Count of total operators
	total Opnd	Count of total operands
	branchCount	Count of branch counts
Class	**defects** (target value)	{false,true} % module has/has not one or more reported defects

TABLE 3.2: DATASET PROPERTIES

Type	Project	Language	Number of Attributes	Number of Instances	% of Defective Modules	Description
Object oriented	KC1	C++	22	2109	15.4	The computer language C++ was used to make KC1. It uses storage management to get ground info and deal with it.
	KC2	Java	22	522	20.5	The KC2 system is designed for processing science data which was developed by multiple developers as an extension of the KC1 project. A limited number of third-party KC1 software libraries were used in this implementation.
Procedural	CM1	C	22	498	9.7	NASA's CM1 spacecraft instrument is written in the programming language C.
	PC1	C	22	1109	6.9	This flight software is designed for earth-orbiting satellites.

3.4 Machine Learning Techniques

I have predicted software system defects using a variety of machine learning techniques. Numerous learner types are covered by the methodologies outlined, including Tree-Based Learners, Ensemble Learners, Bayesian Learners, Neural Networks, Support Vector Machines, and Distance-Based approaches.

Out of these several categories, I have chosen eight machine learning techniques to accurately assess defects in software. Figure 1 displays the algorithms used, along with their corresponding categories. Below is a list of each algorithm.

In order to assess the effectiveness of each learning approach, I utilized a 3-fold Cross-Validation (CV) model.

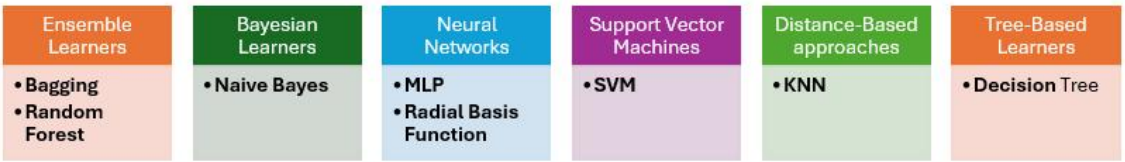


Fig. 3.4.1. Classification of ML Techniques

3.4.1 Ensemble Learners

• **Bagging:** This algorithm was introduced by Leo Breiman. Another name for it is Bootstrap Aggregation, and it falls under the ensemble approach category. Using this process, N subsamples of the training sample's data are created and used to train the predictive model.Subsamples are chosen using a random replacement procedure. As a result, the final model is a composite of multiple models.

• **Random Forest:** The learning algorithm is based on ensemble tree learning.Another name for Random Forest is Random Decision Forest. It makes predictions across individual trees and chooses the best vote among all projected classes across trees to improve generalisation accuracy and decrease overfitting. It is also the most flexible and intuitive method for classification and regression.

3.4.2 Bayesian Learner

• **Naive Bayes:** Naive Bayes is a probabilistic classifier that is simplistic in nature and applies Bayes' theorem while considering strong (naive) independence across the features. It is frequently employed in classification duties as a result of its efficiency and simplicity. It delivers the result with the highest probability after calculating the likelihood of each feature happening in each class.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \quad (1)$$

3.4.3 Neural Networks

• **MLP (Multilayer Perceptron):**A forwarder artificial neural network with many layers of nodes, each fully connected to the next, is called a Multilayer Perceptron (MLP). It is a powerful algorithm that can find complex patterns in data and is useful for supervised learning tasks like regression and classification.

• **Radial Basis Function (RBF):** The neural network architecture known as a Radial Basis Function (RBF) uses radial basis functions as activation functions. They are frequently used for classification and function approximation applications. RBF networks are made up of an

output layer that is utilised for tasks like regression or classification, and a hidden layer where each neuron determines the distance from its input to a centre.

3.4.4 Support Vector Machine

• **SVM (Support Vector Machine)**: The acronym SVM represents "Support Vector Machine." A supervised machine learning technique that is useful for both regression and classification problems is the SVM. The technique finds the hyperplane in the feature space that divides the classes and keeps the gap between them as large as feasible. Support Vector Machines (SVMs) are highly effective in multidimensional spaces. They are also adaptable since they can handle decision constraints that aren't straight lines by utilising various kernel functions.

3.4.5 Distance-Based Methods

• **KNN (K-Nearest Neighbors):** "K-Nearest Neighbours," or KNN, is a machine learning technique that is straightforward and simple to comprehend. It is applicable to tasks involving both regression and classification. The programme searches the feature space for the K data points that most closely resemble a question point in order to complete its objective. Next, in order to create predictions, it utilises the labels (for classification) or numbers (for regression) of the points that are closest to the target. Because KNN is non-parametric and lazy, it does not create a biassed function using the training set of data. Rather, the training cases are stored and retrieved at a later time.

3.4.6 Tree-Based Learner

• **Decision tree algorithm:** A decision tree is a widely used technique in supervised learning that is employed for tasks such as regression and classification. Recursive partitioning is a technique used to divide the feature space into subsets by considering the values of the features that are provided. This creates a hierarchical structure like a tree, where inside nodes make decisions based on features, while external nodes display class labels or numerical values. Decision trees are valuable in several scenarios due to their simplicity in comprehension and ability to process both numerical and categorical data.

3.5 Data Preprocessing

Effective data preprocessing is crucial for building robust and accurate machine learning models. This section outlines the steps taken to prepare the datasets for model training and evaluation.

3.5.1 Handling Missing Values

Missing values in the datasets were addressed using the following strategies:

Imputation: The mean of the feature was used to fill in missing number values, and the mode was used to fill in missing categorical values.

3.5.2 Standardization

The model's performance was ensured by assigning a mean of 0 and a standard deviation of 1 to all numerical features.

3.5.3 Handling Imbalanced Data

The distribution of classes in the data sets was frequently not uniform. This was resolved by more evenly distributing the classes through the use of oversampling techniques like SMOTE (Synthetic Minority Over-sampling Technique).

3.5.4 Hyper-parameter Tuning

Hyperparameter tuning was done to make the model's results better. It's possible to find the best set of hyper-parameters with tools like Grid Search and Random Search.

3.6 Statistical Analysis

This section provides a thorough statistical overview of the machine learning models utilized in predicting software defects. The main objective is to assess the accuracy, reliability, and robustness of these models in predicting software system faults.

3.6.1 Descriptive Statistics

First, I provide an overview of the most important descriptive statistics from the datasets that were used to train and test the models.

• **Mean and Median:** The average and middle values of the features.

• **Standard Deviation (SD):** the measurement of the features' degree of variation or dispersion.
• **Range:** The difference between the maximum and minimum values.

3.6.2 Model Evaluation Metrics

Accuracy, precision, recall, and F-measure are analysed to evaluate learning algorithms. The confusion matrix, also known as an error matrix, provides an overview of the classification prediction results and is used to evaluate the effectiveness of each technique. When it comes to multi-output classification problems, model evaluation is crucial.
True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values are included in the confusion matrix.

• **Positive (P):** Observation is positive (e.g., defective).
• **Negative (N):** Observation doesn't indicate a defect.
• **True Positive (TP):** The model estimates and test results match.
• **False Negative (FN):** The model estimates false yet the test data is true.
• **True Negative (TN):** The model estimates false and test data is false.
• **False Positive (FP):** The model predicts true but the test data is false.

Accuracy: The ratio of correctly predicted instances to the total instances. The following equation provides the accuracy, often known as the classification rate:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (2)$$

Precision: The ratio of correctly predicted positive observations to the total predicted positives.

$$\text{Precision} = \frac{TP}{TP+FP} \quad (3)$$

Recall (Sensitivity): The ratio of correctly predicted positive observations to all observations in the actual class.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (4)$$

F1 Score: The harmonic mean of precision and recall, providing a single metric that balances both concerns.

$$\text{F1 Score} = \frac{2 * Precision * Recall}{Precision + Recall} \quad (5)$$

AUC-ROC: A measure of the model's ability to distinguish between classes.

Confusion Matrix: A table displaying a classification model's true positive, true negative, false positive, and false negative numbers.

3.6.3 Cross-Validation

To ensure the robustness of the models, I perform k-fold cross-validation:

k-Fold Cross-Validation: The datasets are divided into k subsets. Using a different validation set and the remaining k-1 subsets as training sets, the model is trained and verified k times. In evaluating model performance and stability over data splits, this is helpful.

3.6.4 Model Comparison

Comparing different machine learning models to identify the best performing one for software defect prediction:

Visualization: Using box plots, ROC curves, and training vs validation accuracy & loss graph to visually compare model performance.

Ranking Models: Based on performance metrics and statistical tests, we rank the models to determine the most effective one for this prediction task.

3.7 Proposed Methodology

In this research, I present a model that uses information gathered from the NASA public repository to discover and diagnose software problems. I experimented extensively with several machine learning techniques in order to create an efficient learning system for fault prediction. To provide reliable model validation, I used K-fold Cross-Validation (CV) for every learning method. The datasets were first preprocessed in order to accommodate missing data, encode categorical variables, and apply SMOTE (Synthetic Minority Over-sampling Technique) to balance the datasets. The data were then normalised and divided into test and training sets. I trained and evaluated every model after identifying the features and target properties.

I used hyper-parameter tuning to improve performance. In order to fully comprehend the models' performance, I assessed the models using a variety of measures, including accuracy, precision, recall, and F1 score. I also looked at the confusion matrix, ROC curve, validation loss, and validation accuracy. The model demonstrating the best evaluation metrics was finalized for predicting software defects.

3.7.1 Proposed Model Workflow

In order to effectively predict the software defects, several things need to be done. Here is Fig. 3.7.1, which shows the flowchart that shows the suggested model workflow for this work..

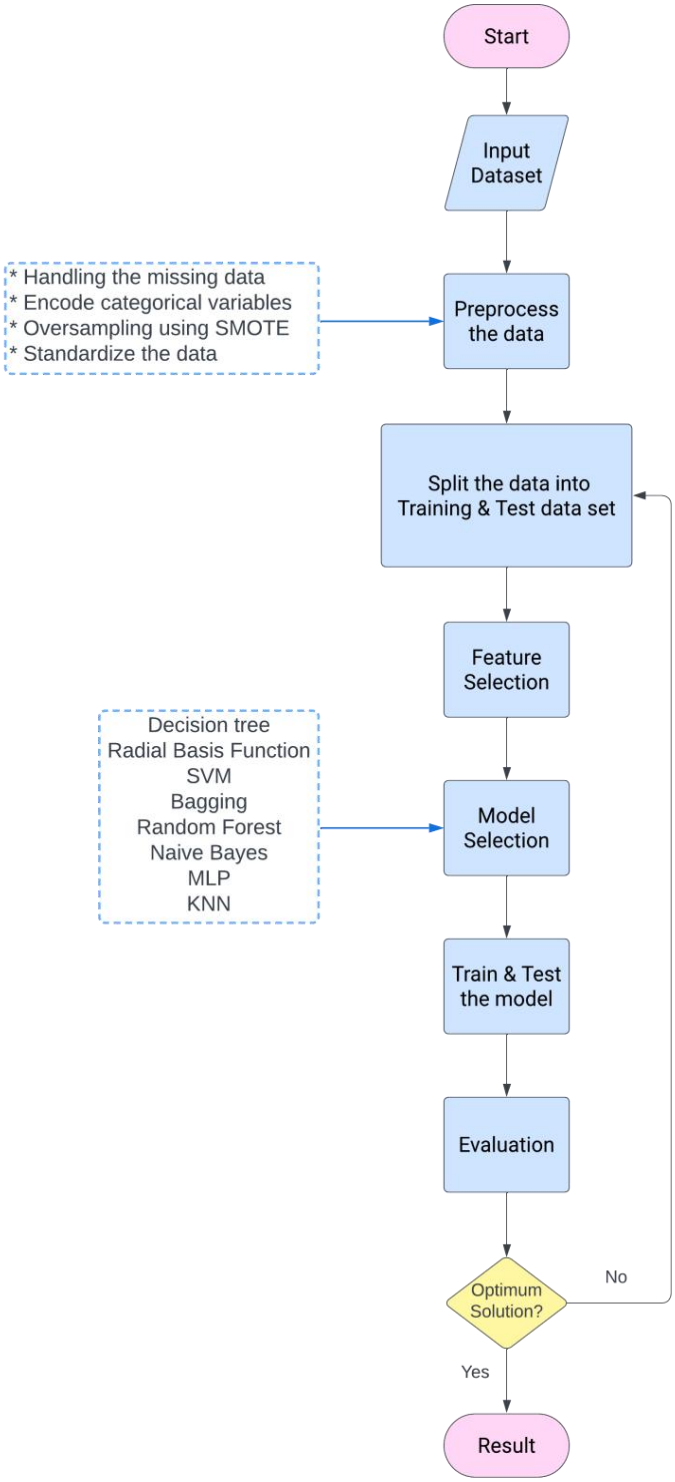


Fig. 3.7.1. Proposed Model Workflow

3.8 Implementation Requirements

To employ machine learning for software defect prediction, particular hardware and software prerequisites are necessary.

Hardware Requirements:

• A running system that is at least Windows 7.

• A hard drive, either portable or built-in, with at least 300 GB of space.

• At least 2 GB of RAM is needed to process info quickly.

Software Requirements:

• Installation of any web browser.

Developing Tools:

• For machine learning, Python with the TensorFlow library is utilized.

• Jupyter Notebook (Google Colab) for interactive development and documentation.

• Notepad++ for code editing, providing a more feature-rich environment compared to basic text editors.

CHAPTER 4
EXPERIMENTAL RESULTS AND DISCUSSION

4.1 Introduction

This chapter will cover the findings of this project's studies on software bug prediction as well as a descriptive analysis of the study data I used. It's important to look into and try a lot of different theories and methods to get good results.

4.2 Experimental Setup

The experimental setup involved several steps to ensure comprehensive evaluation and comparison of various machine learning algorithms:

1. **Data Preprocessing**: Handling missing data, encoding categorical variables, oversampling with SMOTE, and standardizing features.
2. **Train-Test Split**: The datasets need to be split into training and test sets using the conventional 70-30 split method.
3. **Model Selection**: Evaluating several machine learning algorithms, including Random Forest, Bagging, and other methods.
4. **K-fold Cross-Validation**: Using k-fold cross-validation for robust model validation.
5. **Hyper-parameter Tuning**: Fine-tuning model parameters to optimize performance.
6. **Evaluation Metrics**: evaluating models with respect to many criteria, including ROC curve, F1 score, accuracy, precision, recall, and confusion matrix.

4.3 Experimental Results & Analysis

In this study, I developed and evaluated eight different models on four distinct datasets to predict software defects. The evaluation of these models utilized a combination of metrics, though some studies rely on just one metric. My primary evaluation metrics included accuracy, graphical analysis, and confusion matrices.

TABLE 4.1: RESULTS FOR EACH DATASETS

Dataset	Metrics	Decision Tree	Radial Basis Function	Support Vector Machine	Bagging	Random Forest	Naive Bayes	MLP (Multilayer Perceptron)	KNN
CM1	Accuracy	86.66%	78.33%	92.22%	**93.33%**	92.78%	65.00%	76.67%	78.89%
	Precision	0.819	0.742	**0.880**	0.875	0.873	0.714	0.733	0.691
	Recall	0.917	0.821	0.964	**1.000**	0.988	0.416	0.785	0.988
	F1 Score	0.865	0.779	0.920	**0.933**	0.927	0.526	0.758	0.813
KC1	Accuracy	87.11%	75.21%	85.01%	90.89%	**91.73%**	64.28%	76.33%	83.47%
	Precision	0.895	0.739	0.840	**0.923**	0.902	0.800	0.744	0.807
	Recall	0.845	0.791	0.870	0.895	**0.939**	0.396	0.812	0.887
	F1 Score	0.869	0.764	0.855	0.909	**0.920**	0.530	0.777	0.845
KC2	Accuracy	86.74%	80.12%	85.54%	**89.15%**	86.14%	66.26%	78.91%	82.53%
	Precision	0.855	0.791	0.844	**0.886**	0.831	0.875	0.793	0.800
	Recall	0.895	0.837	0.884	0.907	**0.919**	0.407	0.802	0.884
	F1 Score	0.875	0.813	0.864	**0.897**	0.873	0.555	0.798	0.839
PC1	Accuracy	92.97%	85.23%	96.12%	95.15%	**96.85%**	60.04%	85.47%	90.55%
	Precision	0.900	0.807	0.952	0.947	**0.962**	0.777	0.816	0.845
	Recall	0.966	0.922	0.971	0.956	0.975	0.273	0.912	**0.990**
	F1 Score	0.932	0.861	0.961	0.951	**0.968**	0.404	0.861	0.912

The best classification results for each dataset and machine learning algorithm are highlighted in bold letters.

To predict the results, a variety of machine learning algorithms were applied to four different datasets. Following an in-depth analysis of Dataset 1 (CM1), it was discovered that Bagging had the highest accuracy, precision, recall, and F1 score among all approaches. Random Forest produced the highest accuracy and F1 score for Dataset 2 (kc1), although Bagging produced the optimum precision. In dataset 3 (KC2), Bagging had the highest accuracy and precision scores, outperforming Random Forest in recall. Finally, Dataset 4 (PC1) showed that KNN had the best recall, whereas Random Forest had the best accuracy, precision, and F1 score.

These results illustrate the necessity of selecting an appropriate algorithm.

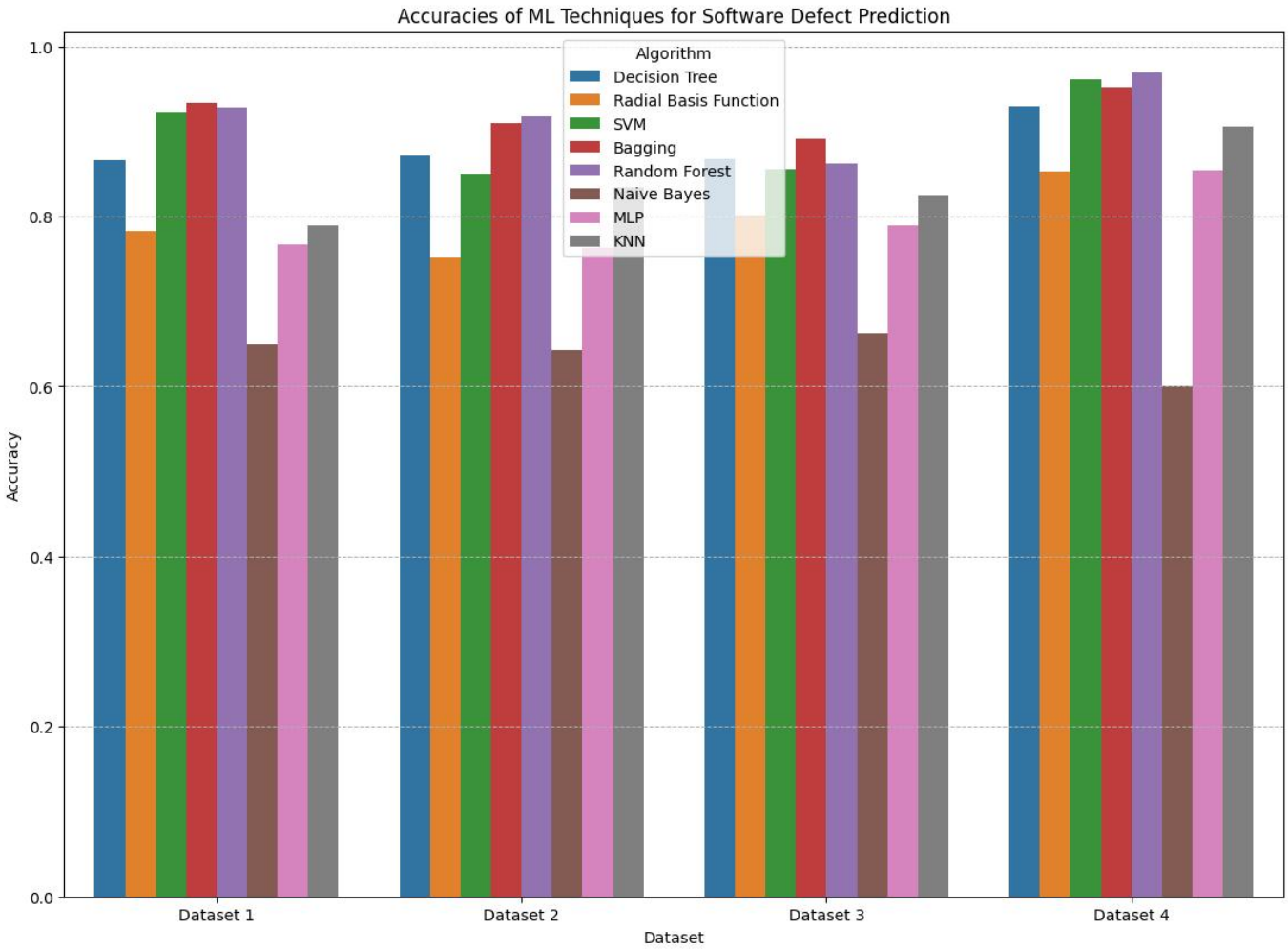


Fig. 4.3.1. Accuracies of ML Techniques for Software Defect Prediction

TABLE 4.2: AUC & PR AUC FOR EACH DATASETS

Dataset	Metrics	Decision Tree	Radial Basis Function	Support Vector Machine	Bagging	Random Forest	Naive Bayes	MLP (Multilayer Perceptron)	KNN
CM1	AUC	0.869792	0.857391	0.966270	0.975818	0.987971	0.715898	0.821429	0.945623
	PR AUC	0.789775	0.785111	0.920320	0.953815	0.985220	0.677587	0.801054	0.902713
KC1	AUC	0.869001	0.831654	0.905740	0.961197	0.967123	0.793510	0.830469	0.904103
	PR AUC	0.839633	0.817865	0.905468	0.955395	0.966999	0.771666	0.826874	0.872000
KC2	AUC	0.869767	0.853052	0.894186	0.950218	0.947238	0.826890	0.874273	0.898038
	PR AUC	0.828986	0.875085	0.867392	0.952742	0.855566	0.961502	0.901237	0.881850
PC1	AUC	0.930042	0.912136	0.991487	0.984955	0.992273	0.704350	0.907176	0.976818
	PR AUC	0.886217	0.871699	0.987550	0.980479	0.990616	0.696071	0.890486	0.960761

This study employed 20 iterations (epochs). The figures below illustrate the correlation between training and validation accuracy, as well as training and validation loss for each dataset.

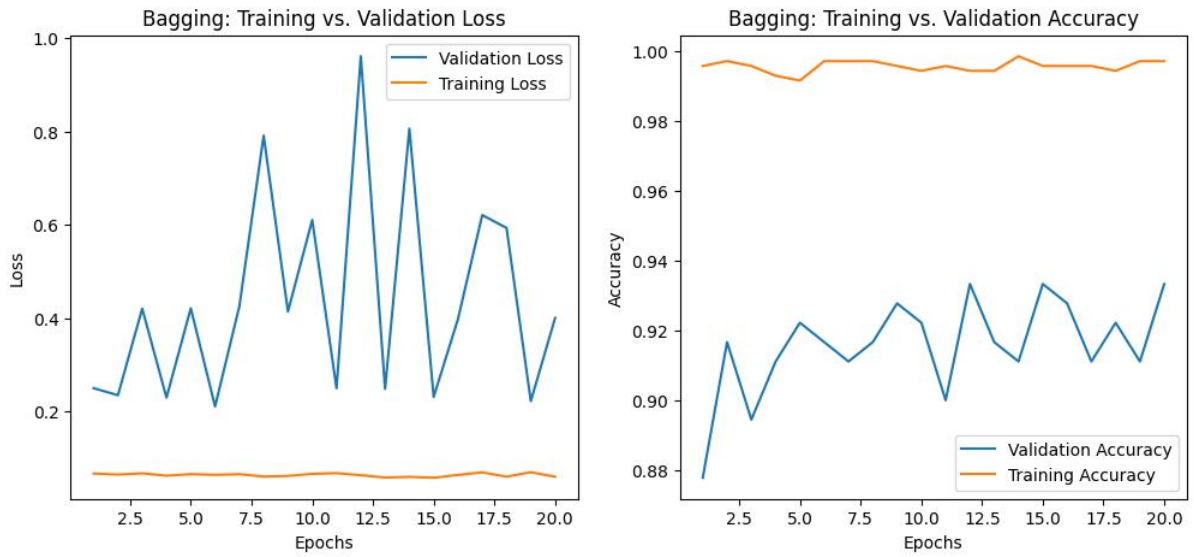


Fig. 4.3.2. Training vs Validation Accuracy & Loss (Bagging in CM1)

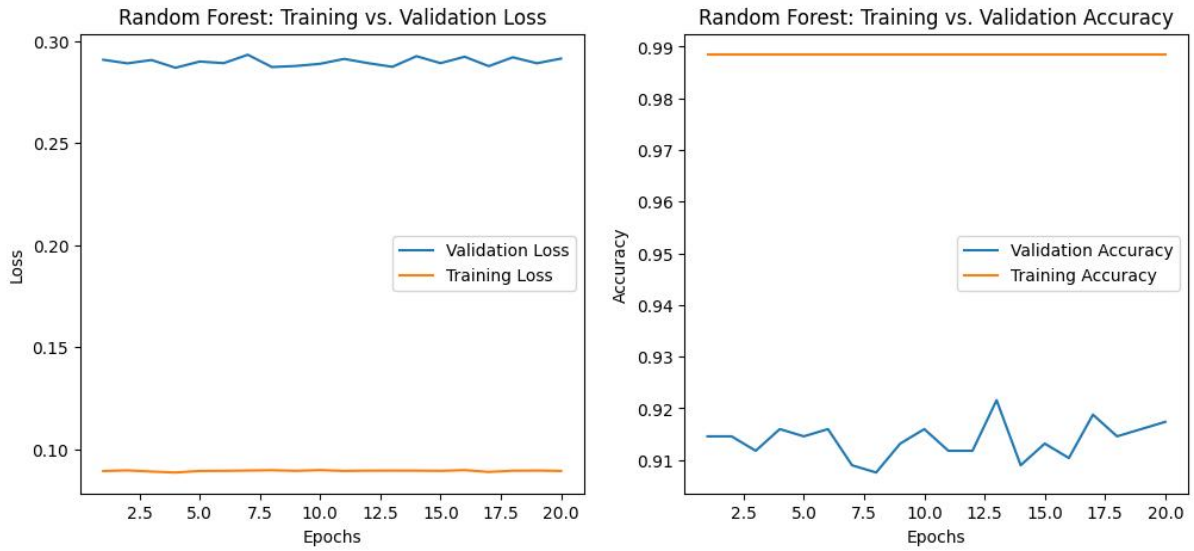


Fig. 4.3.3. Training vs Validation Accuracy & Loss (Random Forest in KC1)

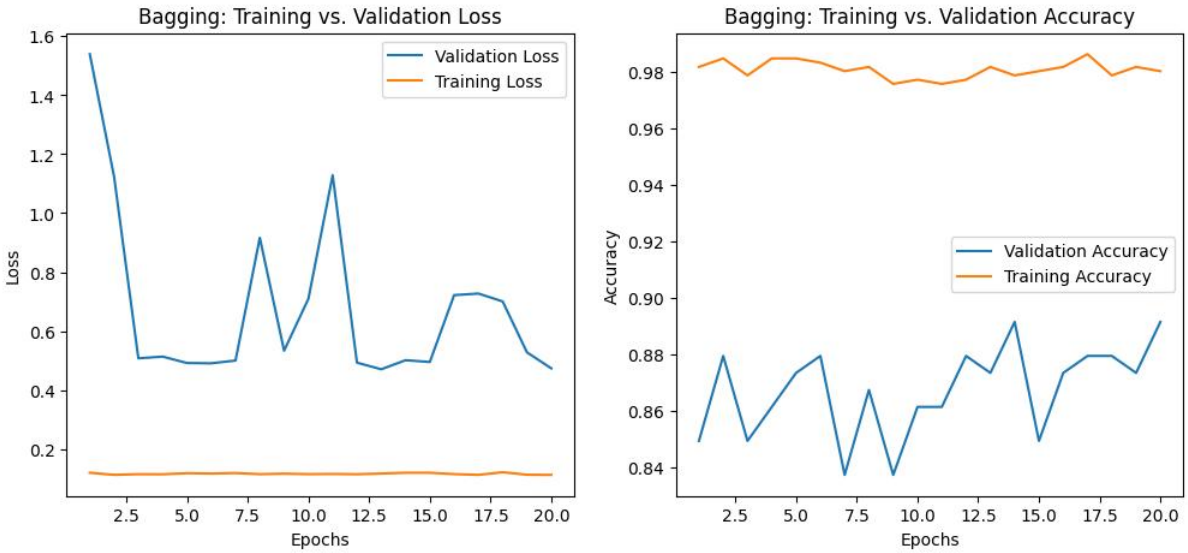


Fig. 4.3.4. Training vs Validation Accuracy & Loss (Bagging in KC2)

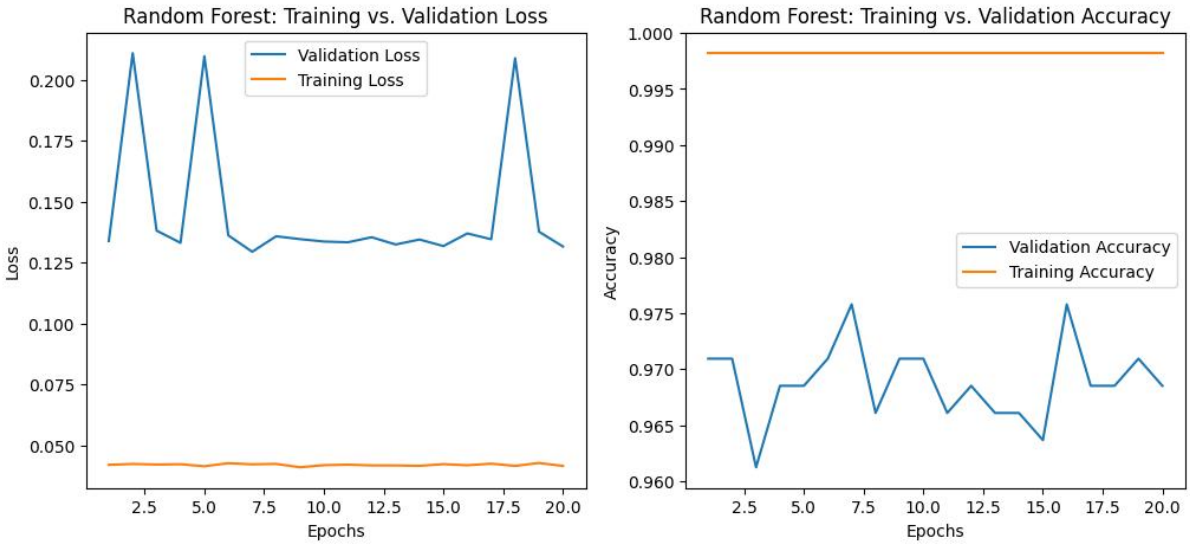


Fig. 4.3.5. Training vs Validation Accuracy & Loss (Random Forest in PC1)

4.4 Discussion

The results indicate that ensemble learners, particularly Random Forest and Bagging, performed exceptionally well in predicting software defects across different datasets. Random Forest performed better than all other algorithms in terms of accuracy and other evaluation criteria after hyper-parameter application. In particular:

Random Forest: Demonstrated superior performance with accuracies of 91% and 96.8% on Dataset 2 and Dataset 4, respectively. It also showed high precision and recall, indicating its robustness in defect prediction.
Bagging: Achieved notable results with accuracies of 93% and 89.15% on Dataset 1 and Dataset 3, respectively. It outperformed in terms of precision and recall, making it a dependable selection.

The strong performance of these ensemble methods underscores their effectiveness in handling the complexities of software defect prediction. Random Forest, in particular, emerged as the best-performing algorithm after hyper-parameter tuning, making it a highly reliable model for practical applications in the software development industry.

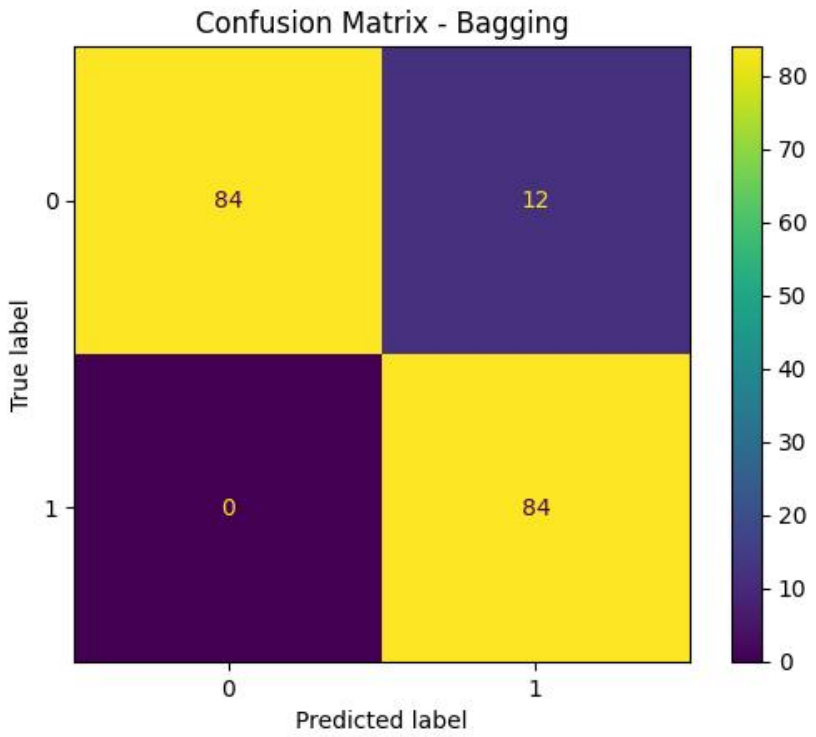


Fig. 4.4.1. Confusion Matrix (Bagging in CM1)

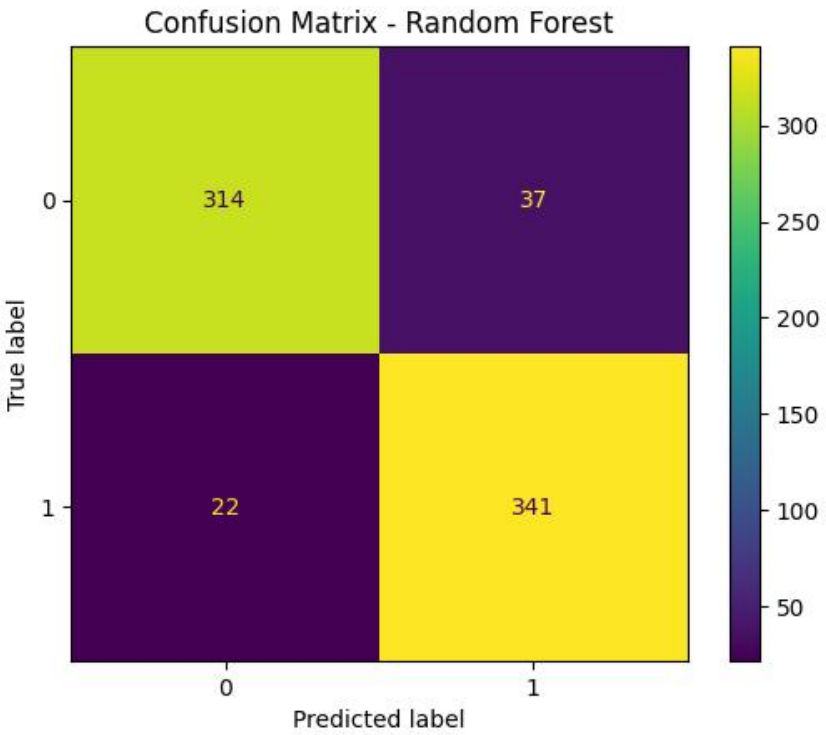


Fig. 4.4.2. Confusion Matrix (Random Forest in KC1)

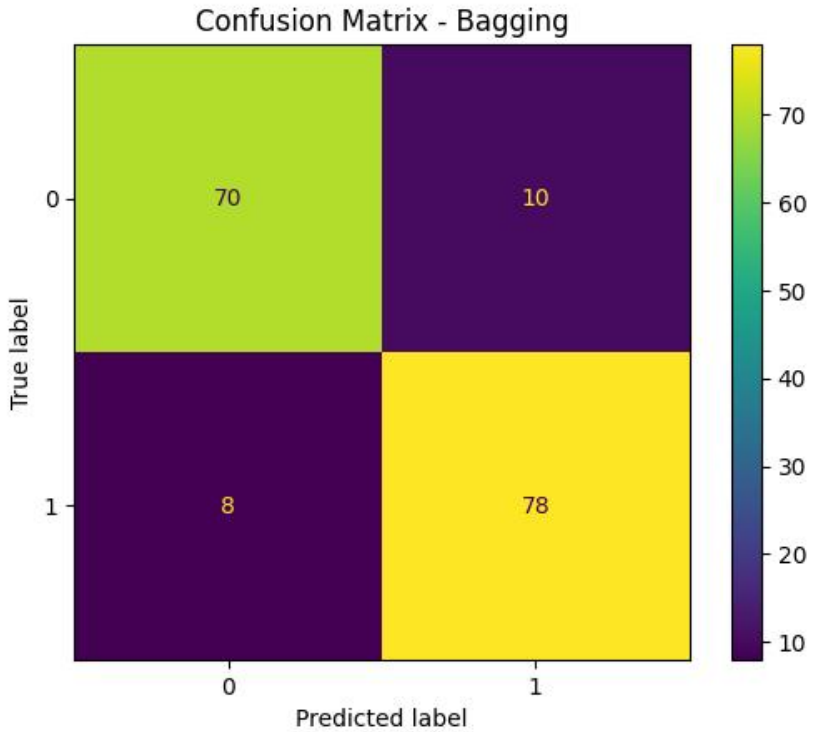


Fig. 4.4.3. Confusion Matrix (Bagging in KC2)

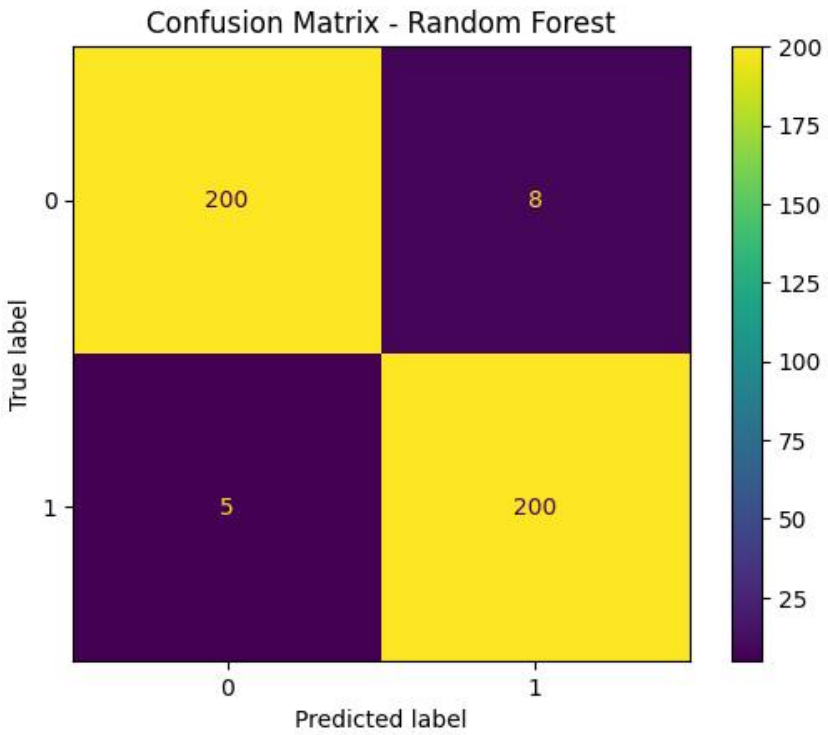


Fig. 4.4.4. Confusion Matrix (Random Forest in PC1)

In the evaluation of four datasets, the Random Forest algorithm consistently outperformed others, exhibiting superior accuracy rates on Dataset 2 and Dataset 4, respectively. These results were corroborated by the confusion matrices, where Random Forest displayed high precision and recall, indicating its reliability in defect prediction tasks. On the other hand, Bagging demonstrated notable performance with greater accuracies on Dataset 1 and Dataset 3, respectively. The associated confusion matrices highlighted Bagging's strong precision and recall values, further reinforcing its efficacy in defect prediction across diverse datasets.

CHAPTER 5
IMPACT ON SOCIETY, ENVIRONMENT, AND SUSTAINABILITY

5.1 Impact on Society

Software systems are an important part of modern life. They support important infrastructure, healthcare, banking, education, and many other areas. Finding and predicting software defects is a key part of making sure that these systems are reliable and safe. By making software better, this research helps stop defects that could have big effects on society, like data breaches, financial losses, and service interruptions for essential functions. It is especially helpful for developing software in areas with low incomes because defect prediction can lower the costs of solving defects and dependability of locally made software, which helps technology move forward and the economy grow.

5.2 Impact on Environment

Software development and maintenance activities, though primarily digital, have environmental impacts through their energy consumption and electronic waste. By decreasing the need for intensive debugging and rework, which in turn lowers energy consumption, efficient defect prediction and management can contribute to more sustainable software development methods. This contributes to a more sustainable software development environment. Moreover, robust and reliable software can extend the lifespan of hardware by reducing the need for frequent updates and replacements, thus mitigating electronic waste. Environmentally, this research promotes more sustainable IT practices and supports the broader goal of reducing the carbon footprint of the software industry.

5.3 Ethical Aspects

This research adheres to strict ethical guidelines, ensuring the security and privacy of user data. The results presented are accurate and verifiable, and the research has been conducted with integrity and transparency. No unethical practices were involved, and the study was carried out with full collaboration among team members under the guidance of our supervisor. The ethical implications of deploying defect prediction models, such as ensuring fairness and avoiding bias in software quality assessments, have also been carefully considered.

5.4 Sustainability Plan

The prediction of software defects is a continuous and evolving process that is critical for the sustainability of the software industry. This research aims to provide long-term benefits by improving the accuracy and efficiency of defect detection, thereby enhancing software quality and reliability. By assisting developers in identifying potential defects early in the development cycle, this research can significantly reduce the time and resources required for software maintenance. Future work will focus on expanding the range of data and employing additional algorithms to further improve prediction accuracy. The goal is to develop a robust, scalable solution that can adapt to the evolving needs of the software industry and contribute to its sustainable development.

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Summary of the study

The study I conducted showed that predicting and detecting defects in software is essential for software system safety and reliability. High-quality software is vital for operational efficiency and data security across sectors. The goal was to use NASA public repository datasets to build a strong machine learning model capable of predicting software bugs.
I preprocessed, handled missing values, and balanced datasets using machine learning methods and K-fold Cross-Validation for model validation. The ultimate goal was to improve software quality and prevent socially significant failures.

6.2 Conclusions

The main goal of this research was to establish a method for predicting software defects using machine learning. The results demonstrate that machine learning models, particularly those fine-tuned through hyperparameter optimization, are highly effective in this domain. Specifically, Random Forest and Bagging algorithms predicted defects well. Random Forest achieved an accuracy of 91% on Dataset 2 and 96.8% on Dataset 4, while Bagging reached 93% on Dataset 1 and 89.15% on Dataset 3. Ensemble learners, in general, performed admirably. After applying hyper-parameter tuning, Random Forest outperformed all other algorithms. In terms of precision and other evaluation metrics, both Random Forest and Bagging showed excellent performance.

I demonstrated great accuracy and reliability in defect prediction through considerable experimentation, as shown by multiple assessment criteria. The software development industry stands to gain much from this study if it can reduce the time and money spent on bug fixes and maintenance. Data quality and model accuracy were the two main areas where the study found drawbacks. Improving prediction capabilities can be achieved through the use of larger datasets and the continuous improvement of algorithms. I will be integrating these prediction models in the future so that developers may see any software problems in real-time.

6.3 Implication for further study

For future research, I recommend numerous improvements to improve the accuracy and applicability of software defect prediction models. To begin, increasing the quantity and diversity of the dataset can lead to improved generalization and model performance. Secondly, exploring and integrating additional machine learning algorithms may provide more robust predictions. Thirdly, incorporating contextual factors such as development environment and project-specific attributes can further refine predictions. In future work, I aim to:

- Improve model accuracy by expanding and refining the dataset.
- Apply a broader range of machine learning models to identify the most effective approaches.
- Develop a real-time prediction tool that can assist developers during the software development lifecycle.
- Incorporate environmental and contextual factors into the prediction model to enhance its reliability and applicability.

By solving these issues, I intend to contribute to the development of more dependable and efficient software systems, which ultimately will benefit the larger technological and societal landscape.

REFERENCE

[1] Assim, Marwa, Qasem Obeidat, and Mustafa Hammad. "Software defects prediction using machine learning algorithms." *2020 International conference on data analytics for business and industry: way towards a sustainable economy (ICDABI)*. IEEE, 2020.

[2] Yalçıner, Burcu, and Merve Özdeş. "Software defect estimation using machine learning algorithms." *2019 4th International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2019.

[3] Basili, Victor R. et al. "A Validation of Object-Oriented Design Metrics as Quality Indicators." *IEEE Trans. Software Eng.* 22 (1996): 751-761.

[4] Ceylan, Evren et al. "Software Defect Identification Using Machine Learning Techniques." *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)* (2006): 240-247.

[5] Elish, Karim O. and Mahmoud O. Elish. "Predicting defect-prone software modules using support vector machines." *J. Syst. Softw.* 81 (2008): 649-660.

[6] Guo, Lan et al. "Robust prediction of fault-proneness by random forests." *15th International Symposium on Software Reliability Engineering* (2004): 417-428.

[7] Kim, Sunghun et al. "Dealing with noise in defect prediction." *2011 33rd International Conference on Software Engineering (ICSE)* (2011): 481-490.

[8] Ruchika Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504–518, 2015

[9] Wang, Shuo and Xin Yao. "Using Class Imbalance Learning for Software Defect Prediction." *IEEE Transactions on Reliability* 62 (2013): 434-443.

[10] Catal, Cagatay, Banu Diri, and Bulent Ozumut. "An artificial immune system approach for fault prediction in object-oriented software." *2nd International Conference on Dependability of Computer Systems* (DepCoS-RELCOMEX'07). IEEE, 2007.

[11] Alenezi, M., Banitaan, S., and Obeidat, Q.. "Fault-proneness of open source systems: An empirical analysis." *Synapse* 1 (2014): 256.

[12] Iqbal, Ahmed, et al. "Performance Analysis of Machine Learning Techniques on Software Defect Prediction using NASA Datasets." *Int. J. Adv. Comput. Sci. Appl* 10.5, 2019

[13] I. A. and A. Saha, ―Software Defect Prediction: A Comparison Between Artificial Neural Network and Support Vector Machine,‖ Adv. Comput. Commun. Technol., pp. 51–61, 2017.

[14] Tomar, Divya, and Sonali Agarwal. "Prediction of defective software modules using class imbalance learning." *Applied Computational Intelligence and Soft Computing* 2016, 2016.

[15] Alsaeedi, Abdullah, and Mohammad Zubair Khan. "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study." *Journal of Software Engineering and Applications* 12.5, 85-100, 2019.

[16] Goel, Lipika, et al. "Cross-project defect prediction using data sampling for class imbalance learning: an empirical study." *International Journal of Parallel, Emergent and Distributed Systems*, 1-14, 2019.

[17] Tong, Haonan, et al. "Transfer-Learning Oriented Class Imbalance Learning for Cross-Project Defect Prediction." *arXiv preprint arXiv:1901.08429*, 2019.

[18] Bowes, David, Tracy Hall, and Jean Petrić. "Software defect prediction: do different classifiers find the same defects?." *Software Quality Journal* 26.2, 525-552, 2018.

[19] M. M. Rosli, N. H. I. Teo, N. S. M. Yusop and N. S. Moham, "The Design of a Software Fault Prone Application Using Evolutionary Algorithm," *IEEE Conference on Open Systems*, 2011.

[20] A. Gunes Koru and Hongfang Liu, "An Investigation of the Effect of Module Size on Defect Prediction Using Static Measures," *PROMISEPredictive Models in Software Engineering Workshop, ICSE* 2005, Saint Louis, Missouri, US, May 15th 2005

[21] Chidamber, Shyam R., and Chris F. Kemerer. "A metrics suite for object oriented design." *IEEE Transactions on software engineering* 20.6, 476-493, 1994.

[22] Mahfoodh, Hussain and Qasem Obediat. "*Software Risk Estimation Through Bug Reports Analysis and Bug-fix Time Predictions*." 2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT) (2020): 1-6.

[23] Kalmegh, S., "Analysis of weak data mining algorithm reptree, simple cart and randomtree for classification of indian news", *International Journal of Innovative Science, Engineering & Technology*, 2(2), 438- 446, 2015.

[24] Niculescu, S. P., *"Artificial neural networks and genetic algorithms in QSAR". Journal of molecular structure: THEOCHEM*, 622(1-2), 71- 83,2003.

[25] A. Kaur and R. Malhotra, "*Application of Random Forest in Predicting Fault-Prone Classes," 2008 International Conference on Advanced Computer Theory and Engineering*, Phuket, pp. 37-43, 2008.

[26] Dey, Ayon. "Machine learning algorithms: a review." *International Journal of Computer Science and Information Technologies* 7, no. 3, 1174-1179, 2016

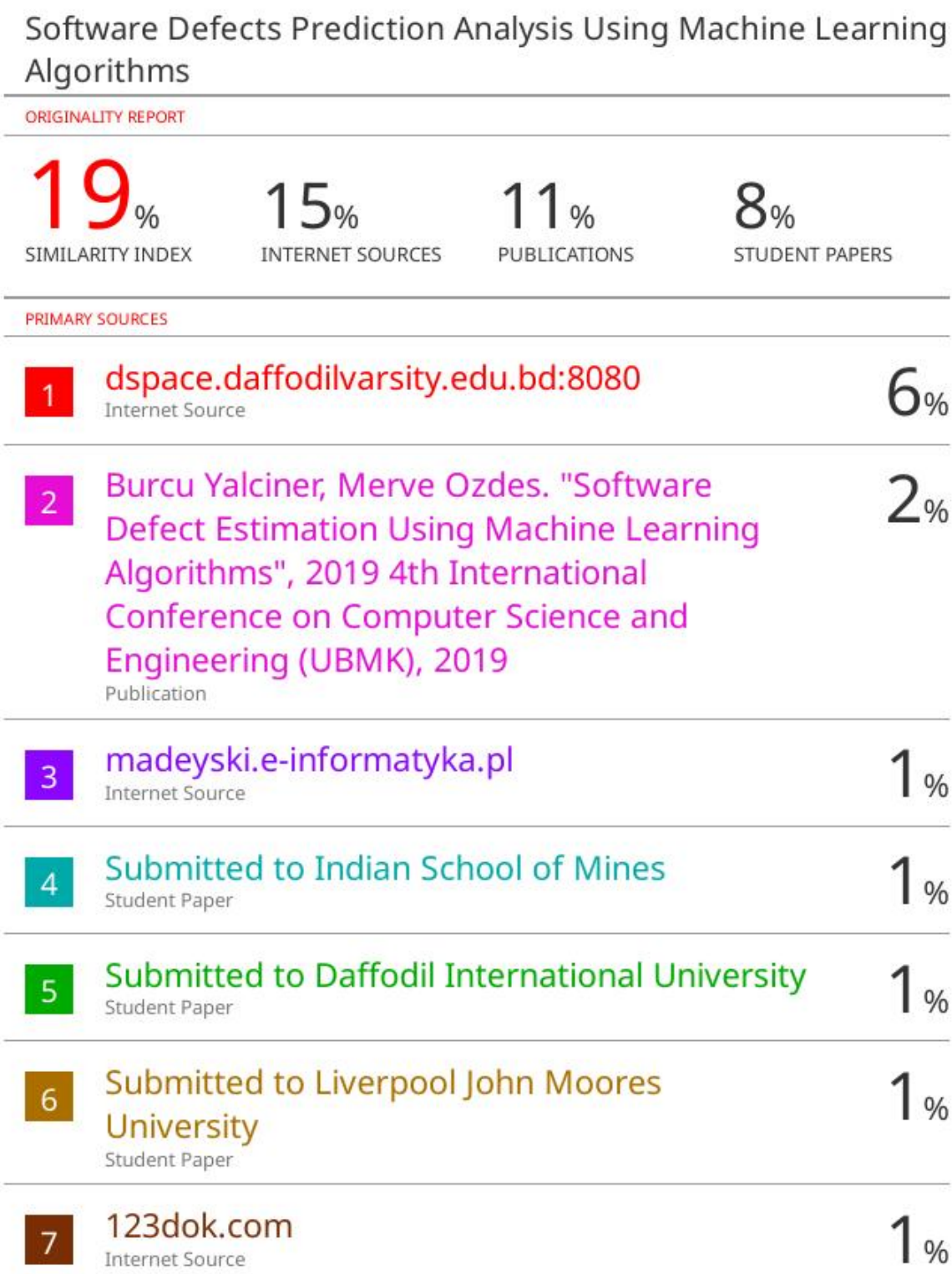
[27] Singh, Yogesh, Arvinder Kaur, and Ruchika Malhotra. "Software fault proneness prediction using support vector machines." *Proceedings of the world congress on engineering*, vol. 1. 2009.

[28] Botchkarev, Alexei, "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology." a*rXiv preprint arXiv*:1809.03006, 2018.

[29] Wu, Fang Jun. "Empirical tests of scale-free geometry in NASA data."*Advanced Materials Research*, vol. 301. Trans Tech Publications Ltd, 2011.

[30] Gayatri, N., et al. "Feature selection using decision tree induction in class level metrics dataset for software defect predictions." *Proceedings of the world congress on engineering and computer science* vol. 1. 2010.

Plagiarism report



Software Defects Prediction Analysis Using Machine Learning Algorithms

ORIGINALITY REPORT

19%	15%	11%	8%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dspace.daffodilvarsity.edu.bd:8080 Internet Source	6%
2	Burcu Yalciner, Merve Ozdes. "Software Defect Estimation Using Machine Learning Algorithms", 2019 4th International Conference on Computer Science and Engineering (UBMK), 2019 Publication	2%
3	madeyski.e-informatyka.pl Internet Source	1%
4	Submitted to Indian School of Mines Student Paper	1%
5	Submitted to Daffodil International University Student Paper	1%
6	Submitted to Liverpool John Moores University Student Paper	1%
7	123dok.com Internet Source	1%