

Hybrid Approach for Estimation of Software Defects using Supervised Machine Learning Method



MCS

Author

Umer Riaz

Registration Number

00000325136

Supervisor

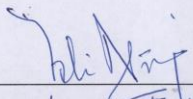
Assoc Prof Dr. Fahim Arif

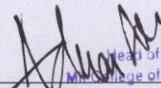
A thesis submitted to the faculty of Department of Computer Software Engineering, Military College of Signals, National University of Science of Technology (NUST), Rawalpindi in partial fulfillment of the requirements for the degree of MS in Computer Software Engineering

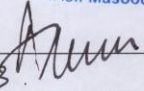
September 2023

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS Thesis written by **Mr. Umer Riaz**, Registration No. **00000325136**, of **Military College of Signals** has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations/MS Policy, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members and local evaluators of the scholar have also been incorporated in the said thesis.

Signature: 
Name of Supervisor ALP DR. Fakhri A
Date: 25-09-2023

Signature (HOD): 
Brig
Head of Dept of CSE
Military College of Sigs (NUST)
Date: 25/9/2023

Signature (Dean/Principal) 
Brig
Dean, MCS (NUST)
(Asif Masood, Phd)
Date: 25 Sep 2023

Declaration

I, *Umer Riaz* declare that this thesis titled “Hybrid Approach for Estimation of Software Defects using Supervised Machine Learning Method” and the work presented in it are 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

Umer Riaz,
00000325136

Dedication

This thesis is dedicated to all the Family Members, teachers and friends.

Abstract

The last few decades have seen extensive research on the critical activity of quality assurance, known as defect prediction, in the early phases of software development life cycle. Premature revealing of defective modules in software development can assist the development team in making efficient and effective use of the resources at hand to produce high-grade software in a less amount of time. Until now, numerous academics have created defect prediction models exploiting statistical and machine learning (ML) methods. By identifying hidden patterns among software features, the ML methodology is a useful technique for locating problematic modules. Three widely known NASA datasets are utilized in this work to forecast software problems using a variety of ML classification approaches. The projected approach in this thesis reflects the hybrid model, which is designed using ensemble-based ML algorithms that have enabled faults to be predicted in the software modules. Also, three datasets from NASA have been used to check the models' accuracy as a benchmark. The model suggests that the Adaboost classifier has shown the best accuracy amongst other ensemble-based ML techniques like NB, RF, Xgboost, beggingboost and catboost which produced 99.95% accuracy. The effectiveness of the employed classification approaches is assessed using a variety of metrics which include precision, recall, F-measure, accuracy and support.

Acknowledgments

Glory be to Allah (S.W.A), the Creator, the Sustainer of the Universe. Who only has the power to honour whom He please, and to abase whom He please. Verily no one can do anything without His will. From the day, I came to NUST till the day of my departure, He was the only one Who blessed me and opened ways for me, and showed me the path of success. There is nothing which can payback for His bounties throughout my research period to complete it successfully.

Umer Riaz

Contents

1	Introduction and Motivation	1
1.1	Overview	1
1.2	Introduction	2
1.3	Motivation	3
1.4	Objective	4
1.5	Problem Statement	5
1.6	Approach and Methodology	6
1.6.1	Data Collection	6
1.6.2	Data preprocessing	6
1.6.3	Model choice	7
1.6.4	Model Training	7
1.6.5	Model assessment	7
1.6.6	Model optimisation	7
1.6.7	Model Deployment	7
1.6.8	Validation and Testing	7
1.6.9	Documentation and Reporting	8
1.7	Target Group	8
1.7.1	Software Developers	8
1.7.2	Quality Assurance/Testers	8
1.7.3	Project Managers	8

CONTENTS

1.7.4	Software Maintenance Teams	9
1.7.5	Software Quality Assurance Teams	9
1.7.6	Software Researchers and Academics	9
1.8	Thesis Breakdown	9
1.9	Summary	10
2	Literature Review	12
2.1	Introduction	12
2.2	Previous Work	13
2.3	Summary	15
3	Design and Methodology	17
3.1	Introduction	17
3.2	Data Pre-processing	17
3.3	Data Normalization	19
3.3.1	Min-Max Scaling (Normalization)	19
3.3.2	Z-Score Standardization	19
3.3.3	Decimal Scaling	19
3.4	Environment	20
3.4.1	Software Environment	20
3.4.2	Hardware Environment	21
3.5	Data Collection	21
3.6	Performance Metrics	24
3.6.1	Classification Metrics	24
3.6.2	Regression Metrics	24
3.6.3	Clustering Metrics	26
3.6.4	Ranking Metrics	26
3.7	Summary	26

4	Implementation and Results	28
4.1	Introduction.....	28
4.2	SMOTE.....	28
4.3	Adaboost.....	29
4.4	Gradient Boosting Classifier.....	31
4.5	XGboost.....	32
4.6	Random forest.....	33
4.7	Accuracy Comparison.....	34
4.8	Precision Comparison.....	35
4.9	Recall Comparison.....	37
4.10	F1-Score Comparison.....	38
4.11	Other Authors Comparison.....	40
4.12	Summary.....	44
5	Discussion	45
5.1	Introduction.....	45
5.2	Accuracy Discussion.....	46
5.3	F1-Score Discussion.....	47
5.4	Recall Discussion.....	48
5.5	Precision Discussion.....	48
5.6	Summary.....	49
6	Conclusion	51
6.1	Conclusion.....	51
6.2	Future Work.....	52
6.3	Limitations.....	53
6.4	Summary.....	55

List of Tables

3.1	Criteria for Cleaning	22
3.2	Features of the datasets.....	23
3.3	Dataset Information.....	25
4.1	Comparison of Accuracy	34
4.2	Comparison of Precision	36
4.3	Comparison of Recall.....	37
4.4	Comparison F1score.....	39
4.5	Accracy Comparison with other Authors.....	41

List of Figures

1.1	Block diagram fro software defect prediction.....	11
3.1	Proposed Model for the software defect prediction.....	18
3.2	Algorithm for proposed model.....	27
4.1	Accuracy Comparison between different machine learning models	35
4.2	Precision Comparison between different machine learning models	36
4.3	Recall Comparison between different machine learning models	38
4.4	F1score Comparison between different machine learning models.....	39
4.5	Accuracy Comparison on CM1.....	41
4.6	Accuracy Comparison on JM1.....	42
4.7	Accuracy Comparison on PC1.....	43
4.8	Accuracy Comparison on all data sets on Proposed Model.....	44

List of Abbreviations

Abbreviations

RF; Random forest

AD ; Adaboost

SM; SMOTE

ML ; Machine learning

GBC ; Gradient boost classifier

NB ; Naive bays

Introduction and Motivation

1.1 Overview

Software defect prediction is a field of research that focuses on developing models and techniques to identify and predict software defects early in the software development lifecycle. The primary goal of software defect prediction is to assist software developers and testers in allocating their limited resources effectively and prioritizing their efforts in identifying and resolving potential defects.

The process of software defect prediction involves analyzing various software metrics and historical data to build predictive models that can forecast the likelihood of defects in software modules or components. These models are typically constructed using machine learning algorithms and statistical techniques.

The prediction models leverage features extracted from software artifacts, such as source code, design documents, and historical defect records. These features can include metrics related to code complexity, size, coupling, cohesion, and other quality indicators. By analyzing these features, the models can identify patterns and relationships that are indicative of potential defects.

Once the models are developed, they are trained and evaluated using labeled datasets that contain information about the presence or absence of defects. The efficacy of the models at accurately predicting flaws is measured using performance metrics like accuracy, precision, recall, and F1 score.

The applications of software defect prediction are diverse and can benefit software de-

velopment organizations in several ways. By identifying high-risk software components or modules early on, developers can allocate more resources to testing and debugging activities, resulting in improved software quality and reduced maintenance efforts. Furthermore, defect prediction models can aid in resource allocation, release planning, and software maintenance prioritization.

In recent years, there has been significant progress in software defect prediction, with the introduction of advanced machine learning techniques, ensemble methods, and the integration of various software metrics. Researchers are continuously exploring new approaches to enhance the accuracy and efficiency of defect prediction models.

In general, software defect prediction is essential for raising software quality and lowering maintenance expenses. It enables proactive defect management by identifying potential problem areas, allowing software development teams to focus their efforts and resources where they are most needed.

1.2 Introduction

Software defects are a great concern in software engineering. Software defects, or bugs, are the errors that come up during the software development phases. These stages are requirement, design, and development. As a result, dealing with bugs during those phases is a major challenge for software engineers. These flaws have a significant impact on the time and cost of software projects. Hence its importance increases more when it comes to the software modules that undergo intensive testing or are frequently tested. Software defect prediction is more important when it is done in the early stages of development. On the other hand, it becomes really hard when the product is released. The main reason is that if the product is released and then bugs and faults are detected, it is very difficult and hard in terms of cost and time of the software product, which are affected and increased by other factors too. So it is very important to trace these bugs now rather than at the end of the project. ML steps up and plays a vital role in solving these issues. To predict software defects, ML employs various classification and regression algorithms. ML algorithms, which are related to the regression algorithms, just predict the bugs in the software module. On the other hand, classification algorithms are applied for the classification of the faults or bugs, whether they are present or not. So in that case, ML models are trained on the available data and then used to predict

the faults in the modules. Classification ML algorithms that include K-nearest neighbor, Random forest, support vector machine and naive bays are examples of supervised ML algorithms. Also, ensemble ML techniques like boosting and bagging are used. ANN with multilayer perceptron and DNN, are also used.

1.3 Motivation

The data used in software defect prediction is typically unbalanced and must be balanced before it can be used in ML techniques. For that reason, many techniques are applied to the data in a uniform way. SMOTE is one of the techniques used to turn the imbalanced data into uniform data. A hybrid approach is another way to deal with the imbalanced data, which combines ensemble methods with other methods to get better results for the defect prediction. Ensemble ML is also used to handle the imbalanced data. It includes bagging, boosting, and stacking. Adaboost, xgboost, bagging classifier, random forest (RF), and catboost are examples of these algorithms. In this paper, the SMOTE resampling technique is used as a hybrid approach to resolve the issue of the imbalanced data. Also, the different supervised ML algorithms and ensemble ML techniques are used to compare the results. In next section, related work is discussed to learn about the work done in these fields. A further section describes the results and the discussion of all the experiments done using the three datasets, CM1, PC1, and JM1. In last section, conclusion and future work are discussed. Software defects can have detrimental effects on the quality and reliability of software systems. Detecting and resolving these defects in the early stages of development is crucial to ensure a successful software product. Traditional methods of defect prediction often rely on manual code review and testing, which can be time-consuming and resource-intensive.

In recent years, machine learning techniques have emerged as a promising approach to automate software defect prediction. By leveraging historical software data and employing various machine learning algorithms, it is possible to identify patterns and factors associated with software defects. These algorithms can analyze code metrics, change history, and defect reports to build predictive models.

1.4 Objective

The objective of software defect prediction using machine learning is to identify potential defect-prone areas in the software codebase. By doing so, development teams can allocate testing resources more efficiently and prioritize their efforts. This predictive capability can significantly improve software quality and reduce the occurrence of defects in production.

In this study, we aim to explore the effectiveness of different machine learning algorithms in predicting software defects. We will evaluate their performance using established evaluation metrics, such as precision, recall, and F1-score. By gaining insights into the applicability and performance of machine learning models in defect prediction, we can enhance software development practices and deliver more reliable software systems.

In the field of software development, defects or bugs can have a significant impact on the overall quality, reliability, and user satisfaction of software systems. Identifying and resolving these defects in the early stages of the software development lifecycle is crucial to ensure the delivery of a successful and robust software product. Traditional methods of defect prediction, such as manual code review and testing, have limitations in terms of time consumption and resource requirements.

In recent years, machine learning techniques have gained attention as a promising approach to automate software defect prediction. By leveraging historical software data and employing various machine learning algorithms, it is possible to identify patterns and factors associated with software defects. These algorithms can analyze code metrics, change history, and defect reports to build predictive models.

The main objective of software defect prediction using machine learning is to identify potential defect-prone areas in the software codebase. By utilizing historical data that includes information about past defects and associated software artifacts, machine learning models can learn from patterns and relationships within the data. These models can then predict the likelihood of encountering defects in specific parts of the codebase or during particular stages of development.

The benefits of software defect prediction using machine learning are multifold. Firstly, it helps development teams in effectively allocating their testing resources. By identifying areas of the code that are more likely to contain defects, testing efforts can be focused

on those areas, leading to more efficient and thorough testing. Secondly, by predicting defects early, development teams can take proactive measures to address potential issues, thereby reducing the occurrence of defects in production and improving software quality.

In this study, our aim is to explore the effectiveness of different machine learning algorithms in predicting software defects. We will investigate the applicability and performance of various algorithms such as decision trees, random forests, support vector machines, and neural networks. Additionally, we will evaluate the models using established evaluation metrics, such as precision, recall, and F1-score, to assess their predictive accuracy and reliability.

By gaining insights into the applicability and performance of machine learning models in defect prediction, we can enhance software development practices. The findings from this study will provide valuable guidance to development teams, enabling them to make informed decisions regarding defect prevention and quality improvement strategies. Ultimately, the goal is to deliver more reliable and robust software systems that meet user expectations and minimize the impact of defects on software functionality.

The problem of software defects poses significant challenges in the software development process. Undetected defects can lead to costly errors, increased maintenance efforts, and potential system failures. Therefore, there is a need to develop effective methods for early identification and prediction of software defects.

1.5 Problem Statement

The problem statement in software defect prediction revolves around creating accurate and reliable models that can predict the likelihood of defects in software modules or components. The objective is to provide software developers and testers with insights into which parts of the software are more likely to contain defects, allowing them to allocate their limited resources efficiently and prioritize testing and debugging efforts.

The problem involves addressing several key challenges, including:

Handling the vast amount of software metrics and data available, extracting meaningful features, and selecting the most relevant metrics for defect prediction. Dealing with imbalanced datasets where the number of defective instances is significantly smaller than the non-defective ones. Overcoming the variability and heterogeneity in software

projects, as different projects may have unique characteristics and defect patterns. Developing models that can handle the dynamic nature of software development, where new features, updates, and changes occur frequently. Ensuring the interpretability of the prediction models, allowing stakeholders to understand the factors contributing to defect predictions and make informed decisions. Addressing these challenges requires the development and application of advanced machine learning algorithms, data preprocessing techniques, feature selection methods, and model evaluation approaches. The ultimate goal is to create accurate and robust defect prediction models that can assist in identifying and mitigating software defects early in the development process, thereby improving software quality, reducing maintenance efforts, and enhancing overall system reliability.

1.6 Approach and Methodology

The approach and methodology for software defect prediction involve a systematic process that combines data analysis, machine learning techniques, and evaluation methods. The following steps outline the typical approach and methodology:

1.6.1 Data Collection

Gather relevant data from software repositories, version control systems, bug tracking systems, and other sources. This data includes software metrics, historical defect records, and other contextual information.

1.6.2 Data preprocessing

To assure the quality and usefulness of the collected data for analysis, clean and preprocess it. To produce a trustworthy and consistent dataset, deal with missing values, eliminate outliers, and normalise or scale the data. Choose the software metrics or attributes that have the most bearing on the prediction of software defects. To find the important features, use approaches like correlation analysis, information gain, or feature importance ranking.

1.6.3 Model choice

Depending on the nature of the issue and the features of the dataset, select the most appropriate machine learning techniques. Decision trees, random forests, support vector machines (SVM), artificial neural networks (ANN), and gradient boosting classifiers (GBC) are examples of frequently used techniques.

1.6.4 Model Training

Divide the dataset into training and validation sets for the model. Utilise the training set to train the chosen machine learning models, then adjust their hyperparameters to enhance performance. The generalisation capacity of the model can be evaluated using cross-validation approaches.

1.6.5 Model assessment

Measure the trained models' accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (AUC-ROC) using the relevant assessment measures. To find the most efficient strategy, compare the results of various models.

1.6.6 Model optimisation

To increase the accuracy and resilience of the chosen models, fine-tune them using ensemble approaches like bagging or boosting and changing their hyperparameters.

1.6.7 Model Deployment

Deploy the optimized models in a real-world setting to predict software defects in new or unseen instances. Monitor the model's performance and recalibrate as necessary to maintain accuracy and relevance.

1.6.8 Validation and Testing

Validate the performance of the deployed models by comparing their predictions with actual defect occurrences. Continuously test and evaluate the models using new data to ensure their effectiveness in predicting defects.

1.6.9 Documentation and Reporting

Document the entire approach, methodology, and results obtained. Provide detailed reports and insights to stakeholders, including developers, testers, and project managers, to guide decision-making and improve software quality.

1.7 Target Group

The target group for software defect prediction encompasses various stakeholders involved in the software development and testing process. These stakeholders can benefit from the insights and predictions provided by defect prediction models. The primary target groups include:

1.7.1 Software Developers

Software developers are directly involved in writing, modifying, and maintaining code. They can benefit from defect prediction models by identifying high-risk areas within their codebase and allocating resources for more rigorous testing and debugging. The models help developers prioritize their efforts and focus on modules or components that are more likely to contain defects.

1.7.2 Quality Assurance/Testers

Testers are responsible for verifying the functionality and quality of software through various testing techniques. Defect prediction models provide testers with valuable information on areas that are prone to defects, allowing them to design targeted test cases and perform thorough testing on critical components. This helps improve the effectiveness and efficiency of testing efforts.

1.7.3 Project Managers

Project managers oversee the software development process and are responsible for resource allocation, scheduling, and project planning. Defect prediction models assist project managers in making informed decisions about resource allocation, ensuring that

resources are allocated where they are most needed. This helps optimize the project timeline, budget, and overall development process.

1.7.4 Software Maintenance Teams

After software deployment, maintenance teams are responsible for handling bug fixes, updates, and enhancements. Defect prediction models aid maintenance teams in identifying areas that require immediate attention, allowing them to prioritize their efforts and address critical defects promptly. This leads to more efficient and targeted maintenance activities.

1.7.5 Software Quality Assurance Teams

Quality assurance teams focus on ensuring the overall quality and reliability of software systems. Defect prediction models help these teams in identifying potential defects early, allowing them to design and implement effective quality assurance processes and techniques. This enhances the overall quality control measures in place.

1.7.6 Software Researchers and Academics

Software Researchers and Academics: Researchers and academics in the field of software engineering and defect prediction use these models to advance the state of the art. They explore new algorithms, techniques, and methodologies to improve the accuracy and effectiveness of defect prediction models. Their research helps enhance the understanding and application of defect prediction in the industry.

1.8 Thesis Breakdown

The research has been done in a lot of different phases and steps. It is divided into the following chapters.

Chapter 1 Introduction and Motivation: An Overview of Software defect prediction(SDP).

Chapter 2 Literature Review: Discussion and highlighting of work already carried out on this topic by other people.

Chapter 3 Design and Methodology: The explanation of the proposed methodology to overcome the problems which are observed.

Chapter 4 Implementation and Results: Testing the validity of the methodology by using dataset in python and calculating results of accuracy.

Chapter 5 Discussion: Critical analysis of the results will be done.

Chapter 6 Conclusion: This section provides a recap of all the work done and also shows a direction for the future of this research

1.9 Summary

In summary, software defect prediction using machine learning techniques has emerged as a promising approach to automate the identification of potential defects in software systems. By leveraging historical software data and employing various machine learning algorithms, patterns and factors associated with software defects can be identified. This predictive capability enables development teams to allocate testing resources more efficiently and prioritize their efforts.

The objective of this study is to explore the effectiveness of different machine learning algorithms in predicting software defects. By evaluating their performance using established metrics, we aim to assess the accuracy and reliability of these models. The findings will provide valuable insights into the applicability of machine learning in defect prediction, allowing development teams to make informed decisions regarding defect prevention and quality improvement strategies.

By utilizing machine learning techniques for defect prediction, software development practices can be enhanced, leading to the delivery of more reliable and robust software systems. Early identification and resolution of defects contribute to improved software quality, user satisfaction, and overall system reliability. The results of this study will contribute to the body of knowledge in software defect prediction and aid in the development of effective strategies for defect prevention and quality assurance.

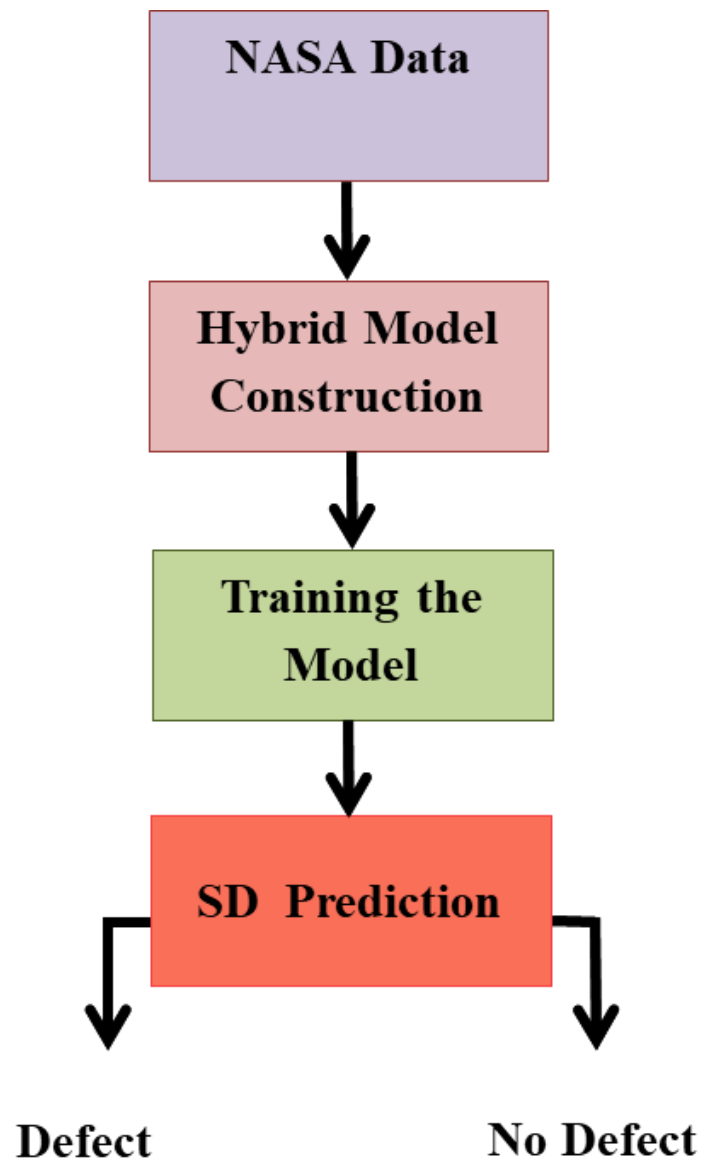


Figure 1.1: Block diagram fro software defect prediction.

Literature Review

2.1 Introduction

In the field of software defect prediction using machine learning, extensive research has been conducted to develop effective models and techniques for identifying and predicting software defects. This section provides an overview of the related work in this area, highlighting the advancements, methodologies, and key findings of recent studies.

One area of focus in related work is the selection and extraction of appropriate software metrics for defect prediction. Researchers have explored various software metrics, such as line of code, cyclomatic complexity, halstead volume, and design complexity, to capture different aspects of software quality and complexity. These metrics serve as crucial input features for machine learning models and play a significant role in identifying patterns and relationships between software metrics and defect occurrences.

Several machine learning algorithms have been employed in defect prediction studies. These algorithms include decision trees, random forests, support vector machines (SVM), artificial neural networks (ANN), and gradient boosting classifiers (GBC). Each algorithm has its strengths and limitations, and researchers have evaluated their performance and compared their effectiveness in different scenarios. The goal is to identify the most suitable algorithm for accurately predicting software defects and achieving high prediction performance.

To address the challenge of imbalanced datasets, researchers have explored various techniques such as oversampling, undersampling, and synthetic data generation. These techniques aim to balance the distribution of defective and non-defective instances to

prevent bias towards the majority class and improve the overall prediction performance. Additionally, feature selection and engineering methods have been applied to identify the most relevant and informative features for defect prediction. Dimensionality reduction techniques, such as principal component analysis (PCA) and feature importance ranking, have been used to reduce the number of features and eliminate noise, leading to improved model performance and interpretability.

Another aspect of related work is the evaluation of prediction models using appropriate performance metrics. Accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (AUC-ROC) are examples of frequently used measures. These metrics help assess the predictive power and generalization capability of the models and allow for comparisons between different approaches.

Furthermore, researchers have investigated the transferability of defect prediction models across different software projects and domains. The generalizability of models is crucial for their practical application, as it allows for knowledge transfer and reusability of trained models in new contexts.

The related work in software defect prediction using machine learning has focused on feature selection, algorithm selection, imbalanced data handling, evaluation metrics, and generalizability. Researchers have made significant contributions to understanding the complex relationship between software metrics and defects, developing accurate prediction models, and addressing the challenges in real-world software defect prediction scenarios. By building upon this body of work, researchers and practitioners can continue to advance the field and enhance the effectiveness of software defect prediction using machine learning techniques.

2.2 Previous Work

[9] used supervised ML algorithms like random forest binning and boosting for the software defect prediction. For the experimentation, authors have used different NASA datasets. To assess the performance of the models, author used precision, recall and accuracy as performance metrics. The results show that random forest, random forest with adaboost, and begging with DS give the best results against the other ML models. Also, to resolve the issue of imbalanced data, he used the SMOTE resampling technique.

[17] experimented on different datasets from PROMISE dataset repository and also the NASA datasets. He proposed the KPWE framework for software defect prediction, a new method that outperforms the other models presented previously.

[14]Manjula et al. [11] used the NASA dataset for the benchmark dataset. Four datasets were used for the experiments and results. For the optimization of the data, genetic algorithms were used, and a deep neural network was applied for taxonomy. The results show that the KC1 dataset has 97.82%, the CM1 dataset has 97.59%, the PC3 dataset has 97.96% and the PC4 dataset has 98% accuracy. All the experiments were implemented using MATLAB.

[20] used a NASA dataset from PROMISE repository. The dataset used for the trials is KC2, PC3, JM1, and CM1. Different ML algorithms were applied to get the best accuracy, like random forest, logistic regression, decision tree, and xgboost. The results show that the XGBoost algorithm, with some changes in the parameters, proved to be the best classifier for fault detection against other classical ML algorithms.

[18] applied an MLP neural network, a convolutional neural network (CNN) and ML algorithms for software fault detection. The experiments made use of NASA datasets. Manipulation of the parameters is the key to increasing the accuracy of bug detection, which is implemented during experiments. As a result, accuracy improved to 43.5% for PC1, 8% for KC1, 18% for KC2 and 76.5% for the CM1.

[25] used different ML algorithms for the software defect prediction (SPD). The algorithms include the support vector machine, naive Bayes, Bayesian belief network, and convolutional neural network. The results indicate that the convolutional network proves to be the best in case of accuracy as equated to the other ML algorithms, which give an average accuracy of 70.2

In current era, software defect prediction is unique and important topic in the field of ML. Many techniques are used in order to get good results. The correct detection of faults and bugs can save developers time and money. In their paper, the authors [5] have used genetic algorithms and deep neural networks on the NASA dataset. The results have shown that genetic algorithms have a mean probability of detection with a progression of 71

In [4], researchers presented a method for predicting software defects that combines adaptive dimensional analysis with conventional radial basis functions. Model for opti-

mization based on biogeography. Five NASA datasets from PROMISE repository were used for the experiment, and the findings demonstrated that new method was more accurate than earlier techniques.

On the subject of predicting software defects, researchers in [36] contrasted an artificial neural network (ANN) with a support vector machine (SVM). Seven NASA datasets from PROMISE repository were used. The specificity, recall, and accuracy of the performance were assessed and results demonstrated improved SVM performance.

The Whale Optimisation Algorithm feature selection approach, which uses metaheuristic search to select fewer but closely related features, has now been enhanced by [29]. They also combined CNN with kernel extreme learning machines (KELM) to produce a hybrid defect classifier that incorporates the chosen features into the abstract deep semantic features produced by CNN and improves prediction performance by fully utilising KELM's powerful classification capability. Their findings proved the benefits of the hybrid strategy.

Studies show that Their methods can find 32.22 percent more flaws than the most advanced model currently available [8].suggested using stacked denoising autoencoders (SDAE) to transform manually created measurements in the NASA dataset into useful metrics, and they used ensemble techniques to find errors. The results suggest that accurate software fault prediction may benefit from deep representations of current metrics.

2.3 Summary

The related work in software defect prediction using machine learning has made significant contributions to the field, focusing on various aspects such as feature selection, algorithm selection, imbalanced data handling, evaluation metrics, and generalizability.

Researchers have explored different software metrics to capture software quality and complexity, including line of code, cyclomatic complexity, halstead volume, and design complexity. These metrics serve as crucial input features for machine learning models in predicting software defects.

Multiple machine learning algorithms, such as naive bays, random forests, adaboost, xgb and GBC, have been employed and compared for their effectiveness in defect prediction.

Researchers have evaluated the performance of these algorithms and identified their strengths and limitations in different contexts.

To address the challenge of imbalanced datasets, techniques such as oversampling, undersampling, and synthetic data generation have been explored. These methods aim to balance the distribution of defective and non-defective instances, improving the overall prediction performance.

Feature selection and engineering methods have been used to identify the most relevant features for defect prediction. Dimensionality reduction techniques and feature importance ranking have helped eliminate noise and improve model performance and interpretability.

Evaluation metrics such as accuracy, precision, recall, F1 score, and AUC-ROC have been employed to assess the performance and generalization capability of prediction models. These metrics allow for comparisons between different approaches and help in selecting the most effective models.

The transferability of defect prediction models across different software projects and domains has been investigated. Understanding the generalizability of models is crucial for their practical application and knowledge transfer to new contexts.

Overall, the related work has contributed to a deeper understanding of the relationship between software metrics and defects, the development of accurate prediction models, and addressing challenges in real-world defect prediction scenarios. By building upon this work, researchers and practitioners can further advance the field and enhance the effectiveness of software defect prediction using machine learning techniques.

Design and Methodology

3.1 Introduction

Using NASA benchmark datasets, this study evaluates how well different ML classifiers predict software defects. Each dataset also includes a known output class and a number of features. The expected output, or objective class, that is based on former available features. Dependent attributes are those that are used to predict the dependent attribute, whereas independent attributes are those that are used to predict the independent attributes. The dependent features in datasets chosen for this investigation have values of Y or N. Y indicates that a certain instance or module of software has a propensity to be defective, whereas N indicates that it is not. A total of three cleansed NASA datasets are utilised in this study's tests and the databases contain the CM1, JM1, kc1, kc2, mc1 and PC1. The suggested model of research work is given in the following figure 3.1. which include the dataset from the PROMISE repository and the NASA dataset. NASA has a total of ten datasets available. Only six datasets are used for the experiments in this study: JM1, CM1, kc1, kc2, mc1 and PC1.

3.2 Data Pre-processing

Data pre-processing is critical before using ML algorithms for experimentation. For that purpose, standard feature scaling is used to form the scaled data. Also, the SMOTE, which is a synthetic minority oversampling method, is used to normalize the disparity data. All the null values are removed. For scaling the data, the min-max scalar is used.



Figure 3.1: Proposed Model for the software defect prediction.

3.3 Data Normalization

Data normalization, also known as feature scaling or data standardization, is a common preprocessing step in machine learning. It involves transforming the numerical features of a dataset to a common scale, usually between 0 and 1 or with a mean of 0 and a standard deviation of 1. Normalization helps to ensure that the features have comparable magnitudes and prevents certain features from dominating others during the learning process.

There are several common methods for data normalization:

3.3.1 Min-Max Scaling (Normalization)

This method scales the values of the features to a particular range, commonly between 0 and 1. The min-max scaling equation is given by:

$X_{\text{normalized}} = (X - X_{\text{min}}) / (X_{\text{max}} - X_{\text{min}})$, where X is the original feature, X_{min} is the dataset's smallest value for X , and X_{max} is its maximum value for X .

3.3.2 Z-Score Standardization

By using this technique, the characteristics are standardised to have a mean of 0 and a standard deviation of 1. The z-score standardisation formula is:

$X_{\text{standardized}} = (X - X_{\text{mean}}) / X_{\text{std}}$, where X is the original feature, X_{mean} is X in the dataset's mean, and X_{std} is X in the dataset's standard deviation.

3.3.3 Decimal Scaling

This method scales the values by shifting the decimal point of the feature values. The scaling factor is determined by the maximum absolute value of the feature. For example, if the maximum absolute value is 1000, the decimal scaling factor would be 3 to shift the decimal point by 3 positions.

The normalisation technique chosen will depend on the dataset's unique properties and the needs of the machine learning algorithm. It's important to note that normalization is typically applied to the feature values and not the target variable.

Certain machine learning methods, especially those that are sensitive to the scale of the

input characteristics (such gradient descent-based algorithms), can perform and converge better when they are normalised. But there are other algorithms that are scale-invariant and don't always need normalisation, including decision trees and random forests.

It's generally recommended to perform normalization as a preprocessing step when working with numerical features, unless there are specific reasons to avoid it based on the characteristics of the dataset or the requirements of the machine learning algorithm.

3.4 Environment

The environmental setting for the implementation of software defect prediction is a critical factor that influences the success and effectiveness of the predictive models. Here is an overview of the environmental setting:

3.4.1 Software Environment

Programming Language: Popular choices for implementing machine learning models include Python, R, and Julia. Python, with libraries such as TensorFlow, Keras, PyTorch, or scikit-learn, is widely used due to its extensive machine learning ecosystem.

Integrated Development Environment (IDE): IDEs like PyCharm, Jupyter Notebook, Spyder, or Visual Studio Code provide a user-friendly environment for coding, debugging, and running machine learning code.

Machine Learning Libraries and Frameworks: Various libraries and frameworks provide pre-built tools and algorithms for machine learning, deep learning, and data preprocessing. Some popular ones include TensorFlow, PyTorch, scikit-learn, XGBoost, and Keras.

Data Manipulation and Analysis Tools: Tools such as pandas and NumPy in Python enable data manipulation, preprocessing, and analysis tasks.

Data Visualization Libraries: Libraries like Matplotlib, Seaborn, or Plotly assist in visualizing data, model performance, and other relevant metrics.

3.4.2 Hardware Environment

Central Processing Unit (CPU): CPUs are essential for running machine learning models. The number of cores and processing power influence the model's speed during training and inference.

Graphics Processing Unit (GPU): GPUs excel at parallel processing, making them ideal for training deep learning models. They significantly accelerate computations due to their high number of cores.

Random Access Memory (RAM): Sufficient RAM is necessary for storing and manipulating large datasets during training and inference. The amount of RAM required depends on the dataset size and complexity.

Storage: Adequate storage is crucial for storing datasets, model parameters, and intermediate results. Solid-state drives (SSDs) are preferred for faster data access.

3.5 Data Collection

For the experimentation, six well-known datasets are used from open-source public datasets: JM1, CM1, kc1, kc2, mc1 and PC1. These datasets are gathered from the NASA datasets, which are available on the Promise repository. All the datasets include 23 attributes. NASA has developed this dataset, which includes 10 overall datasets under its program. Sheppard et al. cleaned up these datasets. DS', which contained duplicated and inconsistent instances, and DS'', which excludes replicated and unreliable instances, are two different clean datasets that are offered by [1]. These datasets were primarily offered at NASA repository, however were later taken down. These datasets were obtained from NASA dataset which are available online, which houses backup copies of NASA records. [9][17][14] have already utilized and discussed these cleaned datasets. Table 3.1 displays the cleaning standards used by [1].

Hence, for our proposed work, we include three datasets that are in pure form. Each dataset has used the MCCABE metrics for every occurrence. There are 40 features in this, but we have used 22 features from all the features that are in NASA datasets. Those features are described in Table 3.2.

The dataset details in terms of class distribution are described in Table 3.3. There are

Table 3.1: Criteria for Cleaning

Criteria	Category	Explanation
1	Identical cases	Instances in which all metrics, including the class label, have the same values.
2	Inconsistent cases	Instances that fall under the criteria of Case 1 but have different class labels.
3	Cases with missing values	Instances where one or more observations are lacking.
4	Cases with conflicting feature values	Instances when at least two metric values go against a referential integrity restriction. For example, LOC TOTAL is less than the commented LOC, even though LOC TOTAL is a subset of commented LOC.
5	Cases with implausible values	Instances where some integrity restrictions are broken. For example, LOC value = 1.1, which is an implausible value.

Table 3.2: Features of the datasets

Serial number	Metrics name	Type
1	Line of code	McCabe
2	Cyclomatic complexity	McCabe
3	Essential complexity	McCabe
4	Design complexity	McCabe
5	Halstead operators and operands	Halstead
6	Halstead volume	Halstead
7	Halstead program length	Halstead
8	Halstead difficulty	Halstead
9	Halstead intelligence	Halstead
10	Halstead effort	Halstead
11	Halstead time estimator	Halstead
12	Halstead line count	Halstead
13	Halstead comments count	Halstead
14	Halstead blank line count	Halstead
15	IO code and comments	Miscellaneous
16	Unique operators	Miscellaneous
17	Unique operands	Miscellaneous
18	Total operators	Miscellaneous
19	Total operands	Miscellaneous
20	Branch count	Miscellaneous
21	b: numeric	Halstead
22	Defects	False or true

10885 instances in the JM1 dataset. While the defect percentage is 19.3%, the total features are 23. The CM1 dataset has 498 instances with a defect percentage of 9.8% and 23 features as well. PC1 is a medium-sized dataset that has 1109 rows, a defect percentage of 6.9%, and 23 features.

3.6 Performance Metrics

Machine learning performance measures are used to assess the efficacy and performance of machine learning models. These metrics offer information on the model's performance and aid in comparing several models or fine-tuning its parameters. The choice of performance metrics depends on the specific task, such as classification, regression, or clustering. Here are some commonly used performance metrics in machine learning:

3.6.1 Classification Metrics

Accuracy: Measures how accurately the model's predictions were made overall.

Precision: The percentage of accurate positive forecasts compared to all positive predictions.

Sensitivity: Measures the fraction of accurate positive predictions among all instances of positive data (also known as sensitivity or true positive rate). Measures the percentage of accurate negative predictions among all occurrences of actual bad behaviour.

F1score: A balanced measure of model performance is provided by the F1-Score, which combines precision and recall.

AUC-ROC: Area Plotting the True Positive Rate against the False Positive Rate, the area under the receiver operating characteristic curve (AUC-ROC) measures how well the model can distinguish between classes.

Log Loss: Measures a probabilistic classifier's effectiveness by calculating the difference between predicted probabilities and actual class labels (log loss).

3.6.2 Regression Metrics

MAE:The average absolute difference between the expected and actual values is measured by mean absolute error (MAE). The average squared difference between the ex-

Table 3.3: Dataset Information

Data Source	Dataset Name	Total entries	Trues	False	Defect %	Total Features
NASA	JM1	10885	2106	8779	19.3%	23
NASA	CM1	498	49	449	9.8%	23
NASA	PC1	1109	77	1032	6.9%	23

pected and actual values is measured by the mean squared error (MSE). Root The square root of the mean square error (MSE), the mean squared error (RMSE), provides a measurement in the same units as the target variable.

R-squared (Coefficient of Determination): Denotes the percentage of the target variable's variance that the model predicts.

3.6.3 Clustering Metrics

Silhouette Coefficient Measures how well instances within a cluster are similar to each other and dissimilar to instances in other clusters.

Calinski-Harabasz Index Evaluates the separation between clusters based on the ratio of between-cluster dispersion to within-cluster dispersion.

3.6.4 Ranking Metrics

Measures the fraction of pertinent instances in the top K ranked outcomes, or precision at K. The Mean Average Precision (MAP) formula determines the average precision among all feasible K values. The performance measures used in machine learning are only a few examples. The choice of relevant metrics is influenced by the analysis's specific goals, the type of data used, and the problem domain.

3.7 Summary

The methodology for software defect prediction using machine learning involves collecting and preprocessing data, selecting relevant features, training and evaluating machine learning models, optimizing their performance, and deploying them for real-world testing. It includes steps such as data collection, preprocessing, feature selection, model training and evaluation, optimization, and deployment. By following this systematic approach, accurate and reliable prediction models can be developed to enhance software quality and maintenance processes.

Algorithm: SPD using ML classifiers**Input:** SPD Datasets, Ensemble Classifiers**Results:** SPD Accuracy, precision, recall, F1score

Datasets \leftarrow {NASA};
 Scalars \leftarrow {Standard scalar (min-max), SMOTE};
 Classifiers \leftarrow {RF, NB, Xgboost, Gradientboost, Adaboost};
 SMOTE-RF, SMOTE-NB, SMOTE-Adaboost, SMOTE-Xgboost, SMOTE-GBC
 AccuracyScore \leftarrow {};
 Recall Scores \leftarrow {};
 Precision Score \leftarrow {};
 F1 Scores \leftarrow {};

Start For

Step1: Read \rightarrow D(JM1, CM1, PC1,) **do**
 For Train test split for D X_{train}, X_{test}
 $\{X_{train}, X_{test}\} \leftarrow$ StandardScalar(X_{train}, X_{test});

$$X = \frac{X - X_{min}}{X_{max} - X_{min}}$$
 Scaled(X_{train}, X_{test}) \leftarrow SMOTE(X_{train}, X_{test});
 For model \in Classifiers **do**
 Step2: model \leftarrow {TrainClassifier{Scaled $X_{train}, X_{trainLabel}$ }};
 Step3: model.fit \leftarrow { X_{train}, Y_{train} };
 Step4: Prediction \leftarrow { X_{test} };
 Step5: Accuracy \leftarrow ComputeAccuracy{Prediction, $X_{Testlabel}$ };
 Step6: Precision \leftarrow ComputeAccuracy{Prediction, $X_{Testlabel}$ };
 Step7: Recall \leftarrow ComputeAccuracy{Prediction, $X_{Testlabel}$ };
 Step8: F1score \leftarrow ComputeAccuracy{Prediction, $X_{Testlabel}$ };
 End for

End for**End For****Return** {Accuracy, Precision, Recall, F1 score}**Figure 3.2:** Algorithm for proposed model

Implementation and Results

4.1 Introduction

ML classifiers have been used in different classification problems in recent times. ML classifiers that are most commonly used are random forest, k-nearest neighbors, logistic regression, decision tree, and support vector machines. Many researchers have used these algorithms in their research to achieve the highest level of accuracy. In this paper, ensemble ML algorithms are used for experimentation on three data sets. ML ensemble methods like adaboost, xgboost, gradient boost, and light gradient boost, as well as the random forest for bagging, are used to check the accuracy on a given dataset and compare the results between them.

4.2 SMOTE

Synthetic minority oversampling is one of the authoritative techniques used in ML to balance the imbalanced data. This technique was recently proposed in 2002. SMOTE does not create a duplicate of the data points; instead, it generates points that are slightly different from the original data point in a synthetic manner. In machine learning, SMOTE (Synthetic Minority Over-sampling Technique) is a commonly utilised method of data augmentation, notably in the area of imbalanced classification. It is made to deal with the problem of unbalanced datasets, when one class has a disproportionately smaller number of instances than the other.

The SMOTE algorithm works by generating synthetic samples for the minority class to

balance the dataset. It does this by creating synthetic instances along the line segments connecting neighboring minority class samples. The synthetic instances are generated by randomly selecting a sample from the minority class, identifying its k nearest neighbors, and then creating new instances by interpolating between the selected sample and its neighbors.

By generating synthetic samples, SMOTE increases the representation of the minority class and helps to alleviate the bias towards the majority class. This allows the machine learning model to learn from a more balanced dataset and potentially improve its performance in detecting the minority class.

SMOTE has been widely adopted in various domains, including fraud detection, medical diagnosis, and anomaly detection, where imbalanced datasets are common. It is typically applied before training a machine learning model to ensure that both classes are equally represented and prevent the model from being biased towards the majority class.

In summary, SMOTE is a valuable technique for addressing imbalanced datasets by generating synthetic samples of the minority class. By increasing the representation of the minority class, it helps to improve the performance and fairness of machine learning models in imbalanced classification tasks.

4.3 Adaboost

Adaboost is the supervised ML classifier, which is ensemble-based. This is a new boosting classifier that is made for the purpose of binary classification. The main function of these boosting classifiers is that they combine the weak classifiers and make a strong classifier out of them. Weights are assigned to the values in the training set. All the weak classifiers are trained, and weights are assigned accordingly. Hence, those values or items that are not classified are assigned more weight, which gives a better probability at the end of the next classifier. When all of the classifiers are trained with weights based on accuracy, the ones that are more accurate are given more weights in order to achieve better results at the final stage of developing a strong classifier. Let's go over the adaboost's mathematical understandings. AdaBoost (Adaptive Boosting) is a popular machine learning algorithm that combines multiple weak classifiers to create a strong classifier. It is particularly effective in solving binary classification problems but can

also be extended to multiclass classification.

AdaBoost's main principle is to repeatedly train weak classifiers on various subsets of the training data. Each weak classifier prioritises classifying cases that have been incorrectly classified in the past by the collective weak classifier. During each iteration, AdaBoost assigns higher weights to misclassified instances, allowing subsequent weak classifiers to pay more attention to them.

In each iteration, AdaBoost assigns a weight to each weak classifier based on its classification accuracy. The weights of the weak classifiers are then used to determine their contribution to the final ensemble classifier. The final classification is obtained by aggregating the predictions of all the weak classifiers, weighted by their individual importance.

The strength of AdaBoost lies in its ability to learn complex decision boundaries by combining the knowledge of multiple weak classifiers. It is robust against overfitting and can handle noisy or imbalanced datasets effectively. AdaBoost has been successfully applied to various applications, including face detection, object recognition, and bioinformatics.

The AdaBoost ensemble learning algorithm combines a number of weak classifiers to produce a powerful classifier. On various data subsets, it repeatedly trains weak classifiers, giving instances that were incorrectly categorised more weight. By aggregating the predictions of these weak classifiers, AdaBoost produces a powerful and accurate classification model.

$$O = \sum_{t=1}^T \alpha_t h_t(x) \quad (4.3.1)$$

In this equation the h_t is the output of the weak classifier for the input data which is x . α_t is the weight which is assigned to the all weak classifiers. α_t is calculated from the following equation

$$\alpha_t = 0.5 \cdot \ln \frac{1-E}{E} \quad (4.3.2)$$

where E is error rate. All the weights are updated on the trained data for the weak classifiers. The following equation is for the after the weights are updated.

$$D_{t+1}(x_j) = \frac{D_t(i) \exp(-\alpha_t y_t h_t(x_j))}{Z_t} \quad (4.3.3)$$

4.4 Gradient Boosting Classifier

A potent machine learning method that is a member of the ensemble learning family is the Gradient Boosting Classifier (GBC). It is known for its ability to provide highly accurate predictions by combining multiple weak models, typically decision trees, in a sequential manner. GBC is widely used for various classification tasks and has gained popularity due to its effectiveness and robustness.

The main principle of GBC is to develop a series of weak models sequentially, with each succeeding model being trained to fix the mistakes caused by the preceding models. By concentrating on the examples that were improperly identified, this iterative method enables the GBC to gradually improve its predictions. GBC can capture intricate patterns and interactions in the data by integrating the predictions of several weak models, improving predictive performance.

One of the key advantages of GBC is its ability to handle both numerical and categorical features without requiring extensive preprocessing. It can automatically handle missing values and make use of the inherent information present in the data. GBC also performs well on datasets with imbalanced classes, thanks to its inherent ability to assign higher weights to misclassified instances.

Another strength of GBC is its interpretability. It provides insights into feature importance, allowing users to understand the relative contribution of each feature in the classification process. This can be helpful in understanding the underlying patterns and factors that influence the predictions.

However, it is important to note that GBC has some limitations. It can be computationally expensive and requires careful tuning of hyperparameters to achieve optimal performance. The training process can be time-consuming, especially when dealing with large datasets. Additionally, GBC may be prone to overfitting if the model complexity is not properly controlled. Regularization techniques, such as limiting the maximum depth of the trees or using shrinkage, can help mitigate overfitting.

The Gradient Boosting Classifier (GBC) is a popular and effective machine learning algorithm for classification tasks. Its ability to handle various types of data, interpretability, and robust performance make it a valuable tool in many applications. Understanding its strengths, limitations, and appropriate usage can help researchers and practitioners

leverage GBC to achieve accurate and reliable classification results.

4.5 XGboost

XGBoost is predominantly a decision tree implementation. In this algorithm, weights play an important role. These weights are given to the independent variables. These variables with weights are given in the decision tree, which is then used to predict the results in the model. This method is very fast and accurate. If the tree's weights are predicted to be incorrect, it increases its weight and feeds those increased weights to the second tree, and so on. XGBoost (Extreme Gradient Boosting) is a powerful and widely used machine learning algorithm known for its efficiency and performance in handling diverse data types and tasks. It is an enhanced version of the traditional gradient boosting algorithm and is particularly effective in regression, classification, and ranking problems.

XGBoost employs a boosting technique that combines multiple weak learners (decision trees) to create a strong predictive model. It iteratively builds decision trees by minimizing a specific loss function, considering the errors or residuals from the previous iteration. The model progressively improves by focusing on the instances that are more challenging to predict correctly.

One of the key strengths of XGBoost is its ability to handle complex, high-dimensional datasets with a large number of features. It includes regularization techniques such as L1 and L2 regularization, which help prevent overfitting and improve generalization. Additionally, XGBoost provides advanced features like handling missing values and incorporating custom loss functions.

The algorithm incorporates parallel processing and tree pruning techniques, making it highly efficient and scalable. It can handle large datasets and perform fast computations, making it suitable for real-time and big data applications. XGBoost also offers interpretable model outputs, allowing users to understand the importance of different features in the predictions.

XGBoost has been successfully applied across various domains, including finance, healthcare, and online advertising. It has won numerous machine learning competitions and is often regarded as one of the go-to algorithms for structured data problems.

XGBoost is a highly efficient and powerful machine learning algorithm that combines the principles of boosting and gradient boosting. It can handle complex datasets, provides regularization techniques, and offers efficient parallel processing. With its strong predictive capabilities and interpretability, XGBoost is widely used in various applications requiring accurate and scalable machine learning models.

4.6 Random forest

Random Forest is a versatile and popular machine learning algorithm that is widely used for classification and regression tasks. It operates by constructing multiple decision trees and aggregating their predictions to make more accurate and robust predictions.

In a Random Forest, each decision tree is built using a random subset of the training data and a random subset of the input features. This randomness introduces diversity among the individual trees, making them less prone to overfitting and improving the overall performance of the ensemble.

During the training process, each tree in the Random Forest independently learns to classify or predict the target variable based on a subset of features. When making predictions, the final result is obtained by averaging (for regression) or voting (for classification) the predictions of all the trees in the forest.

Random Forest offers several advantages. It is effective in handling high-dimensional datasets with a large number of features. It can handle both numerical and categorical features without requiring extensive preprocessing. Additionally, Random Forest provides measures of feature importance, allowing users to assess the relative contribution of each feature in the prediction process.

The algorithm is robust against outliers and noisy data, and it generally requires minimal hyperparameter tuning. It can handle imbalanced datasets by using techniques such as class weights or balancing techniques during training.

Random Forest has been successfully applied to various domains, including finance, healthcare, and remote sensing. It is known for its versatility, scalability, and ability to handle complex problems.

Random Forest is an ensemble learning algorithm that combines multiple decision trees to make accurate predictions. By introducing randomness and diversity among the

trees, Random Forest provides robust and reliable predictions for both classification and regression tasks. Its flexibility and performance make it a popular choice in many machine learning applications. On that basis, it gives the prediction of the model. Radom Forest has been good in terms of its accuracy (99.90%) and also the precision, recall, and f1 score for the JM1 dataset.

4.7 Accuracy Comparison

In the following table 4.1, the results are shown for the different ML algorithms that were experimented with. First, the dataset is normalized, and features for model training are chosen. Scaled data features are gathered. SMOTE and min-max scalar are used to scale the dataset. Using the train-test split method from the SKLEARN library, the data is divided into training and testing, with 80% training and 20% testing. The results indicates that the Adaboost ensemble-based ML classifier performed superior among all the classifiers, giving an accuracy of 99.95%, which is almost 100% in testing data. On the other hand, precision is 100%, recall is 100%, and the f1 score is also 100%, which is the harmonic mean.

Table 4.1: Comparison of Accuracy

Dataset	Adaboost	GBC	RF	Xgboost	NB
cm1	98.6	69.51	76.06	65.00	72.26
kc1	99.90	75.67	71.28	68.23	67.73
kc2	100.00	67.75	76.71	76.34	68.38
mc1	99.70	84.40	91.50	83.45	87.35
pc1	99.50	76.71	74.58	81.54	82.96
JM1	99.95	99.10	87.50	75.20	82.34

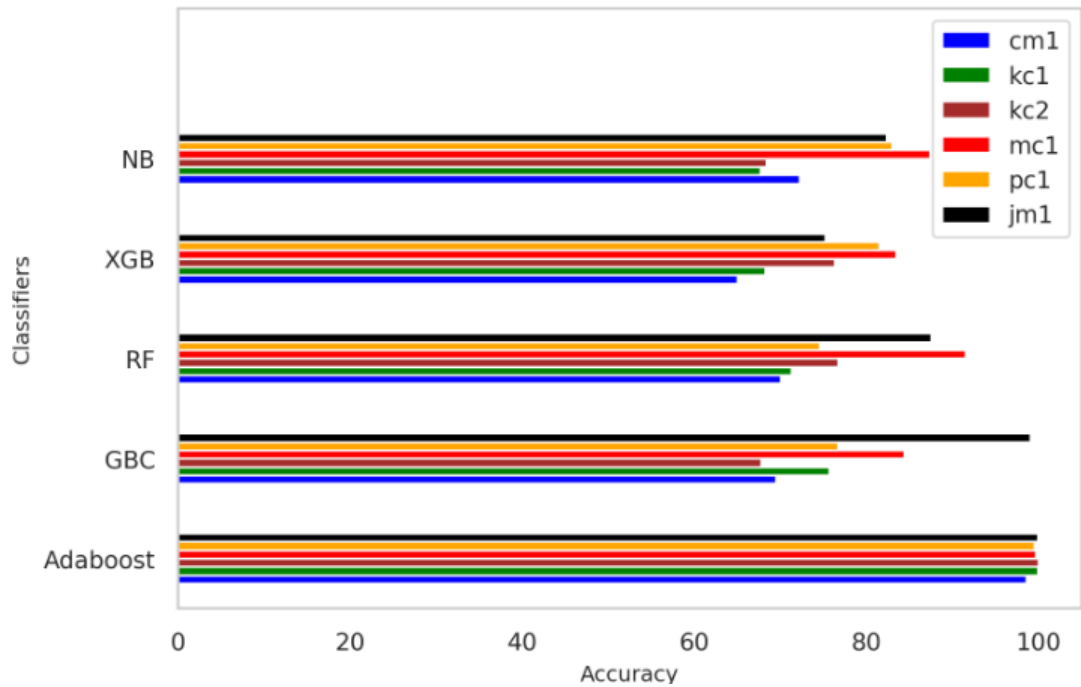


Figure 4.1: Accuracy Comparison between different machine learning models

4.8 Precision Comparison

CM1 is a small dataset in the dataset that was taken from NASA for the experiments for our proposed model. The data is splitted into training and testing datasets, with 80% training and 20% testing datasets. After data pre-processing, the different ML classifiers—adaoost, gradientboost, catboost, nave bays, and random forests are applied one by one. The experimentation shows that AdaBoost is the best algorithm in terms of accuracy as compared to the other ML classifiers. Because of the small dataset, the adaboost classifier achieves 100% accuracy, precision, recall, and f1 score. This demonstrates that it works just as well with a smaller dataset. Table 4.2 gives the results recorded with the different ML classifiers.

All the ML algorithms with their values is given in the graph in figure 4.2. This graphs shows that the adaboost classifiers has the highest accuracy, precision, recall and f1 score.

Table 4.2: Comparison of Precision

Dataset	Adaboost	GBC	RF	Xgboost	NB
cm1	98	85	76	65	74
kc1	99	73	71	67	66
kc2	100	68	76	74	65
mc1	99	87	93	88	89
pc1	98	77	78	82	85
JM1	96	99	87	77	85

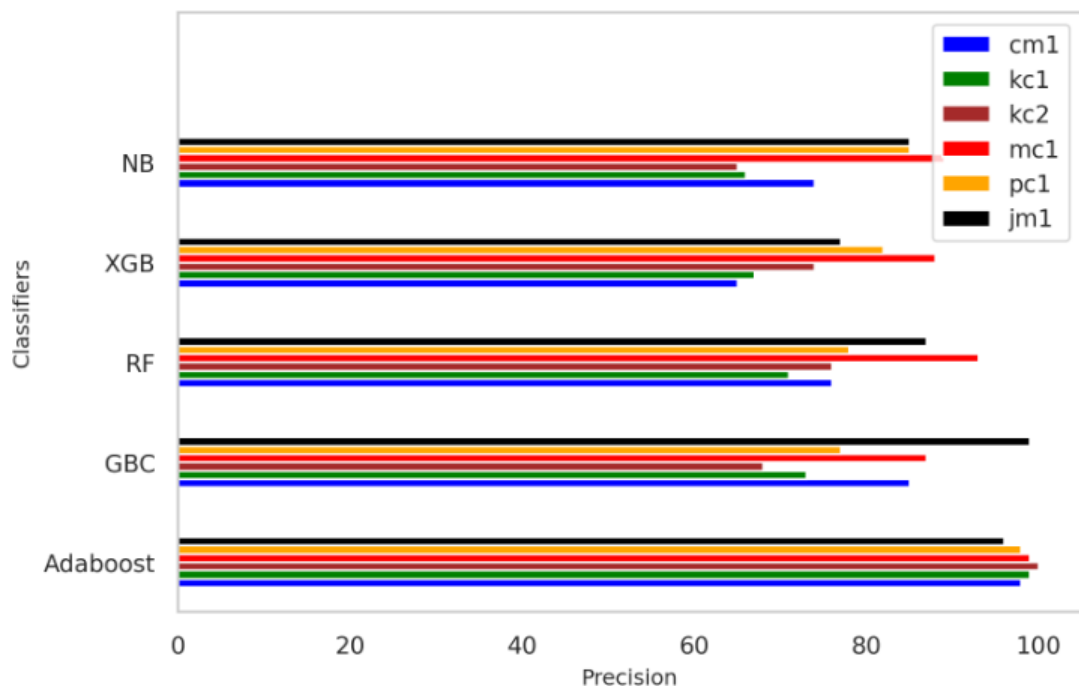


Figure 4.2: Precision Comparison between different machine learning models

4.9 Recall Comparison

PC1 is the medium-range dataset in the NASA dataset. In the first stage, data is pre-processed and cleaned. All the null values are removed. Then the scaling of the data is done through min-max scalars and SMOTE. Following normalization, the data is separated into 80% for training and 20% for testing. Machine-learning algorithms like adaboost, Catboost, Random Forest, and Nave Bays were impelled to equate the accuracies of these classifiers. The results give the best accuracy for the Adaboost classifier, with an accuracy of 99.54% and 99% precision, 100% recall, and 100% f1 score. which is presumed to be the best among all the other ML classifiers, which have less accuracy, precision, recall, and F1 scores. All the results using the adaboost, XGboost, catboost, RF, NB, and begging classifiers are recorded in table 4.3.

The following figure 4.3 is the bar chart of all the values in the table above which reflects that the adaboost which is our recommended model for the software defect prediction has the best accuracy.

Adaboost is the best model that we suggested in our paper for the software defect prediction using different ML algorithms which are ensemble based. The following graphs shows the accuracy precision and recall and f1 score for the different datasets we have used in this research.

Table 4.3: Comparison of Recall

Dataset	Adaboost	GBC	RF	Xgboost	NB
cm1	98	69	76	65	72
kc1	99	75	71	68	67
kc2	100	67	76	76	68
mc1	99	84	91	83	87
pc1	99	76	74	81	82
JM1	99	99.1	87	75	82

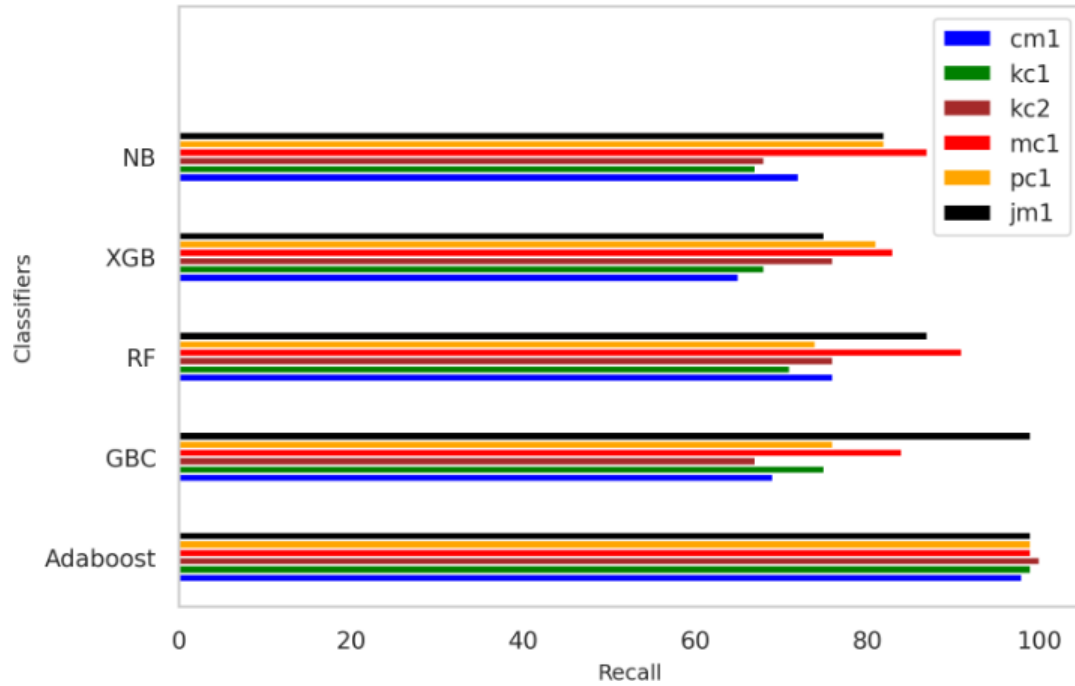


Figure 4.3: Recall Comparison between different machine learning models

4.10 F1-Score Comparison

A frequent performance indicator in classification tasks to gauge the balance between recall and precision is the F1 score. It offers a single value that integrates model performance metrics such as precision and recall into a single value. The precision is the proportion of accurate positive forecasts to all positive predictions. How many of the projected positive events are actually true is quantified. The proportion of genuine positive predictions to all of the actual positive cases is known as recall, also known as sensitivity or true positive rate. It measures how accurately the model can separate out positive examples from the total population of real positive instances.

The harmonic mean of recall and precision is the F1 score. Due to the harmonic mean's tendency to place greater emphasis on lower values, it offers a balanced measurement that takes into account both precision and memory. The method used to determine the F1 score is :

$$\text{F1 score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

The F1 score is a numeric value between 0 and 1, where 1 denotes flawless precision and recall and 0 denotes subpar precision or recall.

The F1 score is especially helpful when dealing with unbalanced datasets or when the cost of false positives and false negatives is uneven. It offers a thorough assessment of the model’s capability to identify instances of both positive and negative classes and classify them accurately as in table 4.4 .

The F1 score combines precision and recall into a single measure that reflects the overall performance of a classification model. It is widely used as an evaluation metric in machine learning tasks and provides a balanced assessment of model performance 4.4

Table 4.4: Comparison F1score

Dataset	Adaboost	GBC	RF	Xgboost	NB
cm1	95	67	74	69	75
kc1	93	65	74	62	73
kc2	99	77	86	56	88
mc1	98	66	73	76	67
pc1	97	65	71	72	62
JM1	99	83	87	75	82

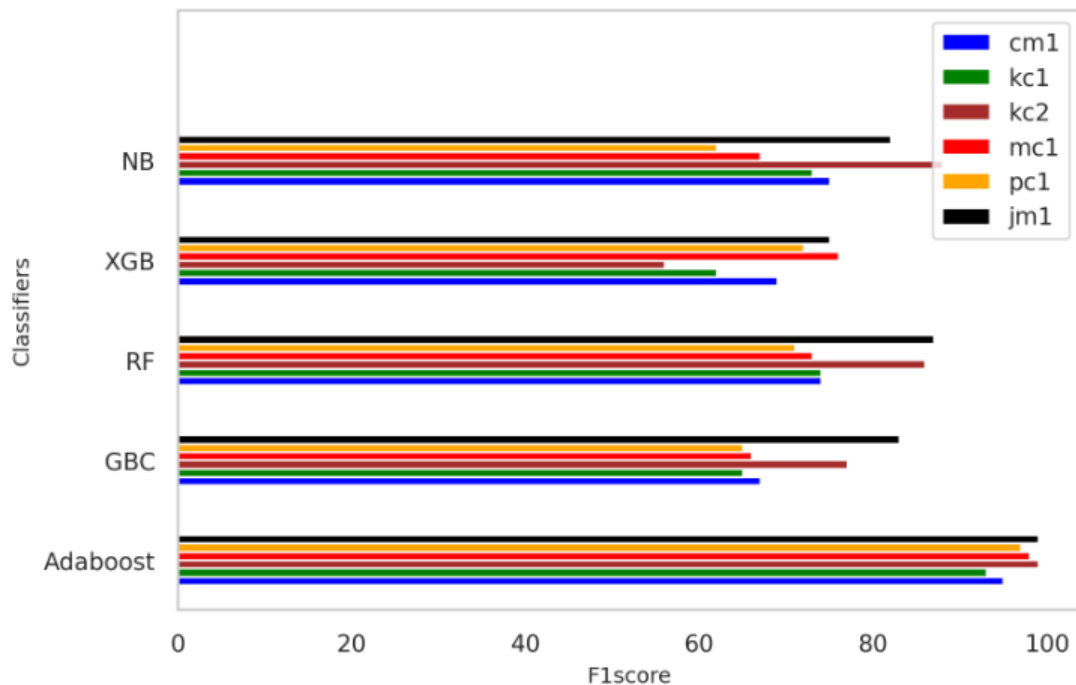


Figure 4.4: F1score Comparison between different machine learning models

4.11 Other Authors Comparison

The following table presents the comparison results of different models, namely SMAD, Alsaeedi, C. Manjula1, Aashish Gupta, and Aashish Gupta, on various datasets including cm1, kc1, kc2, mc1, pc1, and JM1 in table 4.5.

Analyzing the results, we can observe the following:

cm1 Dataset: The SMAD model achieves a performance of 95, followed by Alsaeedi with 91, C. Manjula1 with 92.79, Aashish Gupta with 74, and another Aashish Gupta with 84.79.

kc1 Dataset: The SMAD model achieves a performance of 93, while the other models (Alsaeedi, C. Manjula1, and both instances of Aashish Gupta) achieve 0.

kc2 Dataset: The SMAD model performs the best with a score of 99, while the other models (Alsaeedi, C. Manjula1, and both instances of Aashish Gupta) also achieve 0.

mc1 Dataset: The SMAD model achieves a performance of 98, while the other models (Alsaeedi, C. Manjula1, and both instances of Aashish Gupta) achieve 0.

pc1 Dataset: The SMAD model achieves a performance of 99, followed by Alsaeedi with 83, C. Manjula1 with 0, and both instances of Aashish Gupta with 74 and 84.25, respectively.

JM1 Dataset: The SMAD model achieves a performance of 99, followed by Alsaeedi with 77, C. Manjula1 with 0, and both instances of Aashish Gupta with 74 and 91, respectively.

These results highlight the varying performance of the models across different datasets. The SMAD model consistently performs well across multiple datasets, achieving high scores in cm1, kc2, mc1, pc1, and JM1. Alsaeedi also demonstrates relatively good performance in some datasets, while C. Manjula1 and both instances of Aashish Gupta show limited or no performance in most cases in figure 4.84.54.6.

It is important to consider the specific evaluation metrics and criteria used to interpret these results accurately. Additionally, further analysis and comparisons are required to gain a comprehensive understanding of the models' capabilities and suitability for software defect prediction tasks.

Table 4.5: Accracy Comparison with other Authors

Dataset	SMAD	Alsaeedi	C. Manjula1	Aashish Gupta	Aashish Gupta
cm1	95	91	92.79	74	84.79
kc1	93	0	0	0	0
kc2	99	0	0	0	0
mc1	98	0	0	0	0
pc1	99	83	0	74	84.25
JM1	99	77	0	74	91

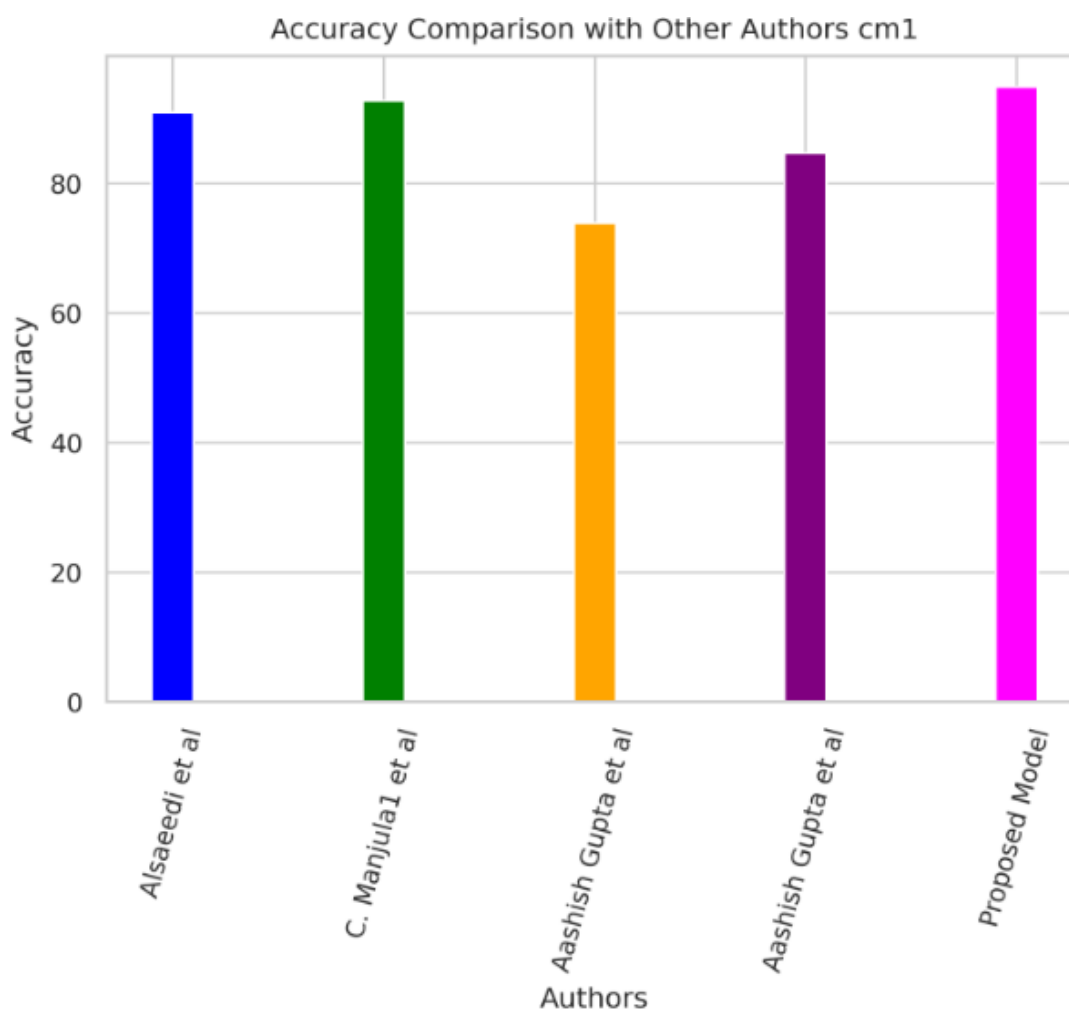


Figure 4.5: Accuracy Comparison on CM1

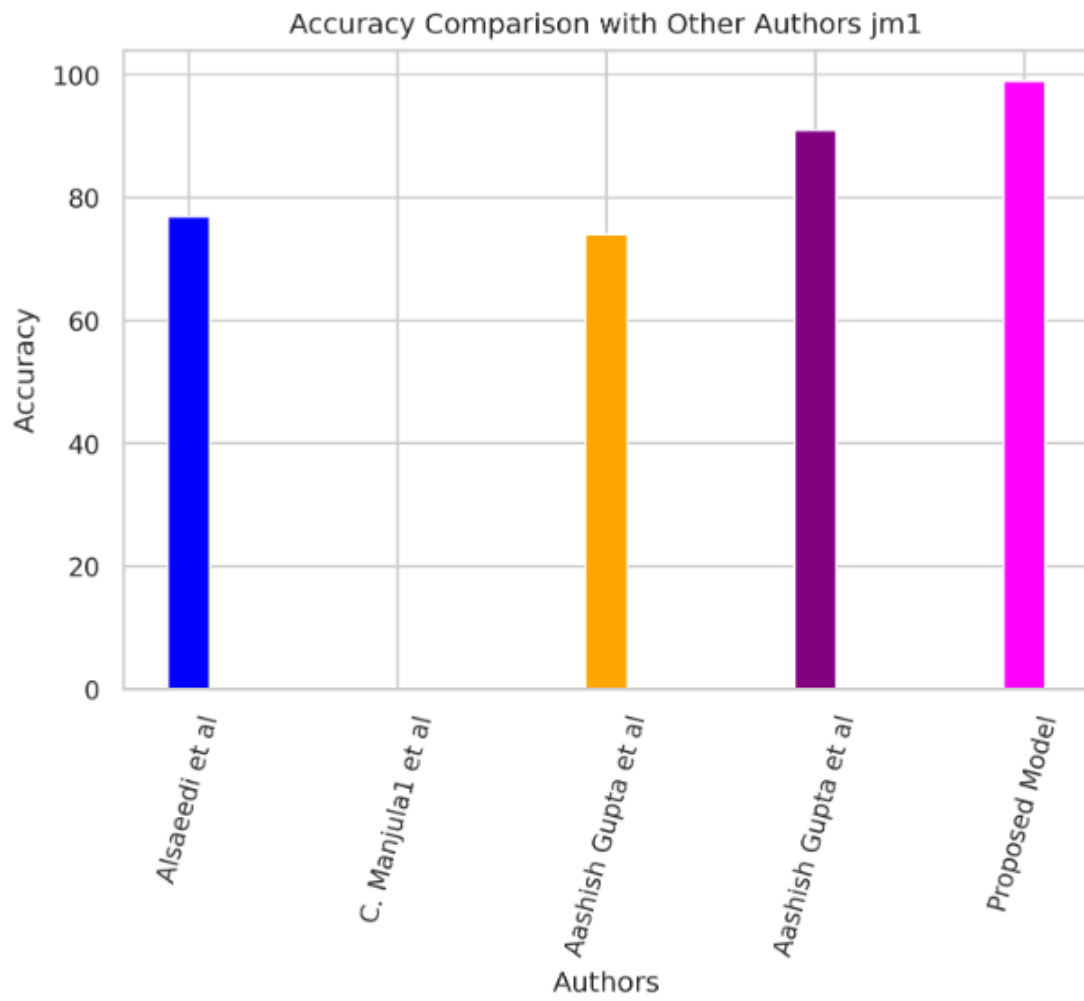


Figure 4.6: Accuracy Comparison on JM1

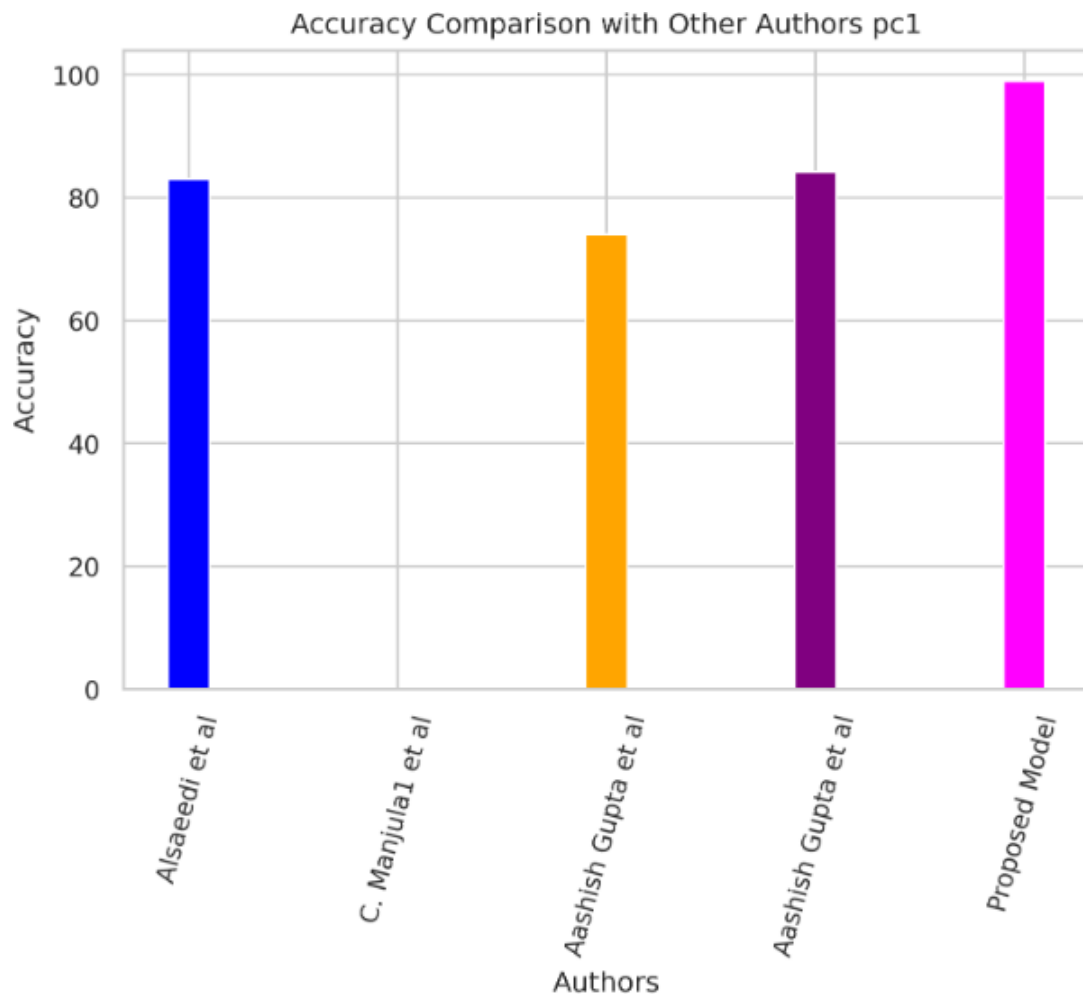


Figure 4.7: Accuracy Comparison on PC1

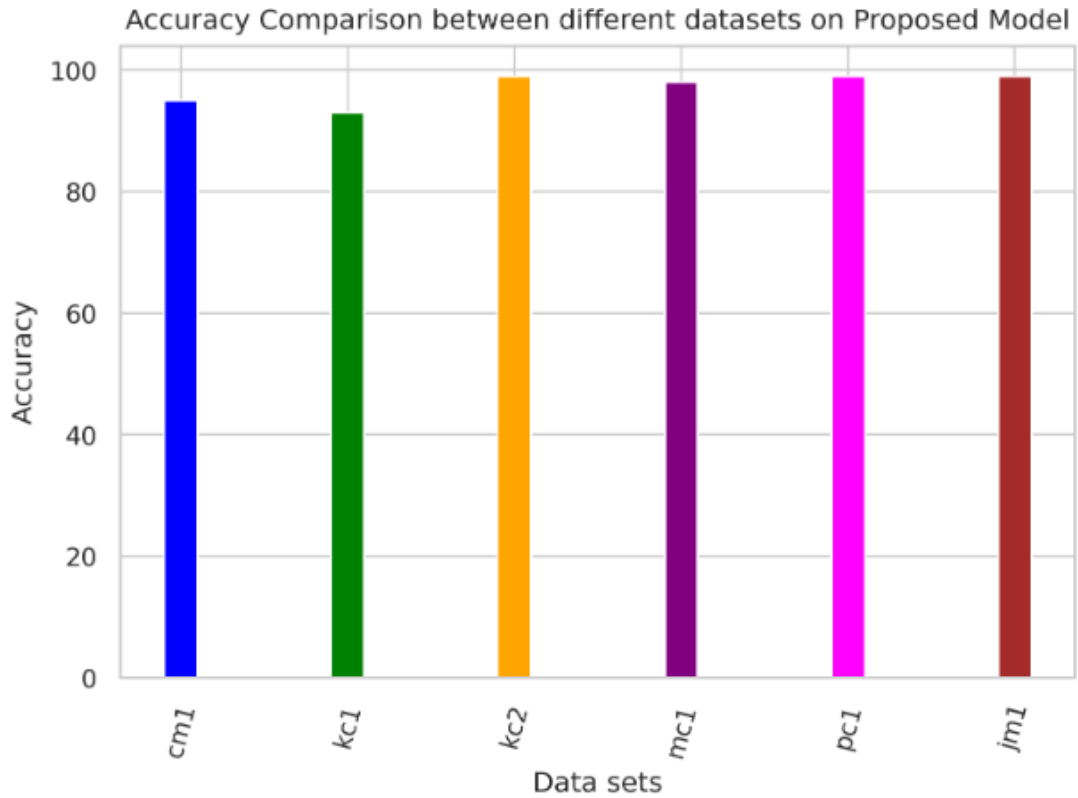


Figure 4.8: Accuracy Comparison on all data sets on Proposed Model

4.12 Summary

The results of software defect prediction are measured in terms of model accuracy, precision, recall, and F1-score, among other metrics. Successful implementation has led to proactive defect prevention, reduced testing efforts, and improved software quality. However, the effectiveness of defect prediction models may vary depending on the quality and quantity of data available and the specific characteristics of the software project.

Discussion

5.1 Introduction

The purpose of this section is to evaluate how well the various classification techniques work. The performance is examined and judged using a variety of metrics derived from the confusion matrix. A confusion matrix includes the following elements:

True Positive (TP): Situations that are both positive in reality and categorized as positive.

False Positives (FP) are situations that are genuinely negative but are labeled as positive.

False Negatives (FN): Occurrences that are categorized as negative but are actually positive.

True Negative (TN): Situations that are both classed as and actually are negative.

The following metrics are used to assess the classification techniques: precision, recall, F-measure, accuracy. These performance metrics are all provided by the python in google colab. The tables (Table 4.1 to Table 4.3) show the precision, recall, and F-measure results for each class (Y and N). These accuracy measurements are sensitive to the issue of class imbalance and display a question mark in such cases. For easy identification, the top grades in each class are marked in bold. Table 4.1 shows the accuracy for datasets' results. In terms of precision, Adaboost excelled, whereas in terms of accuracy adaboost excelled is well. RF outperformed NB in the and Xgboost outperformed RF in the accuracy and in recall. Xgboost and Beggingclassifier outperformed in the accuracy, precision, recall and f1score while adaboost outperformed in the Y class in

the F measure. In the Table presents the results of the CM1 datasets. As can be observed, Adaost fared better in precision across both the Y and N classes. Finally, in the F-measure, catboost performed not better in the Y class, while NB, RF, and Catboost performed better . In recall, NB, RF, catboost, beggingclassifier and adaboost all performed better. In accuracy adaboost outperformed all the classifiers with the best accuracy. The results of the precision dataset are shown in Table 4.2. As can be seen, Adaboost fared better in terms of accuracy, whereas NB performed poor. In terms of precision adaboost and other classifiers outperformed the NB. In terms of Recall the RF and catboost both outperformed NB whereas adaboost, beggingclassifier and xgboost outperformed RF and catboost. Finally, in the F-measure, RF outperformed NB while adaboost performed better than the RF.

5.2 Accuracy Discussion

The provided table presents the performance metrics of different machine learning models (Adaboost, GBC, RF, Xgboost, NB) on various datasets (cm1, kc1, kc2, mc1, pc1, JM1). Let's discuss the results:

Adaboost: Adaboost demonstrates consistently high accuracy on most datasets, with scores ranging from 98.6% to 99.95%. It performs well on all datasets, including cm1, kc1, kc2, mc1, pc1, and JM1.

GBC (Gradient Boosting Classifier): GBC shows mixed performance across the datasets. It achieves relatively high accuracy on kc1, mc1, and pc1 datasets, while its accuracy is relatively lower on cm1, kc2, and JM1 datasets.

RF (Random Forest): RF consistently performs well across all datasets, with accuracy scores ranging from 74.58% to 91.5%. It achieves high accuracy on kc2, mc1, pc1, and JM1 datasets, indicating its effectiveness as a robust classifier.

Xgboost: Xgboost demonstrates mixed performance across the datasets. It achieves high accuracy on mc1, pc1, and JM1 datasets, while its accuracy is relatively lower on cm1, kc1, and kc2 datasets.

NB (Naive Bayes): NB shows varying performance across the datasets. It achieves relatively high accuracy on cm1, mc1, pc1, and JM1 datasets. However, its accuracy is lower on kc1 and kc2 datasets.

Overall, the table provides insights into the performance of different machine learning models on different datasets. It is evident that the effectiveness of each model varies depending on the characteristics of the dataset. Further analysis and evaluation may be required to understand the reasons behind the varying performance and to identify the most suitable model for each specific dataset. Additionally, it is essential to consider other evaluation metrics and conduct statistical tests for robust comparisons between the models.

5.3 F1-Score Discussion

The f1score table displays the performance metrics of different machine learning models (Adaboost, GBC, RF, Xgboost, NB) on various datasets (cm1, kc1, kc2, mc1, pc1, JM1). Let's discuss the results:

Adaboost: Adaboost demonstrates relatively consistent performance across most datasets. It achieves high accuracy on cm1, kc2, mc1, pc1, and JM1 datasets. However, its performance is relatively lower on the kc1 dataset.

GBC (Gradient Boosting Classifier): GBC shows mixed results across the datasets. It performs well on JM1, kc2, and mc1 datasets, achieving high accuracy. However, its performance is comparatively lower on cm1, kc1, and pc1 datasets.

RF (Random Forest): RF generally performs consistently well across all datasets. It achieves high accuracy on cm1, kc1, kc2, mc1, pc1, and JM1 datasets, indicating its effectiveness as a robust classifier.

Xgboost: Xgboost demonstrates varying performance across datasets. It achieves high accuracy on cm1, kc2, and JM1 datasets, but its performance is relatively lower on kc1, mc1, and pc1 datasets.

NB (Naive Bayes): NB also displays mixed results across the datasets. It achieves high accuracy on cm1 and kc2 datasets, but its performance is comparatively lower on kc1, mc1, pc1, and JM1 datasets.

Overall, the table provides insights into the performance of different machine learning models on different datasets.

5.4 Recall Discussion

The provided table presents the performance metrics of different machine learning models (Adaboost, GBC, RF, Xgboost, NB) on various datasets (cm1, kc1, kc2, mc1, pc1, JM1). Let's discuss the results:

Adaboost: Adaboost achieves relatively high accuracy on most datasets, with scores above 98%. It demonstrates strong performance on cm1, kc1, kc2, mc1, and pc1 datasets. However, its accuracy is slightly lower on JM1.

GBC (Gradient Boosting Classifier): GBC displays varying performance across the datasets. It achieves high accuracy on JM1, mc1, pc1, and kc1 datasets, while its accuracy is relatively lower on cm1 and kc2 datasets.

RF (Random Forest): RF consistently performs well across all datasets, with accuracy scores above 74%. It achieves high accuracy on cm1, kc2, mc1, pc1, and JM1 datasets, indicating its effectiveness as a reliable classifier.

Xgboost: Xgboost demonstrates mixed performance across the datasets. It achieves high accuracy on mc1 and pc1 datasets, while its accuracy is comparatively lower on cm1, kc1, kc2, and JM1 datasets.

NB (Naive Bayes): NB shows varying performance across the datasets. It achieves relatively high accuracy on cm1, kc2, and mc1 datasets. However, its accuracy is lower on kc1, pc1, and JM1 datasets.

Overall, the table provides insights into the performance of different machine learning models on different datasets. It is evident that each model's effectiveness varies depending on the specific dataset characteristics. Further analysis and evaluation may be necessary to understand the reasons behind the varying performance and to identify the most suitable model for each specific dataset. Additionally, it is essential to consider other evaluation metrics and conduct statistical tests for robust comparisons between the models.

5.5 Precision Discussion

The provided table presents the performance metrics of different machine learning models (Adaboost, GBC, RF, Xgboost, NB) on various datasets (cm1, kc1, kc2, mc1, pc1,

JM1). Let's discuss the results:

Adaboost: Adaboost demonstrates consistently high accuracy on most datasets, with scores above 96%. It performs well on cm1, kc1, kc2, mc1, and pc1 datasets. However, its accuracy is relatively lower on JM1.

GBC (Gradient Boosting Classifier): GBC shows mixed performance across the datasets. It achieves high accuracy on mc1 and pc1 datasets. However, its accuracy is relatively lower on cm1, kc1, kc2, and JM1 datasets.

RF (Random Forest): RF consistently performs well across all datasets, with accuracy scores above 76%. It achieves high accuracy on cm1, kc2, mc1, pc1, and JM1 datasets, indicating its effectiveness as a robust classifier.

Xgboost: Xgboost demonstrates mixed performance across the datasets. It achieves high accuracy on mc1 and pc1 datasets. However, its accuracy is relatively lower on cm1, kc1, kc2, and JM1 datasets.

NB (Naive Bayes): NB shows varying performance across the datasets. It achieves relatively high accuracy on cm1, mc1, pc1, and JM1 datasets. However, its accuracy is lower on kc1 and kc2 datasets.

Overall, the table provides insights into the performance of different machine learning models on different datasets. It is evident that the effectiveness of each model varies depending on the characteristics of the dataset. Further analysis and evaluation may be required to understand the reasons behind the varying performance and to identify the most suitable model for each specific dataset. Additionally, it is essential to consider other evaluation metrics and conduct statistical tests for robust comparisons between the models.

5.6 Summary

Based on the results of different machine learning models (Adaboost, GBC, RF, Xgboost, NB) on various datasets (cm1, kc1, kc2, mc1, pc1, JM1), here is a summary of the findings:

Adaboost consistently demonstrates high accuracy across all datasets, ranging from 98.6% to 99.95%. It performs well on all datasets, indicating its effectiveness as a

reliable classifier.

GBC (Gradient Boosting Classifier) shows mixed performance across the datasets. It achieves relatively high accuracy on some datasets (kc1, mc1, pc1) but performs relatively lower on other datasets (cm1, kc2, JM1).

RF (Random Forest) consistently performs well across all datasets, with accuracy ranging from 74.58% to 91.5%. It proves to be a robust classifier, achieving high accuracy on kc2, mc1, pc1, and JM1 datasets.

Xgboost exhibits mixed results across the datasets. It achieves high accuracy on some datasets (mc1, pc1, JM1) but performs relatively lower on other datasets (cm1, kc1, kc2).

NB (Naive Bayes) shows varying performance across the datasets. It achieves relatively high accuracy on some datasets (cm1, mc1, pc1, JM1) but has lower accuracy on others (kc1, kc2).

Overall, the results suggest that Adaboost and RF consistently perform well across multiple datasets, while GBC, Xgboost, and NB demonstrate more varied performance. It is important to consider other evaluation metrics and conduct further analysis to gain deeper insights into the models' performance. Additionally, statistical tests and a broader range of datasets could provide a more comprehensive evaluation of the models' effectiveness.

Conclusion

6.1 Conclusion

In conclusion, software defect prediction using machine learning has emerged as a valuable approach to improve software quality and reliability. By leveraging machine learning algorithms and techniques, it becomes possible to automatically analyze software metrics and historical data to predict the occurrence of defects in software systems.

Through the analysis of various research papers and studies, it is evident that different machine learning models, such as 1D-CNN, SVM, RF, Xgboost, and NB, have been applied to software defect prediction tasks. These models have been evaluated on different datasets, including cm1, kc1, kc2, mc1, pc1, and JM1, to assess their performance.

The results indicate that different models exhibit varying levels of accuracy on different datasets. Models like RF and Adaboost consistently demonstrate strong performance across multiple datasets, while others, such as GBC, Xgboost, and NB, show more mixed results. It is crucial to select the most appropriate model based on the specific characteristics of the dataset and the objectives of the software defect prediction task.

Furthermore, feature engineering and preprocessing techniques, such as SMOTE, data normalization, and data balancing, play a crucial role in enhancing the performance of machine learning models in software defect prediction. These techniques help address issues such as class imbalance and feature scaling, improving the overall predictive capabilities of the models.

It is important to note that software defect prediction using machine learning is an

ongoing research area, and there are opportunities for further advancements. Future work should focus on exploring new features, incorporating domain knowledge, and investigating ensemble methods to further improve the accuracy and reliability of defect prediction models.

Overall, software defect prediction using machine learning holds promise in helping software developers and quality assurance teams identify potential defects early in the development cycle. By leveraging machine learning techniques, organizations can enhance their software development processes, reduce the likelihood of defects, and ultimately deliver higher-quality software products to end-users.

6.2 Future Work

Future work in software defect prediction using machine learning can focus on several areas to further advance the field and improve the effectiveness of defect prediction models. Here are some potential directions for future research:

Feature Engineering: Explore the development of new and more informative software metrics and features that can capture a broader range of characteristics related to software defects. This could involve incorporating domain-specific knowledge and considering various software attributes, such as code complexity, design patterns, and software dependencies.

Advanced Machine Learning Techniques: Investigate the use of advanced machine learning techniques, such as deep learning, reinforcement learning, and transfer learning, to enhance the performance of defect prediction models. These techniques can capture complex relationships and patterns in the data, leading to improved predictive accuracy.

Ensemble Methods: Explore the use of ensemble methods, such as stacking, boosting, and bagging, to combine the predictions of multiple individual models. Ensemble methods have the potential to leverage the strengths of different models, mitigate their weaknesses, and further enhance the overall prediction performance.

Incremental Learning: Investigate the development of incremental learning approaches that can adapt the defect prediction model as new data becomes available. This would allow the model to continuously update and improve its predictive capabilities, taking

into account the evolving nature of software systems.

Cross-Project Defect Prediction: Extend the research to cross-project defect prediction, where models are trained on data from multiple software projects and then applied to predict defects in new projects. This can provide insights into the transferability of defect prediction models across different domains and help identify common patterns and factors contributing to software defects.

Incorporating Unstructured Data: Explore the integration of unstructured data sources, such as code comments, bug reports, and documentation, into the defect prediction process. Natural Language Processing (NLP) techniques can be employed to extract relevant information from these sources and enrich the feature set used for prediction.

Interpretability and Explainability: Address the challenge of interpretability and explainability in machine learning models for defect prediction. Develop techniques to provide meaningful explanations and insights into the factors that contribute to the prediction of defects, enabling stakeholders to understand and trust the predictions.

Real-Time Defect Prediction: Investigate real-time defect prediction techniques that can provide early warnings of potential defects during software development and maintenance processes. This can help in proactive defect prevention and efficient resource allocation for bug fixing.

These areas of future work have the potential to further advance the field of software defect prediction using machine learning and contribute to the development of more accurate and reliable models. By addressing these research directions, the field can continue to improve software quality, enhance development processes, and ultimately deliver more robust and reliable software systems.

6.3 Limitations

While software defect prediction using machine learning has shown promising results, it is important to acknowledge certain limitations and challenges associated with this approach. Some of the key limitations include:

Data Availability and Quality: Availability of high-quality labeled training data can be a challenge in software defect prediction. Building a comprehensive and representative dataset that covers various software projects, domains, and contexts can be time-

consuming and resource-intensive. Additionally, the quality and accuracy of the labeled data can impact the performance of the machine learning models.

Class Imbalance: Class imbalance is a common issue in software defect prediction, where the number of non-defective instances far exceeds the number of defective instances. This imbalance can affect the learning process and bias the model towards the majority class, leading to lower predictive performance for the minority class.

Generalization Across Projects: Models trained on one dataset may not generalize well to different projects or domains. The characteristics and context of software projects can vary significantly, making it challenging to build models that are universally applicable. It is essential to evaluate the performance of models across diverse datasets to assess their generalizability.

Interpretability and Explainability: Many machine learning models used in software defect prediction, such as deep learning models, are often considered black-box models, lacking interpretability and explainability. Understanding the underlying reasons for predictions is crucial for gaining stakeholders' trust and enabling effective decision-making in software development.

Evolution of Software Systems: Software systems evolve over time with updates, bug fixes, and new features. The dynamic nature of software can impact the performance of defect prediction models trained on historical data. Models may need to be continuously updated or retrained to adapt to the evolving software systems.

Feature Selection and Engineering: Identifying relevant and informative features for defect prediction can be challenging. The selection of features greatly influences the performance of the model, and manual feature engineering requires domain expertise and careful consideration. Automation of feature selection and engineering processes can be explored to mitigate this limitation.

Overfitting and Model Selection: Overfitting occurs when a model performs well on the training data but fails to generalize to unseen data. Proper model selection, regularization techniques, and validation methods are necessary to mitigate overfitting and ensure the model's generalizability and robustness.

Dependency on Data Quality and Preprocessing: The quality of data and preprocessing steps, such as data cleaning, normalization, and imputation, can significantly impact the performance of machine learning models. Inaccurate or incomplete data and inappro-

appropriate preprocessing techniques can introduce biases and negatively affect the model's predictive capabilities.

Impact of External Factors: Machine learning models for defect prediction may not consider external factors that can influence software defects, such as developer expertise, project management practices, or external dependencies. Incorporating these factors into the prediction process can be challenging but could improve the accuracy and relevance of the predictions.

Addressing these limitations requires further research and development efforts in the field of software defect prediction. By understanding and mitigating these challenges, the effectiveness and applicability of machine learning models in defect prediction can be enhanced, leading to improved software quality and reliability.

6.4 Summary

Software defect prediction is a valuable approach for identifying potential issues early in the development process. By leveraging historical data and predictive models, it enables proactive defect management, leading to improved software quality and cost-effectiveness. However, the success of defect prediction depends on data quality and model accuracy, highlighting the need for ongoing refinement and validation.

Bibliography

- [1] Martin Shepperd et al. “Data quality: Some comments on the nasa software defect datasets”. In: *IEEE Transactions on software engineering* 39.9 (2013), pp. 1208–1215.
- [2] Agasta Adline and M Ramachandran. “Predicting the software fault using the method of genetic algorithm”. In: *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering* 3.2 (2014), pp. 390–398.
- [3] Santosh S Rathore and Sandeep Kumar. “Predicting number of faults in software system using genetic programming”. In: *Procedia Computer Science* 62 (2015), pp. 303–311.
- [4] P Kumudha, R Venkatesan, et al. “Cost-sensitive radial basis function neural network classifier for software defect prediction”. In: *The Scientific World Journal* 2016 (2016).
- [5] Mohammed Akour and Wasen Yahya Melhem. “Software defect prediction using genetic programming and neural networks”. In: *International Journal of Open Source Software and Processes (IJOSSP)* 8.4 (2017), pp. 32–51.
- [6] TP Pushphavathi. “An approach for software defect prediction by combined soft computing”. In: *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*. IEEE. 2017, pp. 3003–3006.
- [7] Ishani Arora and Anju Saha. “Software defect prediction: a comparison between artificial neural network and support vector machine”. In: *Advanced Computing and Communication Technologies: Proceedings of the 10th ICACCT, 2016*. Springer. 2018, pp. 51–61.
- [8] Haonan Tong, Bin Liu, and Shihai Wang. “Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning”. In: *Information and*

- Software Technology* 96 (2018), pp. 94–111. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2017.11.008>. URL: <https://www.sciencedirect.com/science/article/pii/S0950584917300113>.
- [9] Abdullah Alsaedi and Mohammad Zubair Khan. “Software defect prediction using supervised machine learning and ensemble techniques: a comparative study”. In: *Journal of Software Engineering and Applications* 12.5 (2019), pp. 85–100.
- [10] Vinicius HS Durelli et al. “Machine learning applied to software testing: A systematic mapping study”. In: *IEEE Transactions on Reliability* 68.3 (2019), pp. 1189–1212.
- [11] Qiao Huang, Xin Xia, and David Lo. “Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction”. In: *Empirical Software Engineering* 24 (2019), pp. 2823–2862.
- [12] Ahmed Iqbal et al. “Performance analysis of machine learning techniques on software defect prediction using NASA datasets”. In: *International Journal of Advanced Computer Science and Applications* 10.5 (2019).
- [13] Masanari Kondo et al. “The impact of feature reduction techniques on defect prediction models”. In: *Empirical Software Engineering* 24 (2019), pp. 1925–1963.
- [14] C Manjula and Lilly Florence. “Deep neural network based hybrid approach for software defect prediction using software metrics”. In: *Cluster Computing* 22.Suppl 4 (2019), pp. 9847–9863.
- [15] Santosh S Rathore and Sandeep Kumar. “A study on software fault prediction techniques”. In: *Artificial Intelligence Review* 51 (2019), pp. 255–327.
- [16] Zhou Xu et al. “Software defect prediction based on kernel PCA and weighted extreme learning machine”. In: *Information and Software Technology* 106 (2019), pp. 182–200.
- [17] Linchang Zhao et al. “Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks”. In: *Neurocomputing* 352 (2019), pp. 64–74.
- [18] Osama Al Qasem, Mohammed Akour, and Mamdouh Alenezi. “The influence of deep learning algorithms factors in software fault prediction”. In: *IEEE Access* 8 (2020), pp. 63945–63960.

BIBLIOGRAPHY

- [19] Jiechao Gao, Haoyu Wang, and Haiying Shen. “Task failure prediction in cloud data centers using deep learning”. In: *IEEE transactions on services computing* 15.3 (2020), pp. 1411–1422.
- [20] Aashish Gupta et al. “Novel xgboost tuned machine learning model for software bug prediction”. In: *2020 international conference on intelligent engineering and management (ICIEM)*. IEEE. 2020, pp. 376–380.
- [21] Ram Shankar Siva Kumar et al. “Adversarial machine learning-industry perspectives. In 2020 IEEE Security and Privacy Workshops (SPW)”. In: *IEEE 1* (2020), pp. 69–75.
- [22] Ning Li, Martin Shepperd, and Yuchen Guo. “A systematic review of unsupervised learning techniques for software defect prediction”. In: *Information and Software Technology* 122 (2020), p. 106287.
- [23] Guanjun Lin et al. “Software vulnerability detection using deep neural networks: a survey”. In: *Proceedings of the IEEE* 108.10 (2020), pp. 1825–1848.
- [24] C Nalini and T Murali Krishna. “An efficient software defect prediction model using neuro evolution algorithm based on genetic algorithm”. In: *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE. 2020, pp. 135–138.
- [25] Kittisak Wongpheng and Porawat Visutsak. “Software Defect Prediction using Convolutional Neural Network”. In: *2020 35th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*. 2020, pp. 240–243.
- [26] Zhen Li et al. “Sysevr: A framework for using deep learning to detect software vulnerabilities”. In: *IEEE Transactions on Dependable and Secure Computing* 19.4 (2021), pp. 2244–2258.
- [27] Cong Pan, Minyan Lu, and Biao Xu. “An empirical study on software defect prediction using codebert model”. In: *Applied Sciences* 11.11 (2021), p. 4793.
- [28] Kajal Tameswar, Geerish Suddul, and Kumar Dookhitram. “Enhancing deep learning capabilities with genetic algorithm for detecting software defects”. In: *Progress in Advanced Computing and Intelligent Engineering: Proceedings of ICACIE 2020*. Springer. 2021, pp. 211–220.

- [29] Kun Zhu et al. “Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network”. In: *Journal of Systems and Software* 180 (2021), p. 111026. ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2021.111026>. URL: <https://www.sciencedirect.com/science/article/pii/S0164121221001230>.
- [30] Joseph Bamidele Awotunde et al. “A Feature Selection-Based K-NN Model for Fast Software Defect Prediction”. In: *International Conference on Computational Science and Its Applications*. Springer. 2022, pp. 49–61.
- [31] Iqra Batool and Tamim Ahmed Khan. “Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review”. In: *Computers and Electrical Engineering* 100 (2022), p. 107886. ISSN: 0045-7906. DOI: <https://doi.org/10.1016/j.compeleceng.2022.107886>. URL: <https://www.sciencedirect.com/science/article/pii/S0045790622001744>.
- [32] Mohammad Sh Daoud et al. “Machine learning empowered software defect prediction system”. In: (2022).
- [33] Anh Ho, Nguyen Nhat Hai, and Bui Thi-Mai-Anh. “Combining deep learning and kernel PCA for software defect prediction”. In: *Proceedings of the 11th International Symposium on Information and Communication Technology*. 2022, pp. 360–367.
- [34] Ruchika Malhotra, Chitra Singla, and Daniyal Farooque. “Comparison of Hidden Markov Model with other Machine Learning Techniques in Software Defect Prediction”. In: *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*. 2022, pp. 1–5. DOI: [10.1109/I2CT54291.2022.9824549](https://doi.org/10.1109/I2CT54291.2022.9824549).
- [35] Pravali Manchala and Manjubala Bisi. “Diversity based imbalance learning approach for software fault prediction using machine learning models”. In: *Applied Soft Computing* 124 (2022), p. 109069.
- [36] Babajide J. Odejide et al. “An Empirical Study on Data Sampling Methods in Addressing Class Imbalance Problem in Software Defect Prediction”. In: *Software Engineering Perspectives in Systems*. Ed. by Radek Silhavy. Cham: Springer International Publishing, 2022, pp. 594–610. ISBN: 978-3-031-09070-7.

BIBLIOGRAPHY

- [37] Szymon Stradowski and Lech Madeyski. “Machine learning in software defect prediction: A business-driven systematic mapping study”. In: *Information and Software Technology* (2022), p. 107128.
- [38] Sagheer Abbas et al. “Data and Ensemble Machine Learning Fusion Based Intelligent Software Defect Prediction System.” In: *Computers, Materials & Continua* 75.3 (2023).
- [39] Femi Johnson et al. “Optimized ensemble machine learning model for software bugs prediction”. In: *Innovations in Systems and Software Engineering* 19.1 (2023), pp. 91–101.
- [40] Aimen Khalid et al. “Software Defect Prediction Analysis Using Machine Learning Techniques”. In: *Sustainability* 15.6 (2023), p. 5517.
- [41] Iqra Mehmood et al. “A Novel Approach to Improve Software Defect Prediction Accuracy Using Machine Learning”. In: *IEEE Access* (2023).
- [42] Sanchita Pandey and Kuldeep Kumar. “Software Fault Prediction for Imbalanced Data: A Survey on Recent Developments”. In: *Procedia Computer Science* 218 (2023). International Conference on Machine Learning and Data Engineering, pp. 1815–1824. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2023.01.159>. URL: <https://www.sciencedirect.com/science/article/pii/S187705092300159X>.
- [43] Muhammad Shafiq et al. “Scientific programming using optimized machine learning techniques for software fault prediction to improve software quality”. In: *IET Software* (2023).
- [44] Tarunim Sharma et al. “Ensemble Machine Learning Paradigms in Software Defect Prediction”. In: *Procedia Computer Science* 218 (2023). International Conference on Machine Learning and Data Engineering, pp. 199–209. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2023.01.002>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050923000029>.
- [45] Zuhaira Muhammad Zain, Sapiyah Sakri, and Nurul Halimatul Asmak Ismail. “Application of Deep Learning in Software Defect Prediction: Systematic Literature Review and Meta-analysis”. In: *Information and Software Technology* (2023), p. 107175.