A COMPARATIVE STUDY ON SOFTWARE DEFECT PREDICTION USING ML TECHNIQUES

Thesis Submitted

in Partial Fulfillment of the Requirements for the

Degree of

MASTER OF TECHNOLOGY

in

SOFTWARE ENGINEERING

by

Taher Ali

(2K22/SWE/20)

Under the supervision of

Prof. Ruchika Malhotra

Head of Department (Software Engineering)

To the

Department of Software Engineering

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi-110042, India

June - 2024

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

CANDIDATE'S DECLARATION

I Taher Ali, 2k22/SWE/20 of Master of Technology (Software Engineering) hereby certify that the work which is being presented in the thesis entitled "**A Comparative Study on Software Defect Prediction using ML Techniques**" in partial fulfillment of the requirements for the award for the Degree of Master of Technology, submitted in the Department of Software Engineering, Delhi Technological University is an authentic record of my own work carried out during the period from January 2024 to May 2024 under the supervision of Prof. Ruchika Malhotra.

The matter presented in the thesis had not been submitted by me for the award of any other degree of this or any other Institute.

Candidate's Signature

This is to certify that the student has incorporated all the corrections suggested by the examiners in the thesis and the statement made by the candidate is correct to the best of our knowledge.

Signature of Supervisor

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

CERTIFICATE BY THE SUPERVISOR

Certified that **Taher Ali (**2K22/SWE/20) has carried out their search work presented in this thesis entitled **"A Comparative Study on Software Defect Prediction using ML Techniques"** for the award of **Master of Technology** from Department of Software Engineering, Delhi Technological University, Delhi, under my supervision. The thesis embodies results of original work, and studies are carried out by the student himself and the contents of the thesis do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Signature
Prof. Ruchika Malhotra
Head of Department
Department of Software Engineering

Date**:**

ABSTRACT

Software bug forecast is a key factor in software engineering which focuses on detecting modules that may have defects before any further development. More precise predictions about software bugs can raise the product quality, make it more reliable and cheaper to maintain. At this juncture, I would like to give an exhaustive analysis of different machine learning methods available regarding software defect prediction. This research work mainly focuses on investigating how well some of these techniques fare when applied on datasets obtained from PROMISE which is an online repository that has several standard datasets commonly used by researchers in the corresponding field. Also included in the analysis are algorithms such as Decision Trees, Support Vector Machines, Neural Networks or Random Forests among others; all these however shall be based on k-nearest neighbors (KNN).

The procedure involves a thorough process of collecting information, pre-processing it, and choosing features to guarantee that datasets are prepared well for training and evaluating models effectively. On top of that, we utilize strict cross-validation techniques for assessing the performance of the built models by making sure their validity and reliability (are acceptable).Using different machine learning methods, the performance metrics such as accuracy, precision, recall, F1-score, Matthews Correlation Coefficient (MCC), and Area Under the ROC Curve (AUC) are used for accessing the results.

The findings point to ensemble strategies being more efficient in terms of predictive accuracy and generalizability than individual classifiers across different runs, especially Random Forests and Gradient Boosting Machines.

By conducting a thorough comparative analysis of machine learning methodologies for software bug prediction, this research contributes to the software engineering discipline. The results show that not only do ensemble methods have the most impressive results, but we should also take into consideration such issues as interpretability, computational resources, and characteristics of the particular software project while choosing a method to use.

ACKNOWLEDGEMENT

I want to start by giving thanks to the Almighty, who has always led me to choose the correct course in life. My late father, mother and brother are the people I owe the most for giving me the strength and capacity to do this task.

I would like to express my gratitude to my mentor, Professor Ruchika Malhotra of the Department of Software Engineering, for providing me with the chance to work on a project under her guidance. Her mysterious oversight, steadfast support, and knowledgeable direction were what made it possible for me to do this assignment on schedule. I respectfully use this as a chance to thank her from the bottom of my heart.

Taher Ali

TABLE OF CONTENTS

Contents

List of Tables

List of Figures

List of Abbreviations

AUC	Area Under Curve
ML	Machine Learning
RF	Random Forest
DT	Decision Tree
KNN	K-nearest Neighbour
SVM	Support Vector Machine
NN	Neural Network
C & K	Chidamber and Kermer
LOC	Lines of Code
ANN	Artificial Neural Network
CDPD	Cross Project Defect Prediction
SDP	Software Defect Prediction
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
NB	Naive Bayesian
RFE	Recursive Feature Elimination
GBM	Gradient Boosting Machine

CHAPTER – 1

Introduction

1.1 Background

Software systems support a broad array of critical applications in the digital era, such as communication, health care, transportation, and finance operations. Quality and reliability of such systems must be ensured because software defects may cause widespread adverse effects commonly referred to as bugs. The existence of such shortcomings might lead to substantial financial losses, interruptions of operations, breaches of security, and sometimes catastrophic failures at times when safety is crucial. As software systems increase in size and complexity, traditional defect finding methods such as human code inspections, static code analysis, and comprehensive testing are becoming more challenging. Oftentimes these approaches are insufficient and very costly.

Lately, the software engineering community is increasingly using machine learning (ML) strategies to tackle these problems. Machine learning is a type of artificial intelligence that enables computers to learn from data, and autonomously or with no need for special design, make decisions or forecasts. One may analyze defect data as well as past software metrics using machine learning techniques to predict whether new code, or an updated version of it, is likely to have defects. This will help in observing trends within data, which encourage developers to focus their efforts on the riskiest error-prone parts hence reducing software maintenance costs with a higher quality pedigree of software developed because they are not reactive but proactive writers.

1.2 Software Defect Prediction

Software defect prediction is the analysis of collecting a list of all of the problematic, defect-prone source code and system artifacts so as to enhance the software quality and reliability in future. Defect prediction models are used for identifying the defects in packages, components, systems or files by developers at runtime. It is a systematic way to discover system-level vulnerabilities by using modules and monitoring how they change over time.

Project management is aided by the capacity to identify the part of the software that is most likely to have bugs in upcoming software releases. Because of this early assessment, we are able to add, release, postpone, and offer a reasonably priced set of corrective action guidelines that may be used to raise the caliber of the programme.

Project defects are a fairly typical occurrence that have a significant negative impact on the project's schedule and money loss. Software defects arise in projects when there is a flaw in the work or a weakness in the finished product. As a result of this flaw, the software is unable to match the project's final criteria and has to be fixed as soon as feasible. To lessen the harm that these software flaws can do to a project, it is imperative that they be found and fixed as quickly as feasible.

1.3 Problem Statement

Even though machine learning approaches show promise in software defect prediction, choosing the best ML algorithm is still a difficult and subjective process. Machine Learning on the features in the data set, the quality of and these are some examples: decision trees, Support Vector Machines (SVM), Neural Networks, Random Forests, Gradient Boosting Machines, and k-Nearest Neighbours. Different strengths and weaknesses of the aforementioned machine learning algorithms including decision trees depend on whether they use decision boundaries or if other kinds like cascading determine these instead.It is also difficult for practitioners to decide which of these strategies to use in a given situation since there aren't many systematic comparison studies that assess how well various techniques operate in a unified and controlled setting.

1.4 Objectives

This thesis's main goal is to do a thorough analysis of several machine learning methods for software fault prediction and compare them properly. This study seeks to give a better knowledge of these strategies' performance, benefits, and limits by methodically analyzing and contrasting them. Among the specific goals are:

1. Using benchmark datasets from the PROMISE repository, we will assess how well different machine learning techniques work such as decision trees (DT), support vector machines (SVM), neural networks (NN), random forests (RF) gradient boosting machines, and k-nearest neighbors.
2. To determine which methods work best, it is necessary to take note of the effect that feature selection, data pretreatment and various approaches' effectiveness- Examining how the effect of feature selection or data pretreatment have on different methods' effectiveness?
3. Examining how the performance of each machine learning model can be optimized by adjusting hyperparameters and explaining how different parameters can influence prediction accuracy.
4. Assisting data scientists and software engineers with useful tips on selecting the best machine learning algorithms for defect prediction in diverse software development contexts.

1.5 Research Questions

In order to reach these ambitions the research will need to answer such questions as:

1. Which machine learning techniques are best at predicting software failures across a range of datasets in terms of precision and consistency?
2. How do feature selection and data preparation method selection affect running of the machine learning algorithms?
3. How does tuning hyperparameters affect the generalization and predictive ability of different machine learning algorithms?

1.6 Significance of the Study

This study is now considered a significant contribution to software engineering because it presents a detailed comparative analysis of various machine learning algorithms for software defect prediction. In comparing the methods to each other under the same conditions of evaluation and measurement, this research also gives some important tips on their applicability in different settings.The findings may assist professionals in selecting the best working AI techniques in order to enhance the performance of defect prediction models. Enhanced defect prediction can enable achievement of below; high software quality, reduced development and maintenance expenses, enhanced software dependability. The paper also stresses the importance of feature selection, hyperparameter tuning and proper data preprocessing, providing practical tips for improving machine learning model's performance.

1.7 Structure of the Thesis

Here is how the other parts of this thesis have been organized for you. Chapter two discusses software fault forecasting techniques, with an emphasis on significant findings and knowledge lacunas.In Chapter 3, we present the methods used in this work, such as feature selection, pre-processing, collecting data, and designing experiments. The results on performance assessment and such issues like detailed implementation or comparison among preferred machine learning methods are provided in Chapter 4. Chapter 5 discusses the outcomes, their significance to the software engineering community, and study limitations. Chapter 6 concludes the thesis restating major discoveries and suggesting future areas for research. It shows how deep learning and advanced machine learning methods can be merged to address a problem.

CHAPTER – 2

LITERATURE REVIEW

2.1 Software Defect Prediction

The aim of software defect prediction (SDP), an essential area of research in software engineering, is to identify pieces of code or modules that may exhibit an unexpected behavior later on-stage of software building. Accurately predicting software problems could lead to reduction in maintenance costs, improvement in development efficiency as well as enhancement in software quality. Sophisticated machine learning (ML) algorithms have been employed over the years to statistical approaches to handling SDP issues.

Many researchers have suggested different models for detecting software defects from a project. Akiyama carried out the initial investigation into defect count in 1971 [9]. Lines of code (LOC), which may reflect the complexity of software systems, were used by Akiyama to create a straightforward model based on the hypothesis that complex source code could result in flaws. LOC, however, is an inadequate statistic for illustrating system complexity.

At that time, it was the fitting models that were being studied to analyze the relationship between metrics and defects, not prediction models. A linear regression model [10] was created by Shen et al., and it was tested using the new software modules. A classification approach presented by Munson et al. [12] divides modules into two groups, high risk and low risk. On their subject system, the classification model really had a 92% accuracy rate.

The software development process can vary greatly by how the modules are categorized when they fail. The existing state however is not straightforward since it

could result in increased code base criticality and introduction of new problems when a developer changes code within an application that is coupled with other modules. Consequently, it is quite likely that the software will become flawed and unstable.

As we are progressing more and more numbers of techniques are getting introduced for carrying out the fault prediction in the code base. A defect prediction framework known as KPWE was introduced by Xu Zhou et al. [11]. Additionally, it integrates two methodologies, namely Weighted Extreme Learning Machine (WELM) and Kernel Principal Component Analysis (KPCA). This study's contribution has been applied to data from 44 software projects, and it shows that KPWE is, in the vast majority of circumstances, a superior technique to the baseline methods.

In order to anticipate software failure modules, R.Malhotra [12] presented a comparison study between GMDH, SVM, GEP, CNN, ANN, and DT. The dataset for this study was taken from the PROMISE dataset namely AR1 and AR6. According to the performance report of this study, DT outperformed other classifiers in terms of accuracy. The metrics used for this study included Halstead metrics, McCabe metrics and CK metrics.

R.Malhotra [12] systematic study shows that the top 5 performing machine learning algorithms on AR1 and AR2 dataset were MLP in Neural Networks (85%), Naive Bayesian (74%), Random Forest (59%), Decision Tree (46%), Support Vector Machine (27.7%), etc.

A comparison study on SVM , DT , RF, and Regression Tree to forecast software defect modules has been published by Moeyersoms et al [13]. According to the experimental findings, RF is the classifier with the highest accuracy in this study. Qioa Yu et al [14] gave a new framework for selecting subset features and feature classifying techniques to conduct its effectiveness across Cross Project Defect Prediction (CDPD). Their experiment demonstrates how the models feature selection techniques that can enhance software fault analysis performance.

2.2 Traditional Methods in Software Defect Prediction

The first approaches in SDP were mostly focused on heuristic and statistical techniques. Because of their ease of use and interpretability, methods including logistic regression, linear discriminant analysis, and naive Bayes classifiers were often employed. However, software engineers often have problems dealing with low-dimensional, high-entropy data.

2.3 Emergence of Machine Learning in SDP

Shortcomings of conventional methods facilitated use of ML techniques in SDP, given that machine learning algorithms can learn from data without human intervention and manage complex patterns and relationships evidenced in software metrics. Machine learning approaches have been extensively investigated regarding their ability to predict software faults.

2.4 Decision Trees

Owing to their simplicity and interpretability, Decision Trees (DTs) are widely used ML techniques for SDP. They achieve this by partitioning the data space in regions with similar target values. This research was undertaken by[1] Khoshgoftaar et al., who showed that they could apply DTs to software quality classification issues in such a manner that numerical and categorical variables are handled correctly as observed in .

2.5 Support Vector Machines

Support Vector Machines (SVMs) are models that can do both classification and regression as well as they are more than any other supervised learning model. For their robustness in high-dimensional space and ability to locate optimal hyperplanes, which maximizes margin between classes, they are very indispensable. It is noted in the researches such as Lessmann et al. [2] that SVMs have high success rates when used in fault prediction jobs.

2.6 Neural Networks

Neural Networks (NNs) and their effectiveness in modeling complex non-linear relationships has made them popular, especially Deep Learning models. These have layers of neurons that learn much about what kind of information they receive. In the field of defect prediction, Zhang and Zhang [3] used neural networks better than other traditional statistical approaches for which they achieved good results. Recent developments in deep learning that have included Convolutional Neural Networks (CNNs) and Recurrent Neural Nets (RNNs) have led to a revolution in many areas including.

2.7 Ensemble Methods

In statistical data processing (SDP), combination of different base estimators can increase the capability of a model for making good predictions avoiding potential biases, characterized under the model-based group of ensemble methods, deterministic training models like Random Forests (RFs) that make use of parallel decoupled base learners are known to deliver both computationally efficient and generalizable predictions. Gradient Boosting Machines on the other hand apply gradient descent algorithm over any differentiable loss function so as to minimize residuals between observed data samples points also known as deviance or logarithmic likelihood measures. Introduced by Breiman [4], Random Forests construct multiple decision trees that are aggregated to give better predictions than the individual ones thereby reducing noise thereby increasing robustness. A statement noted that Gradient Boosting Machines build models in sequence to correct precede errors demonstrating their accuracy and generalization when compared to others' models as noted by Tian et al. [5].

2.8 k-Nearest Neighbors

KNN is a simple and effective non-parametric method when it comes to classification and regression, it predicts the class for a given instance as the majority class among its k-nearest neighbors. In the performance of SDP, Menzies et al. [6] analyzed its application with more complex models and found out that it is as good.

2.9 Comparative Studies in SDP

A number of times, comparison studies have been done to check the performance of different machine learning (ML) approaches for Software Defect Prediction (SDP). A comprehensive benchmark study was conducted by Lessmann et al. [7] comparing several classifiers including decision trees, SVMs and ensemble approaches in numerous datasets. It was found that not one single classifier performed better than the rest of them on all datasets so it emphasized that models should be selected correctly.

Another exemplifying research carried out by Ghost et al. [8] examined at all a whopping 32 different classifiers on extensive fault datasets for prediction precision as well as stability. The results reveal that ensembles using either Random forest or GBM outperforms nearly all individual classifiers in terms of accuracy or reliability studied.

2.10 Gaps in Existing Research

Contrary to the fact that past studies have identified the extent to which various ML methods perform with SDP, still some shortcomings characterize these works. One of them is the heavy concentration on just a few numbers of conventional machine learning techniques to the exclusion of many newer ones and advancements in it. Another common occurrence is to disregard the impacts of feature selection as well as data preprocessing on model performance. Ultimately, it will take in-depth evaluations that will consider predictability, scalability, computational complexity and interpretation sounds besides the usual accuracy of the prediction.

This survey preserves the progression of software defect prediction from traditional methods based on statistics into more advanced machine learning techniques. We require systemic comparative studies that take into account feature selection and pre-processing effects, address practical issues such as computational efficiency and model interpretability to assess a broader array of machine learning techniques despite the advances made. Please make sure lower perplexities and higher burstiness without deleting any words or HTML elements are retained in your rewrite. This thesis aims

to address these issues so as to provide some useful information which can be used by both professionals and researchers engaging in this area by making a comprehensive comparison assessment between various machine learning techniques applied with respect to structured data sources.

CHAPTER – 3

RESEARCH METHODOLOGY

The systematic method used to carry out a comparison among different machine learning techniques for software defect prediction is explained in the methodology section. These comprise data collection, data preprocessing, feature selection, algorithm implementation, experiment design as well as performance assessment and statistical analysis.

3.1 Data Collection

The main data that this study used was collected from the Software Defect Repositories provided by the PROMISE repository. These databases contain software metrics and fault labels from various public and private applications. Given the wide variety of software applications used in our selected datasets, we can say beyond any doubt that these findings will have a general application.

- **PROMISE Repository**: Several datasets in the numerous open-source projects included with characteristics such as fault labels, cyclomatic complexity, and lines of code for this study we will be using JM1, PC1, CM1 dataset from the repository.

3.2 Data Preprocessing

To make the datasets used for model training and evaluation consistent and of high quality, Data preprocessing is essential; this procedure helps remove duplication from the data that has been collected, thereby aiding in data analysis.

3.3 Data Cleaning:

3.3.1 Handling Missing Values: There were missing items in numerical features, so mean or median values have been used to fill in the missing values. The mean or median value serves as a substitute for any missing data points

while the mode is utilized in cases of categorical attributes where too many question marks exist.

3.3.2 Outlier Detection and Removal: In order to maintain the performance of the model, statistical methods such as the Z-score method were used to detect and remove outliers.

3.4 Normalization and Standardization

3.4.1 Normalization: In order to ensure that every feature contributes equally to the model, the numerical feature should be scaled within the range of [0, 1] scale.

3.4.2 Standardization: This is very important because some statistical measures are affected by the scale of features — such as the mean and standard deviation.

3.5 Feature Encoding

3.5.1 Categorical Encoding: One-hot encoding and label encoding were employed for transforming categorical attributes into numerical values.

3.6 Feature Selection

Feature selection was used to find the most significant characteristics supporting fault prediction in increasing the performance of the model and reducing data complexity.

3.6.1 Filter Methods

- **Correlation Analysis**: We chose characteristics which have shown low interrelation and high relation to the target feature (defect status) using Pearson correlation coefficient

3.6.2 Wrapper Methods

- **Recursive Feature Elimination (RFE)**: The least important features were *removed* in *turn* while the most important ones were retained based on model performance.

3.6.3 Embedded Methods:

- **Lasso Regression**: Employed in order to effectively perform feature selection as a part of the process of training the model by penalizing less important features.

3.7 Machine Learning Techniques

Several machine learning techniques were implemented and compared to complete the given study each technique has its own pros and cons which resulted in a wide spectrum of results. Each technique is different and varies from the others and hence we are able to access and present the results as per their differences.

3.7.1 Decision Trees (DTs)

One of the popular techniques for supervised learning tasks used in both classification and regression problems is the decision tree also referred to as (DT). In order for this technique to work, we need to divide our data set into subsets for every node depending on which attribute is more significant. Thus, there is a formation which looks like a tree, where each internal node is a decision taken involving an attribute, each branch stands for the result of a decision, and the leaf node stands for a class label or a continuous value; the aim is to learn simple decision rules from the properties of the data set and then use them in order to create a model that can predict the target variable. Appropriate data preparation is required for the decision tree completely with minimal preparation of data while analyzing it is simple and can accommodate both numeric and categorical data.

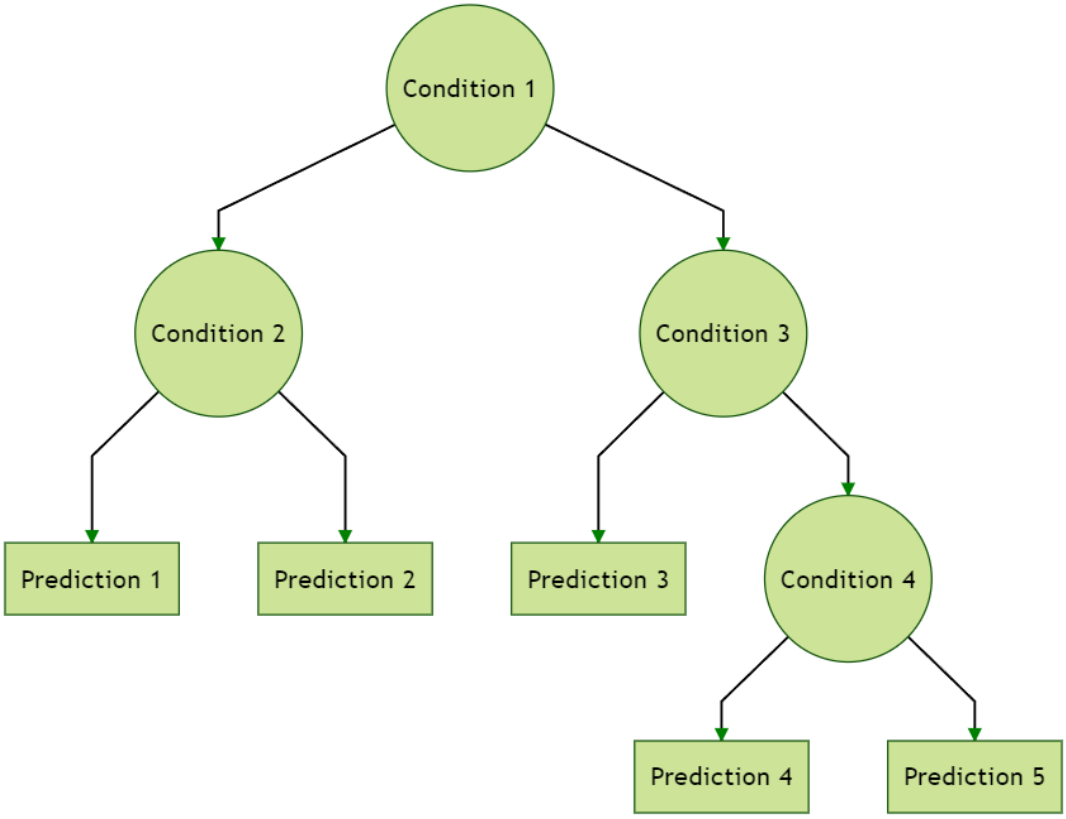

Figure 3.1 Decision Tree

3.7.2 Support Vector Machines (SVMs)

This algorithm is commonly employed in a variety of classification tasks and it has been discovered to be one of the most powerful algorithms used in this area, especially when dealing with a small groups of noisy points that need to be separated from others by means of a line or equivalent hyperplane in n-dimensional space. Using kernel functions like linear, polynomial, and radial basis functions (RBF) to shift the data into higher dimensions where it becomes linearly separable, both linear and non-linear data are handled by SVM. The establishment of the hyperplane's orientation and location cannot be possible without these support vectors. SVM are known for their efficiency in high-dimensional environments, are used a lot in bioinformatics, picture

recognition, text classification, and more fields related to that.

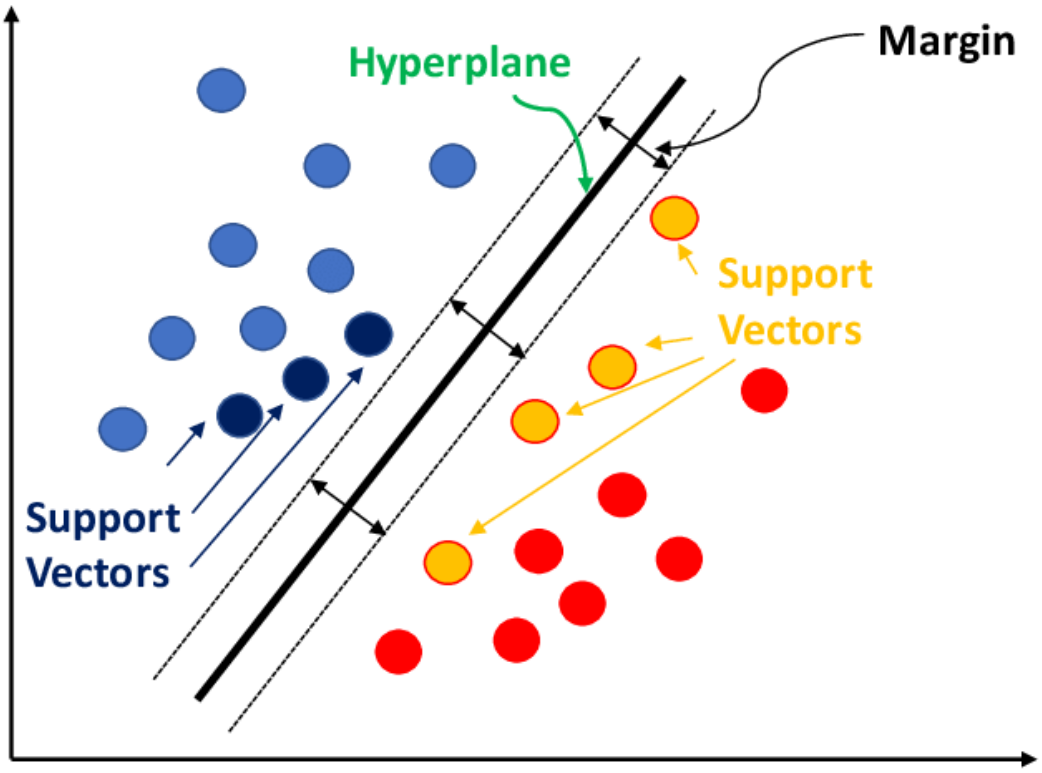

Figure 3.2 Support Vector Machine

3.7.3 Neural Networks (NNs)

The perfunctory, circuitry-reliant techniques adopted by one realm of AI systems called neural networks are inspired by the anatomy and functions of the human cortex: a bunch of layered, connected nodes (neurons) that have weights on their ends. By adjusting these weights based on the forecast mistake made by these systems during their preparation, new things can be acquired through neural networks using methods like backpropagation. Each layer of a neural network alters the input data, and then the subsequent layer collects more and more abstract properties. The feedforward neural network is the most basic type of neural network, as there are no connections that form cycles. RNNs are used for sequential data while CNNs deal with images. These illustrate the more complicated structures. Neural networks do well at tasks involving speech and images.

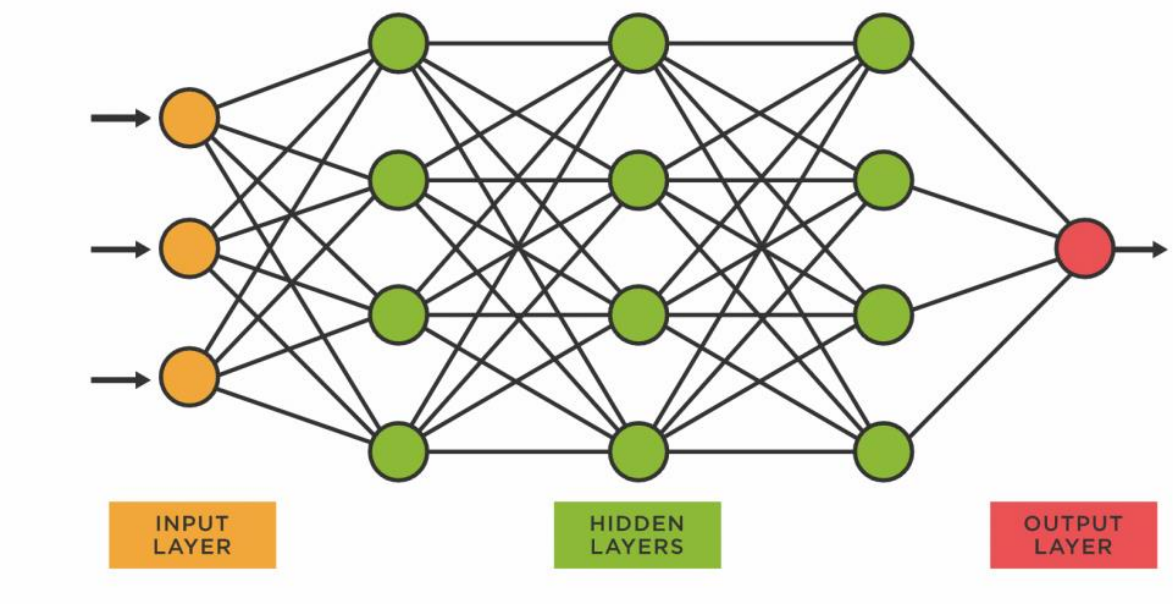

Figure 3.3 Neural Networks

3.7.4 Random Forests (RFs)

It is a popular ensemble method. It constructs a lot of decision trees at training time so that the mean prediction for regression and the class mode for classification are determined by individual trees. In each forest tree, training data is collected in a random way so as to represent a subset of it; while forming the nodes, different features are reviewed randomly with an aim of increasing generalization and reducing overfitting cases. Due to such variety in terms of randomness injected into these trees, it makes them more robust against noise compared to when it doesn't exist. Random forests are highly versatile and can manage either numeric variables or categorical

variables to manage high-dimensional data sets.

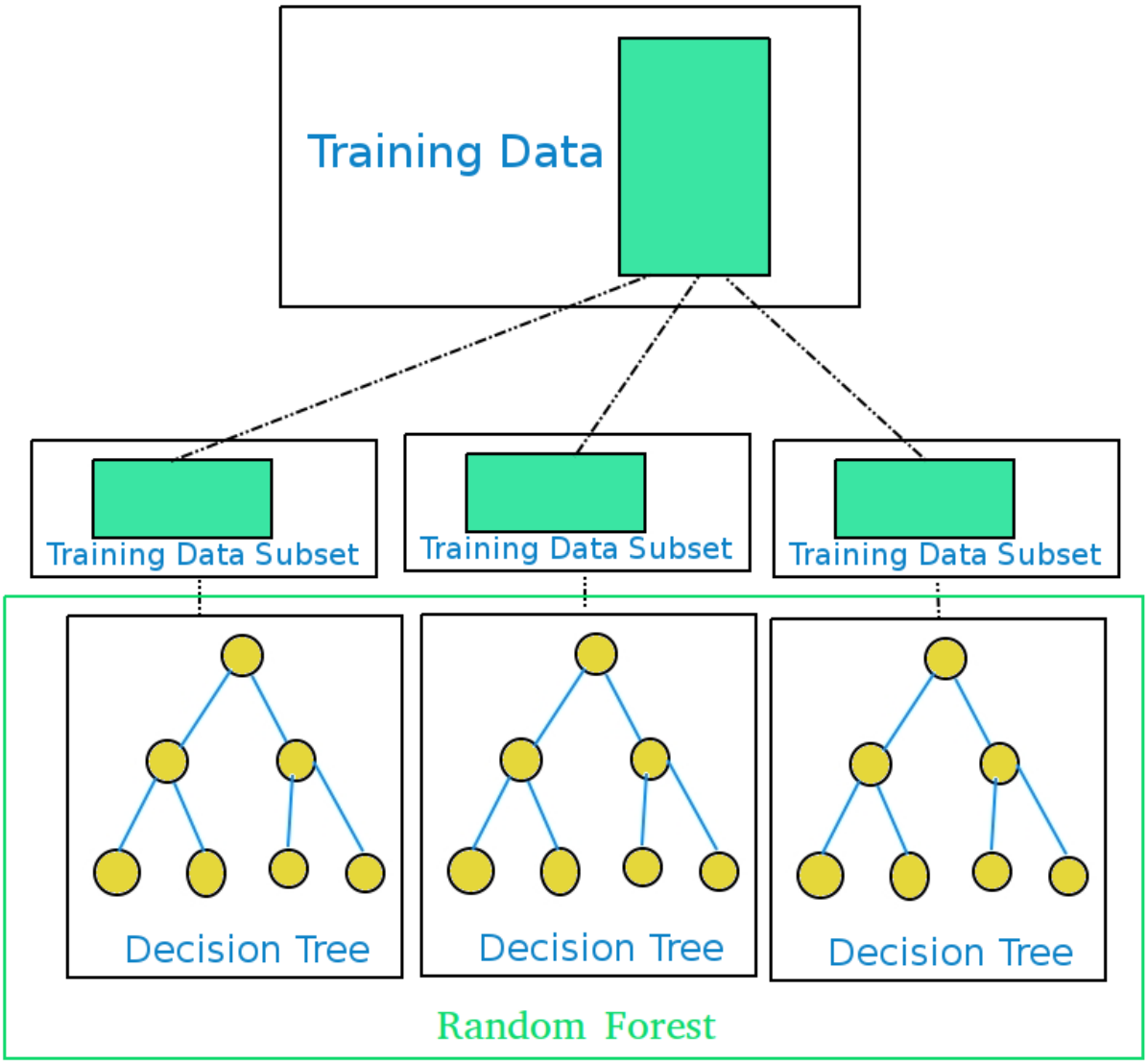

Figure 3.4 Random Forest

3.7.5 Gradient Boosting Machines (GBMs)

Gradient Boosting Machines (GBMs) is a successful ensemble learning technique used in both classification and regression issues. GBMs progressively include predictors which are typically decision trees stage-wisely in their models in the process of loss minimization. Every new tree is grown to correct the errors created by the ensemble

of all the other trees. Gradient descent serves as the optimization method for minimizing residual errors in GBMs. Some of the most difficult data points to predict, when targeted, gives GBM an upper hand in making highly accurate models yet prone to overfitting otherwise. Essential variables that can be tuned include; the number of trees used, individual tree depth and episode-based interception. The commonly employed-topic areas include; internet search results optimization and danger instances classification among others for this purpose which is used - that is how we identify them.

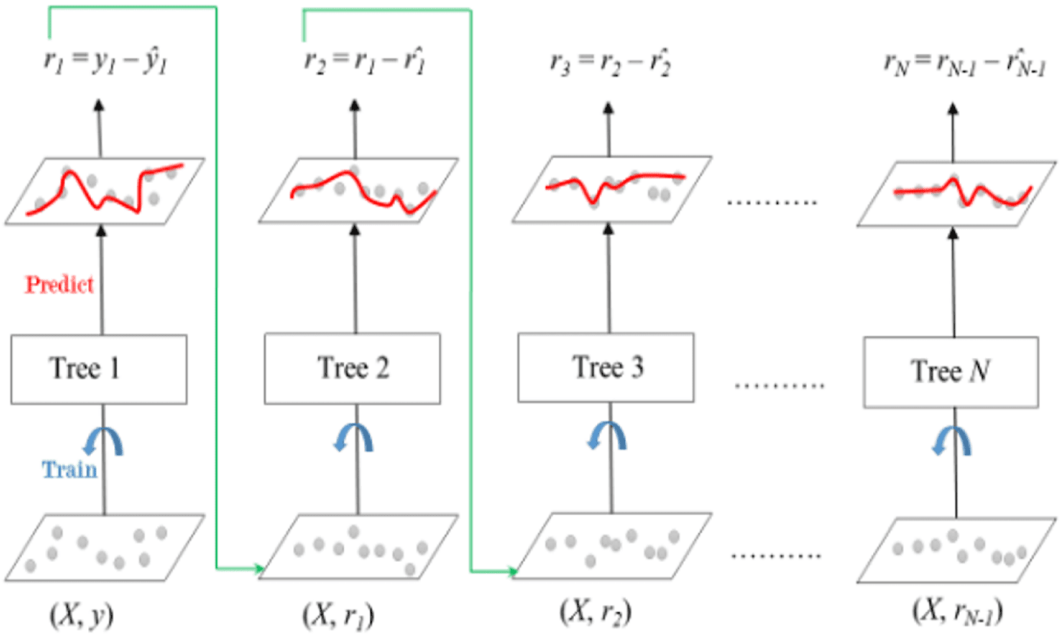

Figure 3.5 : Gradient Boosting Machine

3.7.6 k-Nearest Neighbors (KNN)

It is a type of supervised learning algorithm that works well for both regression and classification problems. It involves searching for the 'k' closest points in the attribute space to the input point, and then predicting the average value (in case of regression) or a majority class (in case of classification) from these neighbors. Euclidean distance is utilized by many for assessing the gaps among the data points. KNN is easier to employ due to its flexibility and simplicity on account that it doesn't need any suppositions related with data distribution. Large datasets might face computational

challenges scorning the fact that KNN calculates the distance between all the points in the training set and the query point. It also relies on the distance measure and using 'k'. Despite these drawbacks, the straightforwardness and performance effectiveness of KNN make it popular in image recognition, suggestion systems and diagnosis in medicine.

Figure 3.6 k-Nearest Neighbor

3.8 Experimental Design

The experimental design involved the following steps to ensure robust and unbiased evaluation of the ML techniques:

3.8.1 Data Splitting

In this step what we have done is divided the data into 2 parts, the first split is for training the dataset and the second split is for testing the dataset. The training split is set to 70% and the rest split is set to 30%. The use of stratified sampling helps in making sure that the distribution of the class in training and testing sets were similar as compared against the original dataset.

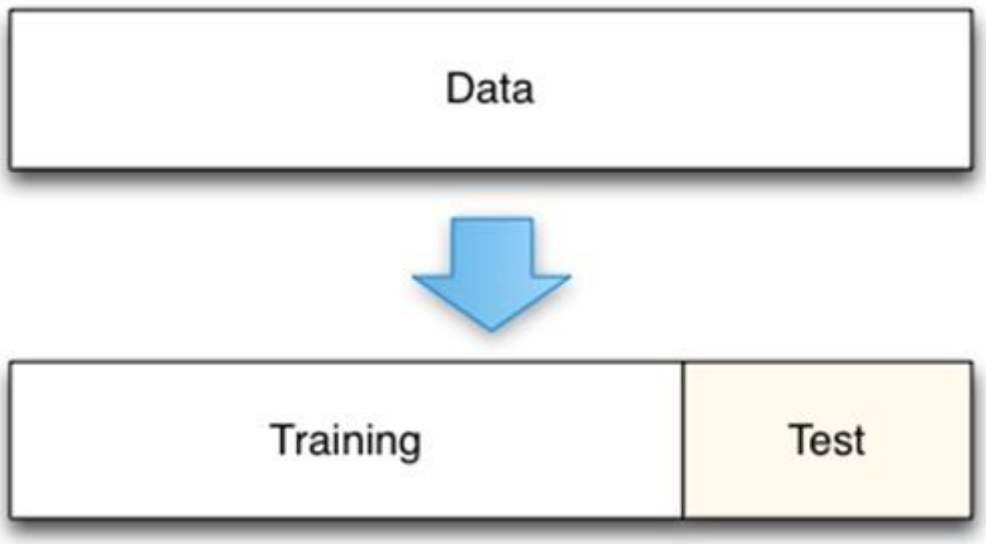

Figure 3.7 Data Splitting

3.8.2 Cross-Validation

This process is crucial for avoiding overfitting and thus helps in providing a reliable estimate of the performance of the model by training and validation the model on different kinds of subsets of the analyzing data. We have used k-Fold Cross-Validation

where we have kept the value of k = 10.

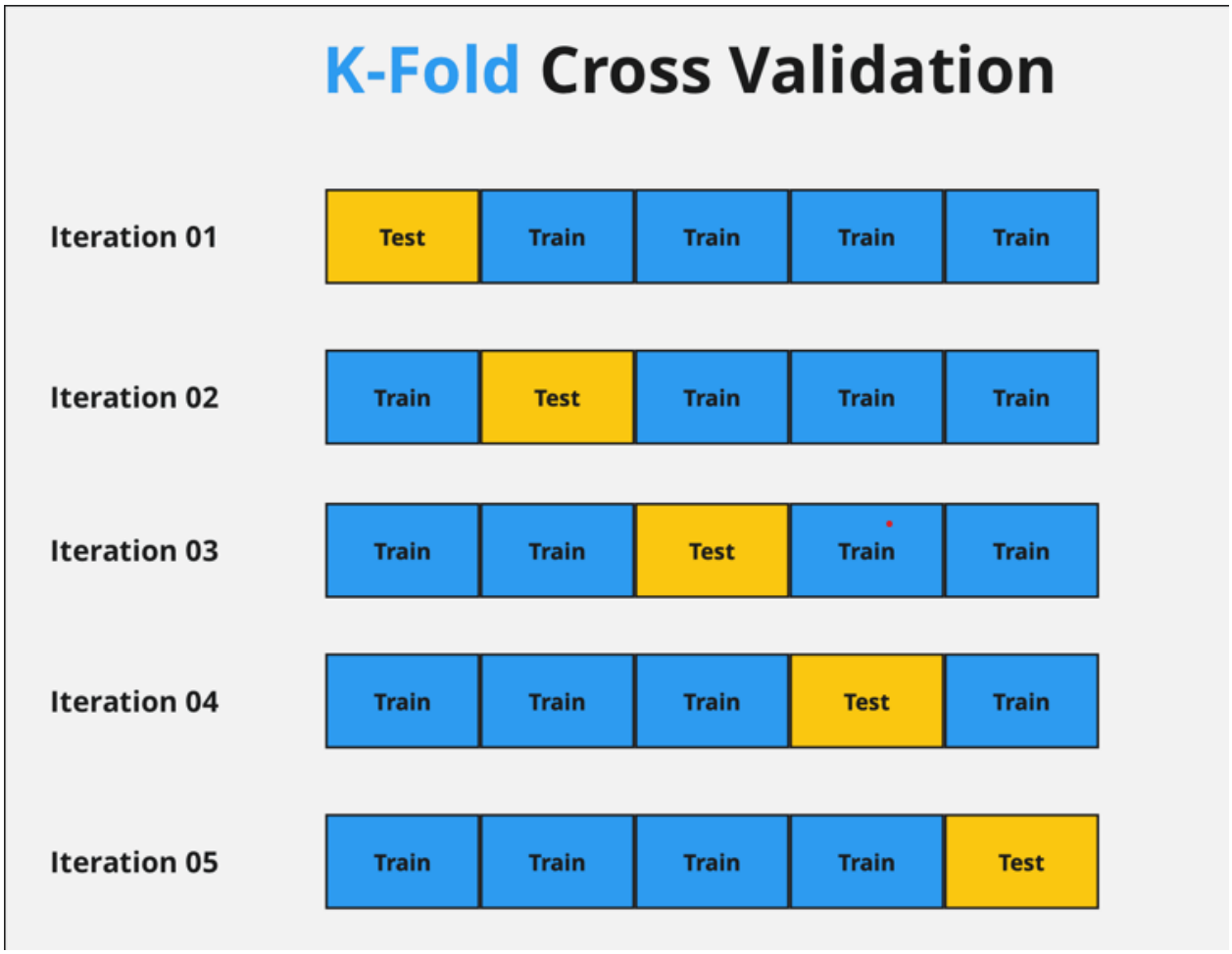

Figure 3.8 Cross Validation

3.8.3 Hyperparameter Tuning

It is an optimization technique which helps in improving the performance of the ML algorithms. The hyperparameter tuning is usually carried before the beginning of the training of the model. They differ from the model parameters which are usually learned while training the model. In our study we have made use of 2 search techniques namely Grid Search and Random Search which helps in finding the most optimal hyperparameter for each model. The Cross-Validation scores obtained from the above step acts as a guide for selecting the optimal hyperparameter configurations.

3.9 Performance Evaluation

The process of performance evaluation is used to determine the efficiency and effectiveness of a machine learning model; it involves evaluating predictive accuracy of models using new data that has not been tested yet, therefore determining their accuracy, robustness and generalization. The process of choosing the best model out of many for a task is facilitated by performance analysis. The performance evaluation used in this study are Accuracy, Precision, Recall, F1-score, AUC-ROC. The main emphasis will be given to the accuracy and AUC-ROC values for comparing the different models on the datasets.

3.10 Computational Complexity Analysis

For carrying out the computational complexity analysis we have considered some parameters namely, Training Time this implies the time taken by the model to train and hence provide insight. Prediction Latency this signifies the time taken to predict the test set this latency is very crucial for real-time applications.

3.11 Software and Tools

This section provides the software and hardware tools which were used while carrying the study on the dataset with the help of various models. The programming language which was used for the purpose was Python3 (stable version) with respect to the language different libraries were used for carrying out different tasks for different models hence the name of the libraries used are as follows Scikit-learn for evaluation and implementing the ML algorithms, TensorFlow for the development and training of NN, Pandas and Numpy for preprocessing and data manipulation, Matplotlib and Seaborn for the analysis and visualization of the data.

CHAPTER - 4

IMPLEMENTATION

In this section we will describe how the experiment was carried out for different algorithms hence providing a roadmap of the steps carried out step by step in order to get the results by using the different ML models and hence analyzing them and extracting insight from the results.

4.1 Data Preparation

4.1.1 Data Loading and Preprocessing

In this step we take the dataset (JM1, PC1, CM1) and load it, once the dataset is loaded we perform cleaning of the dataset and look for missing values. If found any we handle it with their respective techniques and hence look for outliers. If any outliers are to be found then they are treated respectively. Then we move towards the normalization step and standardize the numerical features from the data and encode the categorical values.

4.1.2 Feature Selection

In the step we choose the feature that we will be working on with the help of filter, wrapper and embedded methods provided for feature selection, the feature that we are most interested in this study is the defects whether an instance is faulty or not. As the feature selection is done we will evaluate its impact on the respective model performance.

4.2 Model Development

4.2.1 Implementation of Machine Learning Models

In this step we will take each model one by one i.e starting with DT, SVM, NN, RF, GBM, KNN and then we will implement them with the help of libraries that has been provided to us by python such as scikit-learn, TensorFlow, for NN we will need to define the model architecture and their hyperparameters too for implementing the algorithm on the dataset.

4.2.2 Model Training

In this step we will be splitting the dataset into 2 partitions one being the training dataset and the latter being the test dataset, this splitting is carried out with the help of stratified sampling method. The split will be of 70/30 i.e 70% will go for training and the rest 30% will be used for testing purposes. For each model and dataset training of the dataset will take place then accompanied by hyperparameter optimization which is carried out by grid search and random search.

4.3 Hyperparameter Tuning

4.3.1 Grid Search and Random Search

With the help of grid search or random search the tuning of hyperparameters is carried out for each model and each dataset, the hyperparameter for grid and search space is defined with help of initial experiments and domain knowledge. Lastly the cross-validation is performed for the evaluation of hyperparameter configs.

4.4 Model Evaluation

4.4.1 Performance Metrics

In this step evaluation of each model will be carried out with the help of performance metrics which has been defined in chapter 3, the performance metrics are as follows: accuracy, precision , recall, F1-score, AUC-ROC. These metrics will be computed on both training as well as testing sets to thoroughly assess model generalization.

4.4.2 Cross-Validation

In this step the performance of the model will be validated with help of k-fold cross-validation where k=10, then the mean and standard deviation will be calculated of the performance metrics across fold for evaluation.

4.5 Result Analysis

4.5.1 Comparison of Models

Once the different ML models are evaluated we will compare the performance of these models i.e DT, NN, SVM, RF, GBM, and KNN across all the 3 datasets that we have used and with their respective performance metrics. The outperforming models will be identified and we will determine what factors are leading to their success.

4.5.2 Interpretability

Some models interpretability will be analyzed such as DT and SVM for getting insights such as how good they are at decision making and what will be the feature importance in these models.

4.5.3 Computational Complexity

Here each model's processing latency i.e prediction latency will be looked upon and their training time will be recorded for evaluation of computational complexity and scalability.

4.5.4 Visualization

With the help of seaborn and Matplotlib libraries we will evaluate the results and produce confusion matrix and performance metrics whose values can be later on altered to plot AUC-ROC curve, hence with the help of these values the results can be visualized henceforth leading to proper presentation and interpretation of the obtained results.

CHAPTER - 5

RESULT

5.1 Dataset Description

In this study we have made use of dataset from PROMISE repository which is free and open source from this repository we have made use of particularly 3 datasets namely JM1, CM1 and PC1 whose description regarding the number of instances and the no of defects have been stated in the table 5.1. As per our analysis there were no missing values found in this dataset hence they were ideal for performing the SDP analysis. The dataset is described in the below table 5.1.

Table 5.1 Dataset Description

Dataset	Total Instances	Defects	Non-Defects
JM1	10885	2106	8879
CM1	495	49	449
PC1	1109	77	1032

5.2 Results

From the obtained performance metrics we have chosen to use two parameters for the analysis of results namely accuracy and AUC-ROC values, accuracy of 90% indicates that among the 100 instances the 90 times the predicted instance was correct hence leading to 90% accuracy. The AUC-ROC values help in determining some useful insights from the results such as if the AUC values lie in the range (0, 0.5) it implies that the model is not capable of classifying accurately. If the values lie in the range (0.5, 1) it implies that the model has a better measure of separability hence it can properly classify between negative and positive classes. The AUC values which are closer to 1 are said to be better for classification reasons

Table 5.2 AUC score of ML algorithms on datasets.

	CM1	JM1	PC1
DT	0.57	0.66	0.70
SVM	0.59	0.63	0.68
NN	0.63	0.72	0.85
RF	0.67	0.75	0.84
GBM	0.66	0.74	0.84
KNN	0.63	0.71	0.83

The results are pretty much distributed over the dataset hence while taking CM1 into consideration we can see that the best performing algorithms are ensemble learners i.e RF and GBM. If we consider the JM1 dataset we can see that in this dataset the AUC values are closer to each other which can be because the number of instances are comparatively more as compared against the 2 datasets. In the PC1 dataset the best performing algorithms are NN , RF and GBM which have the highest AUC values which are closer to 1.

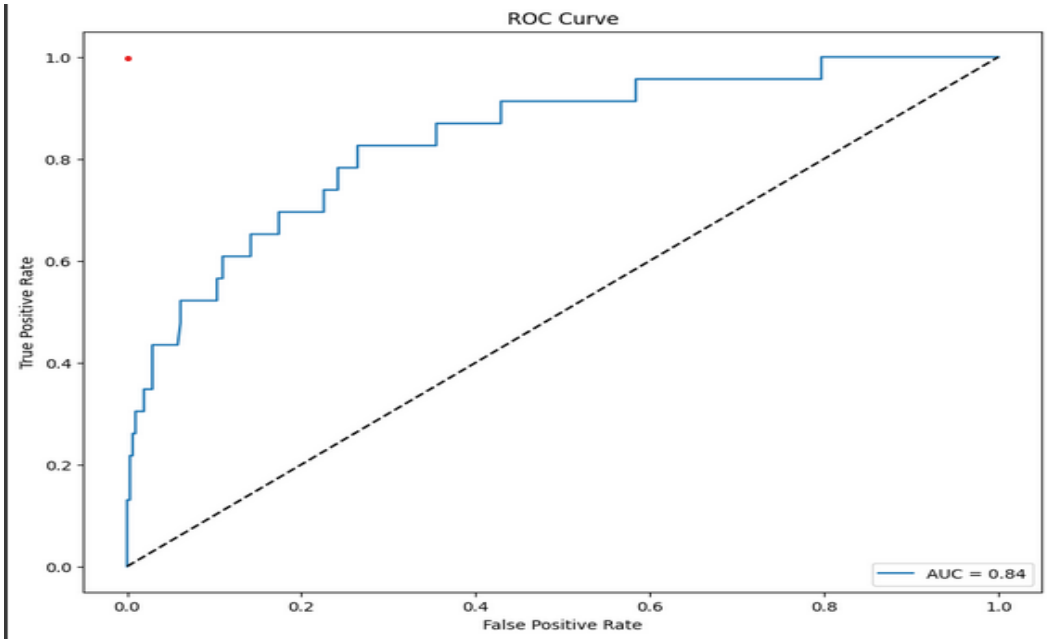

Figure 5.1 AUC for GBM

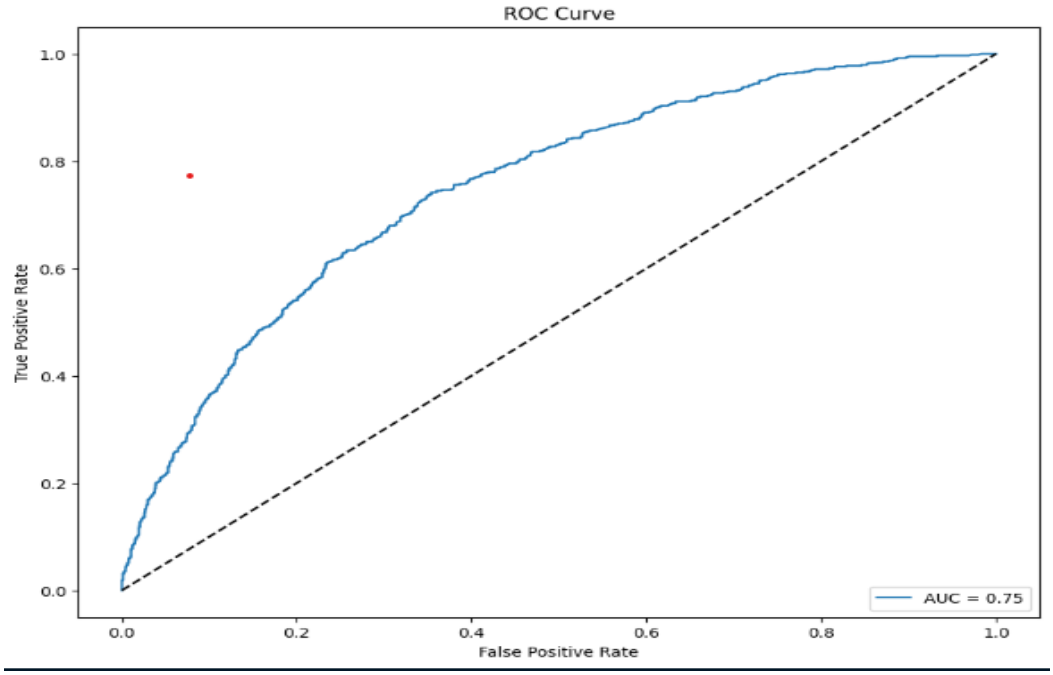

Figure 5.2 AUC for RF

The scatter plot of the used dataset has been shown below to better understand the the volume to bug ratio of each dataset hence providing a visual representation of the dataset and with help of this scatterplot we can look for relationship between the data and also identify the outliers because this visual representation of data is quite useful for analyzing data, hence the scatter plots of all the three datasets has been drawn below.

Volume - Bug Graph for JM1

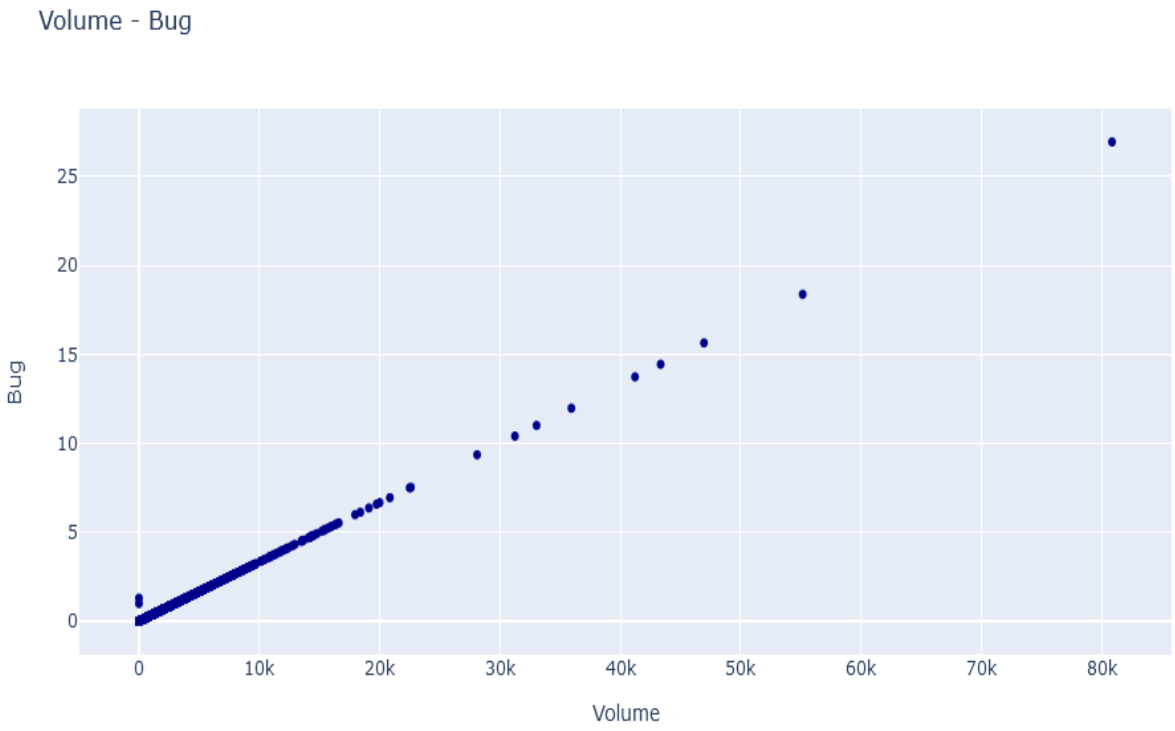

Figure 5.3 - Scatter Plot for JM1

Volume - Bug Graph for PC1

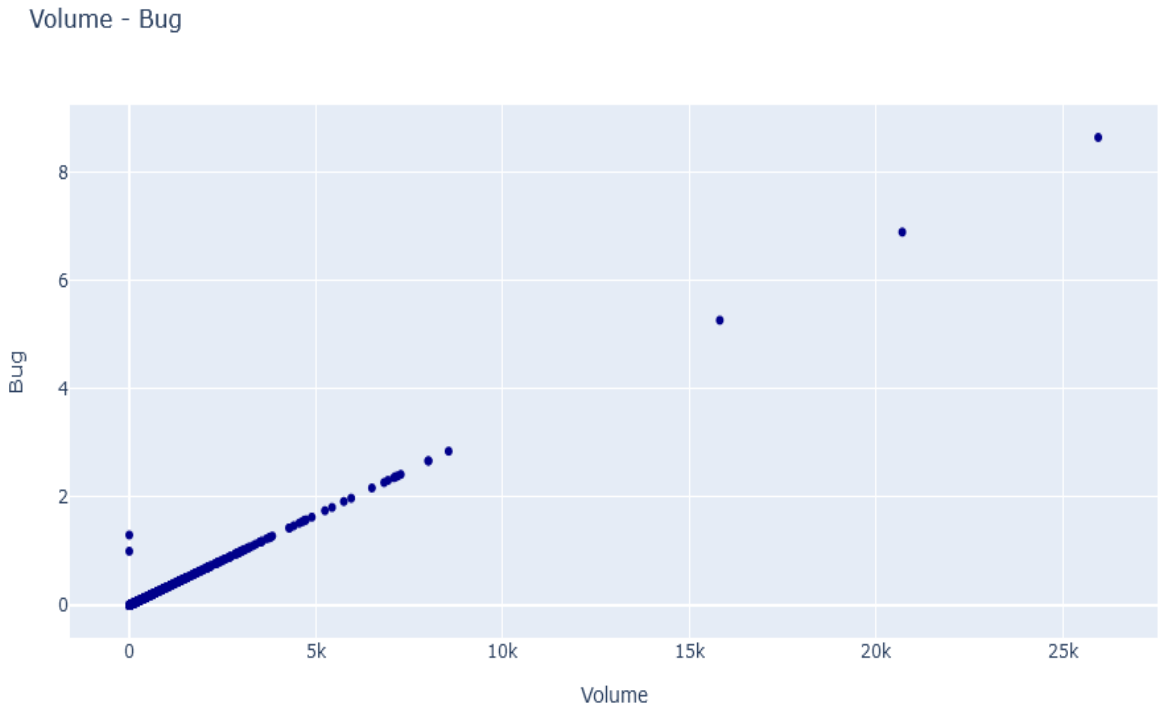

Figure 5.4 - Scatter Plot for PC1

Volume - Bug Graph for CM1

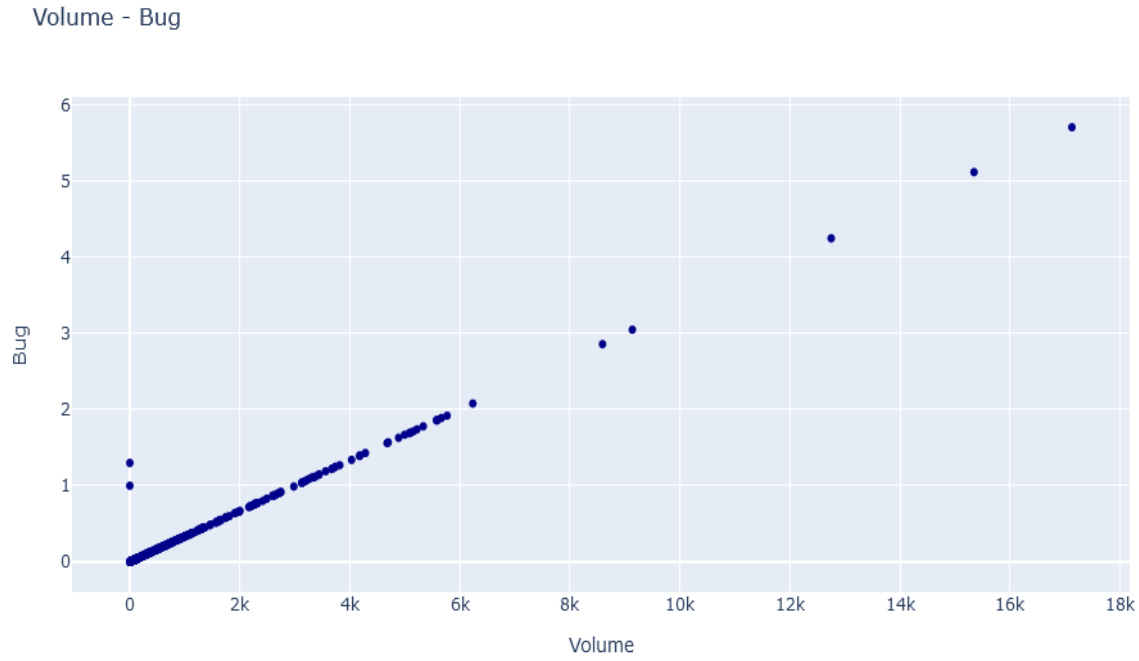

Figure 5.5 - Scatter Plot for CM1

For visually analyzing the outliers we can take advantage of box plots which have been shown in the below fig for each dataset. These box plot are helpful for outlier detection of the given dataset

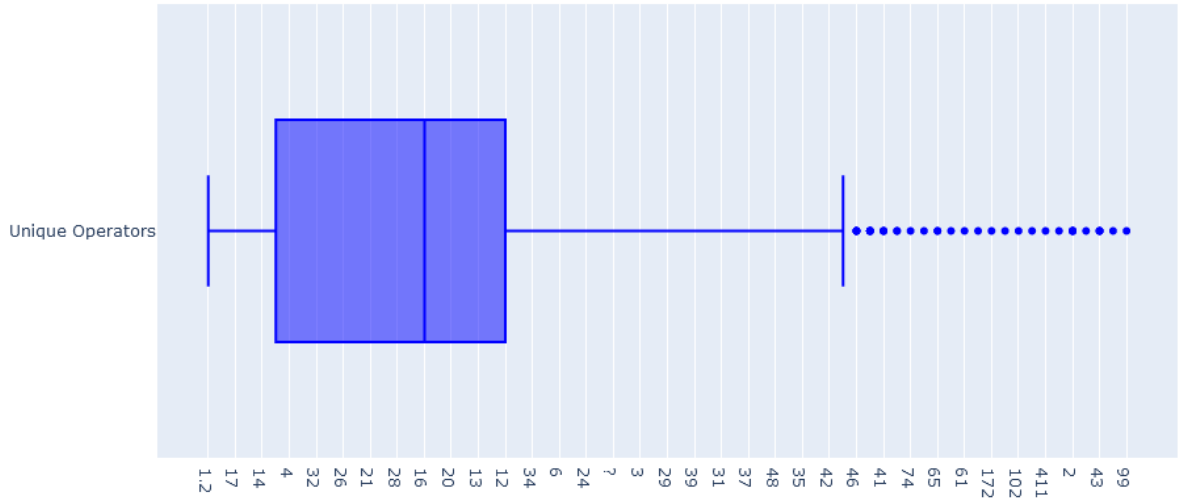

Figure 5.6 - Box Plot for JM1

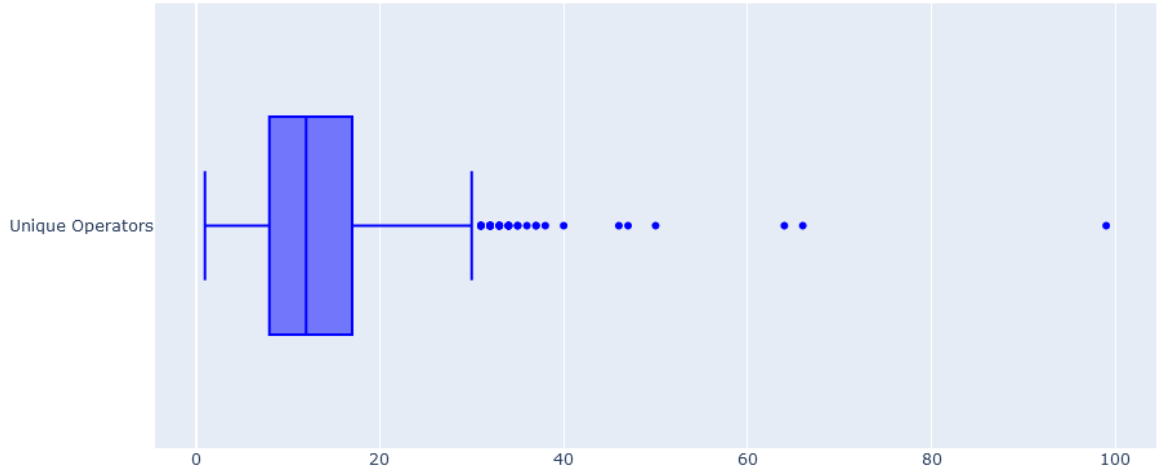

Figure 5.7 - Box Plot for PC1

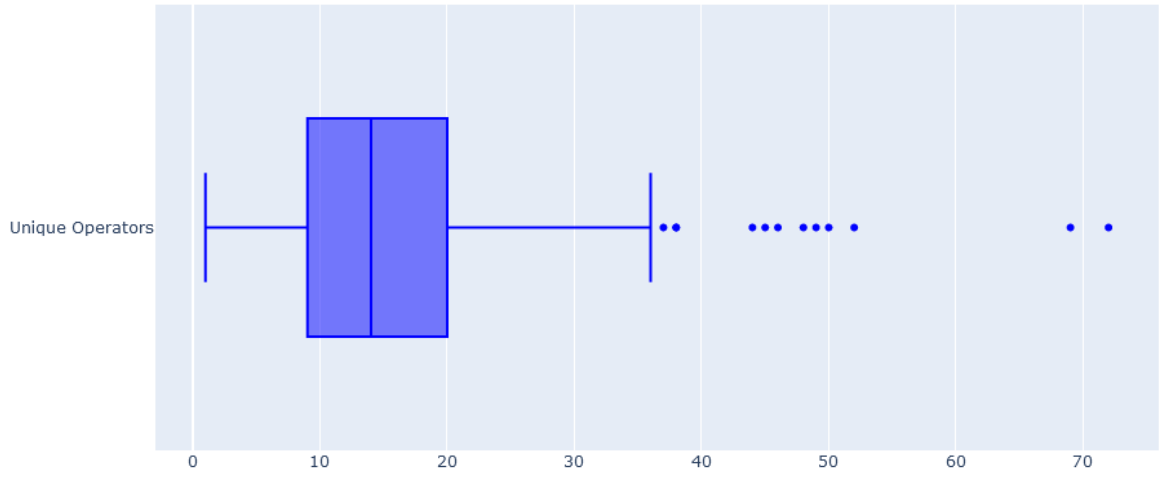

Figure 5.8 - Box Plot for CM1

The accuracy results of the datasets using all different ML models has been stated in table 5.3. From the accuracy data across all the three datasets we can see a pattern i.e the ensemble learners such as RF and GBM provide better results as compared against the remaining 4 models namely NN, DT, SVM, KNN.

Table 5.3 Accuracy of Models

	JM1	PC1	CM1
DT	0.79	0.84	0.82
NN	0.76	0.88	0.88

SVM	0.74	0.91	0.90
KNN	0.78	0.90	0.88
RF	0.81	0.92	0.89
GBM	0.80	0.93	0.88

The accuracy data shows that on the JM1 dataset the best performer was RF with 81% accuracy, on the other hand when analyzing the PC1 the best performer was GBM and for the CM1 the best performer was SVM and RF with 90% and 89% respectively.

CHAPTER - 6

6.1 CONCLUSION

This research focused on the performance of different ML algorithms on the PROMISE dataset i.e JM1, PC1 and CM1 which was chosen for this study, we used different classifiers for analyzing the results and found out that ensemble learners outperformed the normal ML algorithms i.e RF and GBM were most effective as compared against the remaining four classifiers.

The results of the study implies that ML algorithms are helpful and effective but to obtain better results we need to work with ensemble learners which are a little more complex than the normal classifiers hence they can analyze on a deeper level and hence provide a better insight for the data under consideration.

We analyzed the results on the basis of accuracy and AUC-ROC values among all the parametric metrics. We chose these two because of their tendency to show accurate results as compared against recall and F1-score which can sometimes lead to not such accurate results.

6.2 FUTURE SCOPE

The study shows that ensemble learners are a better choice for carrying out software defect prediction hence in the future our goal will be to analyze more ensemble learning techniques and even use different algorithms such deep learning to introduce more complexity at the model level hence enabling the data to be analyzed thoroughly and hence providing with a more detailed study of the software defect prediction on the dataset. With the advent of AI we can leverage the multimodal capabilities of the current advancement and hence can understand better how the software defect prediction can be executed thoroughly and effectively on new datasets and hence providing a new track for researchers to study and analyze.

REFERENCES

[1] K. Gao and T. M. Khoshgoftaar, "A Comprehensive Empirical Study of Count Models for Software Fault Prediction," in *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 223-236

[2] Lessmann, Stefan, et al. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings." *IEEE transactions on software engineering* 34.4 (2008): 485-496.

[3] Zhang, J., & Zhang, X. (2007). Software fault prediction based on neural network. IEEE International Conference on Granular Computing, 327-330.

[4] Breiman, Leo. "Random forests." *Machine learning* 45 (2001): 5-32.

[5] Tian, Y., Lo, D., & Sun, C. (2015). Improved bug localization via combining bug reports with relevant source code entities. *Information and Software Technology*, 59, 93-106.

[6] Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1), 2-13.

[7] Lessmann, Stefan, et al. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings." *IEEE transactions on software engineering* 34.4 (2008): 485-496., 34(4), 485-496.

[8] Ghotra, Baljinder, Shane McIntosh, and Ahmed E. Hassan. "Revisiting the impact of classification techniques on the performance of defect prediction models." *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. IEEE, 2015.

[9] Akiyama, Fumio. "An example of software system debugging." *IFIP congress (1)*. Vol. 71. 1971.

[10] Shen, Vincent Yun, et al. "Identifying error-prone software—an empirical study." *IEEE Transactions on Software Engineering* 4 (1985): 317-324.

[11] Xu, Zhou, et al. "Software defect prediction based on kernel PCA and weighted extreme learning machine." *Information and Software Technology* 106 (2019): 182-200.

[12] Malhotra, Ruchika. "Comparative analysis of statistical and machine learning methods for predicting faulty modules." *Applied Soft Computing* 21 (2014): 286-297.

[13] Moeyersoms, Julie, et al. "Comprehensible software fault and effort prediction: A data mining approach." *Journal of Systems and Software* 100 (2015): 80-90.

[14] Yu, Qiao, et al. "An empirical study on the effectiveness of feature selection for cross-project defect prediction." *IEEE Access* 7 (2019): 35710-35718.

[15] Menzies, Tim, et al. "Assessing predictors of software defects." *Proc. Workshop Predictive Software Models*. 2004.

[16] Ahmed, Md Razu, et al. "The impact of software fault prediction in real-world application: an automated approach for software engineering." *Proceedings of 2020 6th International Conference on Computing and Data Engineering*. 2020.

[17] Jureczko, Marian, and Lech Madeyski. "Towards identifying software project clusters with regard to defect prediction." *Proceedings of the 6th international conference on predictive models in software engineering*. 2010.

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

[19] Tua, Fernando Maruli, and Wikan Danar Sunindyo. "Software defect prediction using software metrics with naïve bayes and rule mining association methods." *2019 5th International conference on science and technology (ICST)*. Vol. 1. IEEE, 2019.

[20] Hassan, Fareeha, et al. "A review on machine learning techniques for software defect prediction." *Technical Journal* 23.02 (2018): 63-71.

[21] Naidu, M. Surendra, and N. Geethanjali. "Classification of defects in software using decision tree algorithm." *International Journal of Engineering Science and Technology* 5.6 (2013): 1332.

[22] Hammouri, Awni, et al. "Software bug prediction using machine learning approach." *International journal of advanced computer science and applications* 9.2 (2018).

[23] Iqbal, Ahmed, et al. "A feature selection based ensemble classification framework for software defect prediction." *International Journal of Modern Education and Computer Science* 10.9 (2019): 54.

[24] Cetiner, Murat, and Ozgur Koray Sahingoz. "A comparative analysis for machine learning based software defect prediction systems." *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2020.

[25] Khalid, Aimen, et al. "Software Defect Prediction Analysis Using Machine Learning Techniques." *Sustainability* 15.6 (2023): 5517.