

Software Defect /Fault Prediction using code metrics

Course No.: MBAZG622T

Course Title : Final Project Work

Final Project Report

Student Name : Harsimrat Singh

BITS ID : 2021mb21621

Program : MBA in Business Analytics

Project Area :

Predictive Analytics

Dissertation / Project Work carried out at:

ASM Technologies , Bangalore (WFH)



Work Integrated Learning Programmes Division (WILPD),

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI,

VIDYA VIHAR , PILANI, RAJASTHAN - 333031.

(November - 2023)

ACKNOWLEDGEMENTS

Its my sincere gratitude and appreciation towards ASM technologies India team , for facilitating and uplifting with healthy and conducive working environment. I am also thankful to faculty members of Bits Pilani, Wilp India for their seamless nurturing , teachings at level best and guidance at every step.

- Krishnan Narayana
- Arunachalam Palaniappan
- Rajesh Prabhakar Kaila

ABSTRACT

The underlying concept of SDP involves utilizing measurements derived from sources like source code and processes related to development and implementation assessing whether these measurements can offer insights into potential defects. I learned that testing activities often consume more time in development. In my project, I explored Software Defect Prediction (SDP), initially relying on simple equations with source code measurements for predictions. However, contemporary approaches emphasize statistical analysis, expert estimations, and machine learning in SDP.

ML encompasses supervised and unsupervised learning. In supervised learning, algorithms, often called classifiers, include decision trees, classification rules, neural networks, and probabilistic classifiers, offering a diverse range of approaches for various tasks

TABLE OF CONTENTS

Section No.	Section Name	Page No.
1	Requirement Statement / Problem Statement	5
2	Project Objectives	5
3	Project Scope and Limitations	6
4	Software Metrics	8
5	Methodology	13
6	Data Preprocessing and Feature Selection	14
7	Architecture	16
8	Components of Software Defect Prediction	17
9	Model Selection, Development and Performance Metrics	21
10	Model Development, Model Training, and Evaluation (Cross-Validations)	25
12	Results, Visualizations and Analysis	33
13	Resource Requirements and Plan for their availability	50
14	Risks and Mitigation Plan	51
15	Issues and Resolutions	52
16	Conclusions and Recommendations	53
Annexure-1	Computer Programs / Code	55
	References	57
	Glossary	58
	Summary of how the feedback for Project Outline and Mid-Semester Project Report have been addressed	61

1. Requirement Statement / Problem Statement

Software defection prediction, also known as defect prediction, aims to forecast potential defects in software systems before they occur. By leveraging code metrics and machine learning techniques, in this project I seeks to develop an advanced prediction tool that assists developers in identifying and mitigating defects proactively.

2. Project Objectives

The primary objective of this MBA project is to create a software defect prediction model using code metrics to forecast potential defects in software code. The project seeks to analyse the correlation between code metrics and defect occurrences, and develop a predictive model that can effectively identify defect-prone areas in the codebase. The following are the specific requirements for the project:

The primary objectives of this project are as follows:

- Develop a defect prediction model using code metrics and machine learning algorithms.
- Data Collection and Preparation
- So, will be using public repositories for fault / defect datasets.
- Identify the most relevant code metrics that significantly influence defect occurrence.
- Evaluate the prediction tool's performance on real-world software projects using datasets from public repositories.
- Assess the impact of the prediction tool on defect prevention and software quality improvement.

3. Project Scope and Limitations

The scope of the project encompasses the development of a software defect prediction solution that utilizes code metrics and machine learning techniques to forecast potential defects in software components or files. The project will focus on creating a proactive defect management system to assist software development teams in identifying defect-prone areas early in the development lifecycle. Also, used machine learning tools to implements various fault prediction techniques, and for model training purposes. The project's scope includes the following key aspects:

1. Data Collection and Preparation:

- Collect historical data from the software development repository, including code metrics (e.g., cyclomatic complexity, lines of code, code churn) and defect status (defective or non-defective) for each software component or file.
- Preprocess the data to handle missing values, outliers, and irrelevant features.

2. Code Metrics Analysis and Feature Selection:

- Analyze the collected code metrics to understand their significance in predicting software defects.
- Select the most relevant and informative code metrics as features for the defect prediction model.

3. Model Development and Evaluation:

- Implement and train machine learning models, such as Random Forest, Logistic Regression, or Support Vector Machines, for defect prediction using the selected code metrics as input features.
- Evaluate the performance of the trained models using appropriate evaluation metrics, including accuracy, precision, recall, F1-score, and AUC-ROC.

4. Defect Prediction and Visualization:

- Deploy the trained defect prediction model to forecast potential defects in new code

changes or components.

- Visualize the model's predictions to identify defect-prone areas of the codebase.

This project will focus on software defection prediction using code metrics at module and file level. It will involve data from secondary sources i.e., public datasets, extracting relevant code metrics, and training machine learning models to predict defects.

In this MBA project, the data methodology involves the collection and utilization of secondary data that has already been collected and recorded. The project will leverage existing datasets containing code metrics and corresponding defect labels from software development repositories. These repositories store historical information about software code changes, allowing us to analyse patterns and correlations between code characteristics and the occurrence of defects.

This Predictive Analytics project will not include (Limitations):

The project will not address defects arising from external factors or environmental issues.

Creating a tool for data analysis, similar to the process carried out within the Weka environment. Additionally, the prediction tool's performance will be evaluated on a limited set of software projects, and the generalizability of the models will be assessed.

1. Actual Code Fixing: The project focuses on defect prediction rather than fixing defects. While the tool will identify defect-prone areas, the actual fixing of defects will be performed by the development team separately.
2. Integration with Specific Development Environments: The project does not involve the integration of the defect prediction tool with specific software development environments.

4. Software Metrics

Software Metrics : During the software development, stakeholders/ team members are required to evaluate the quality of the software and the process used to build it.

In order to do so, it is required to capture various attributes of software artifacts at different phases of software development and need to measure them to evaluate the software and its process. Software metrics helps to quantify the attributes of software artifacts, and using these quantitative values, in this project I will be evaluating the software quality.

Software metric can be defined as a measurement-based technique applied to the software process, product, and services to supply the engineering and management-based information about the software and can be used to provide feedback to improve the software process, product, and services.

In terms of software fault prediction, software metrics can be classified into two broad categories:

- i. Product Metrics
- ii. Process Metrics.

Product metrics: Product metrics are categorized into three groups: traditional metrics or static code metrics, object oriented metrics, and dynamic metrics.

Static Code Metrics derived directly from the source code of a software system. They provide insights into the characteristics of the codebase. One common set of static code metrics includes the following:

Source Lines Of Code (SLOC): This metric family is centered around counting the lines in source code files. It includes various sub-metrics.

These metrics help in assessing the size, complexity, and maintainability of the source code. They also assist in identifying code lines that are relevant to the software's functionality, code structure, and documentation.

- Size Metrics
- Quality Metrics
- System Complexity Metrics

Cyclomatic Complexity Number (CCN), commonly referred to as the McCabe metric, serves as an indicator of a module's decision structure complexity.

This divergence occurs when the source code contains constructs like if statements, for loops, while loops, switch cases, catch blocks, logical operators such as && and ||, or the ternary operator ?.

Object-oriented metrics: These metrics relate to different aspects encompassing factors such as coupling, cohesion, and inheritance. They are designed to capture various object-oriented characteristics.

I am using the CK Metric Suite, which includes eight distinct metrics. Six of these metrics are inherited the CK metric set, and two metrics have been outlined and used in this.

CK metrics suite: “Coupling between Object class (CBO), Lack of Cohesion in Methods (LCOM), Depth of Inheritance Tree (DIT), Response for a Class (RFC), Weighted Method Count (WMC) and Number of Children (NOC)”.

The Coupling Between Objects (CBO) metric quantifies the number of other classes with which a given class is interlinked, meaning it measures within the class in question.

Weighted Methods per Class (WMC): This metric counts the number of methods within a class, providing insights into the complexity and size of the class.

Depth of Inheritance Tree (DIT): DIT calculates the length of the maximum path from a class to its root class within the inheritance hierarchy. It helps in understanding the depth of class inheritance.

Number of Children (NOC): NOC quantifies the number of subclasses derived from a particular class. It indicates the class's position in the inheritance hierarchy.

Lack of Cohesion in Methods (LCOM): It reflects the cohesion of methods within a class.

Additional OO metrics employed include:

Afferent Couplings (Ca): Ca under evaluation, similar to the concept of Coupling Between Objects (CBO). It measures the class's incoming dependencies.

Number of Public Methods (NPM): NPM quantifies the number of methods within the class that are declared as "public." It provides insights into the class's interface and accessibility of its methods.

These metrics help in assessing various aspects of object-oriented software classes, such as their complexity, inheritance depth, relationships, cohesion, and interface.

MOODS metrics suite: “Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Polymorphism Factor (PF), Coupling Factor(CF)”.

Wei Li and Henry metrics suite: “Coupling Through Inheritance, Coupling Through Message passing (CTM), Coupling Through ADT (Abstract Data Type), Number of local Methods (NOM), SIZE1 and SIZE2”.

Lorenz and Kidd’s metrics suite: “PIM, NIM, NIV, NCM, NCV, NMO, NMI, NMA, SIX and APPM”.

Bansiya metrics suite: “DAM, DCC, CIS, MOA, MFA, DSC, NOH, ANA, CAM, NOP and NOM”.

Dynamic metrics : are calculated based on the execution of a program and help identify objects that exhibit the highest runtime coupling and complexity during execution. These metrics provide various insights into the quality of the software design.

Yacoub metrics suite: “Export Object Coupling (EOC) and Import Object Coupling (IOC)”.

Arisholm metrics suite: “IC_OD, IC_OM, IC_OC, IC_CD, IC_CM, IC_CC, EC_OD, EC_OM, EC_OC, EC_CD, EC_CM, EC_CC”.

Mitchell metrics suite: “Dynamic CBO for a class, Degree of dynamic coupling between two classes at runtime, Degree of dynamic coupling within a given set of classes, RI, RE, RDI, RDE”.

In dynamic metrics, the allocation of tasks among different software components can be determined during the software development's design phase. This allocation of responsibilities can significantly influence the software's quality.

In my project, have undertaken efforts to carry out dependency mapping and identify potential fault-prone source files and packages.

Process metrics: it is categorized into various subcategories to provide a more detailed analysis:

Code churn metrics: “Total LOC, Churned LOC, Deleted LOC, File count, Weeks of churn, Churn count and Files churned”.

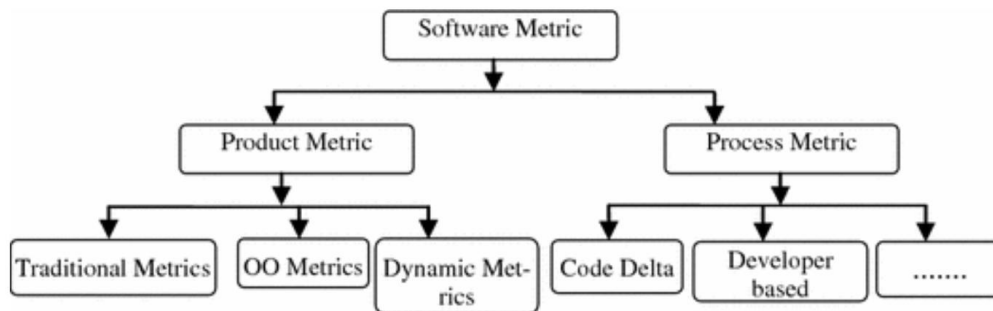
Change metrics: “Revisions, Refactoring’s, Bugfixes, Authors, LOC added, Max LOC Added, Ave LOC Added, LOC Deleted, Max LOC Deleted, Ave LOC Deleted, Codechurn, Max Codechurn, Ave Codechurn, Max Changeset, Ave Changeset and Age”.

Developer-based metrics: “Personal Commit Sequence, Number of Commitments, Number of Unique Modules Revised, Number of Lines Revised, Number of Unique Package Revised, Average Number of Faults Injected by Commit, Number of Developers Revising Module and Lines of Code Revised by Developer”.

Requirement metrics: “Action, Conditional, Continuance, Imperative, Incomplete, Option, Risk level, Source and Weak phrase”.

Network metrics: “Betweenness centrality, Closeness centrality, Eigenvector Centrality, Bonacich Power, Structural Holes, Degree centrality and Ego network measure”.

Classification of Software Metrics



Test metrics

Applied specifically to the source code of the tests. Moreover, Nagappan introduced a dedicated test metric suite named the Software Testing and Reliability Early Warning metric suite (STREW), comprising nine metrics organized into three distinct families.

STREW serves the purpose of offering an early-stage estimation of the software's quality and the identification of modules that are more likely to contain defects.

5. Methodology

Software Fault Dataset

The software fault dataset serves a dual purpose—it serves as a training dataset for training and act as testing dataset for evaluating the model's performance.

These repositories can be categorized into three main types:

1. Private/Commercial Datasets Repository: These repositories are privately owned and maintained by commercial entities.
2. Partially Public/Freeware Dataset Repository: These repositories are partially public and may be freely accessible or available as freeware datasets.
3. Public Datasets Repository: These repositories are entirely public, making their data freely accessible to the broader community.

Here in this project, I am using public fault/defect data repository.

In this project I am using PROMISE data repo.

<http://promise.site.uottawa.ca/SERepository/>

<https://github.com/feiwww/PROMISE-backup/tree/master/bug-data>

<https://www.kaggle.com/datasets/toniesteves/desharnais-dataset>

The datasets within these repositories pertain to open-source software projects. Details about both the number of faults and the severity of those faults in software modules, and there may be variations in the extent of information available from one dataset to another.

6. Data Preprocessing and Feature Selection:

I have utilized high-quality datasets from public repositories, such as PROMISE and ECLIPSE, for software fault prediction work. Proper handling and cleaning of these datasets before using them to build a software fault prediction model is crucial for reliable results.

In my data preparation process, I have addressed various quality-related issues:

1. **Outliers:** These are those observations which significantly deviate from the general dataset pattern. Careful detection and removal of outliers are necessary for fault prediction. However, it's important to be cautious, as some outliers may contain valuable fault information, and their removal can lead to a loss of important insights.
2. **Missing Values:** Missing values in the dataset can occur due to human error or unavailability of information. Different fault prediction models handle missing values differently, either by ignoring corresponding observations or employing corrective measures to address them.
3. **Repeated Values:** Removing one of the attributes with the same values helps in building an accurate fault prediction model.
4. **Redundant and Irrelevant Values:** These occur when the same attribute or feature is present in multiple observations with the same class label. They can lead to overfitting and reduced model performance, making it essential to handle such data points.
5. **Class Imbalance:** Class imbalance happens when there are significantly fewer positive class observations (minor class) compared to negative class observations (major class). This imbalance can bias the model towards the major class, leading to poor results for the minor class. Techniques like oversampling or undersampling can be employed to address this issue.
6. **Data Shifting Problem :** This issue can adversely affect prediction model performance. Ensuring that the datasets adhere to the same joint distribution.
7. **High data dimensionality,** which occurs when dataset observations ($p > m$), can lead to decreased performance in fault prediction models, higher misclassification errors, and increased computational costs and memory usage during model development.

Wrappers: utilize machine learning algorithms to select attributes and assess the usefulness of different features. Wrappers typically operate more slowly but can be applied to any classification problem.

Filters: Filter-based methods utilize heuristics based on data for assessing features. Filters operate faster and are more suitable for selecting attributes in large feature sets. However, they often require the classification problem to be discrete.

Unlike other filter algorithms that evaluate individual features, “Correlation-based Feature Selection” assesses calculates the merit of a feature subset, considering the trade-off between a feature group's predictiveness and the redundancy between the features in the subset. Features highly correlated with other features in the feature set are excluded from the reduced feature set.

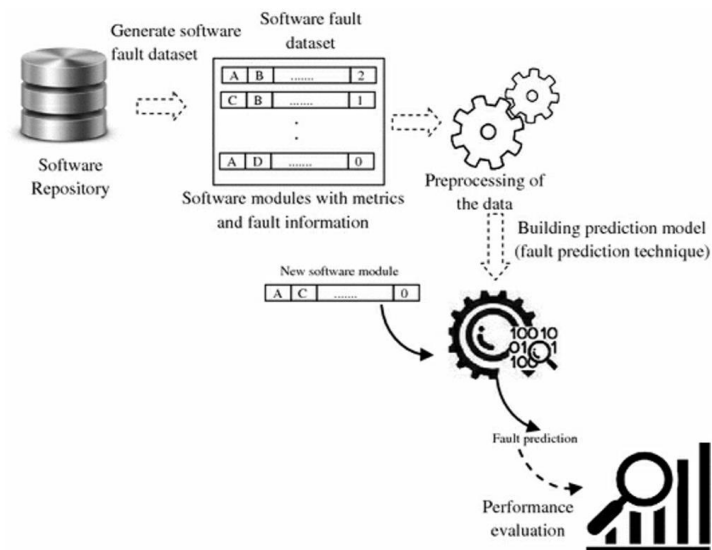
By applying such feature selection methods, you can effectively reduce high data dimensionality, improve model performance, and reduce computational overhead during fault prediction model development.

7. Architecture

In this MBA project, the data methodology involves the collection and utilization of **secondary data** that has already been collected and recorded. The project will leverage existing datasets containing code metrics and corresponding defect labels from software development

repositories. These repositories store historical information about software code changes, allowing us to analyse patterns and correlations between code characteristics and the occurrence of defects.

Architecture of software defect/ fault prediction process:



Before moving directly towards chosen methodology,

8. Components of Software Fault/Defect Prediction

Meta information about a software project - comprises fundamental details that can offer additional context and facilitate working with the project's data. This information typically includes:

- Domain
- Programming Language
- Number of Releases or Versions

Considering this meta information when building fault prediction models is crucial. By taking into account these variables/factors related to the software project, you can enhance the accuracy and applicability of your fault prediction models.

The source of data is a crucial factor that provides insights into the domain of the software project and how it is used. It can encompass various characteristics, such as whether the software project is open-source or commercial, its specific type (e.g., web browser or integrated development).

The source of the fault dataset - can indeed impact the data shifting issue. This issue arises when the training and testing of the models are carried out on different datasets, leading to differences in data distributions. Handling this data shifting problem is essential to ensure the accurate and reliable performance of fault prediction models, as variations in data sources can affect and predict faults.

The maturity of a software system - is a measure of how long the software has been in existence and how many versions or releases it has gone through. Each new release typically corresponds to changes or the addition of new functionality in the software. A mature system has a longer history of development and refinement, which often leads to a lower chance of errors in the system.

More mature systems tend to have more stable and well-tested code, which can affect the distribution of faults and the characteristics of the data. Therefore, understanding the maturity of the software system is important when building and applying defect prediction model.

The size of a software project : There is a common belief that as the LOC value increases, the likelihood of the software containing faults also increases.

In general, fault prediction models based on larger software projects tend to produce better performance. This is because larger projects provide more data points and a broader range of fault-related information, which can lead to more accurate predictions. However, it's essential to consider the quality of data and the potential complexities associated with larger projects when building and applying these models.

Application Domain: The characteristics and dynamics of software development can vary significantly across different domains, influencing the model's performance.

Binary Class Classification & Confusion Matrix Importance: the goal is to categorize software modules as either faulty or non-faulty. Modules with multiple predicted faults are marked as faulty, while those with zero predicted faults are labeled as non-faulty.

Performance evaluation measures derived from confusion matrices, such as accuracy, precision, recall, and F-measure, are commonly employed to assess model performance. Additionally, metrics like AUC (area under the ROC curve) and g-means have been utilized.

Class-level software metrics - have proven to be more effective for various regression techniques.

Various Performance Evaluation Measures : are essential for assessing effectiveness prediction model. These measures are used to evaluate how well a trained model performs. Performance evaluation measures are categorized as: numeric measures and graphical measures.

Numeric Performance Evaluation Measures: These measures provide a quantitative description of a prediction model's performance. Some commonly used numeric performance evaluation measures include:

1. Accuracy: The proportion of correctly predicted instances to the total number of instances.
2. Precision: The ratio of true positive predictions to the total number of positive predictions, indicating

the model's ability to correctly identify positive cases.

3. Recall: The ratio of true positive predictions to the total number of actual positive instances, measuring the model's ability to find all positive cases.
4. F-measure: The harmonic mean of precision and recall, providing a balance between the two.
5. False Positive Rate: The proportion of false positive predictions to the total number of actual negative instances.
6. G-means: A geometric mean of sensitivity and specificity, used for imbalanced datasets.
7. False Negative Rate: The proportion of false negative predictions to the total number of actual positive instances.
8. J-coefficient: Measures the relationship between true positive, true negative, false positive, and false negative predictions.
9. Specificity, also known as the true negative rate, is the proportion of true negative predictions to the total number of actual negative instances. It measures how well a fault prediction model can correctly identify non-faulty instances.
10. Average Absolute Error: A measure of the average difference between predicted and actual values for regression tasks.
11. Average Relative Error: A measure of the average relative difference between predicted and actual values for regression tasks.

Graphical Performance Evaluation Measures: These measures visually represent information for easier understanding. Examples of graphical performance evaluation measures include:

12. ROC Curve (Receiver Operating Characteristic): A graphical representation of the trade-off between true positive rate and false positive rate at various thresholds.
13. Precision-Recall (P-R) Curve: A graphical representation showing precision and recall across different classification thresholds.
14. Cost Curve: A graph showing the trade-off between model performance and the associated cost, useful for decision-making in cost-sensitive applications.

These performance evaluation measures help in assessing the strengths and weaknesses of a fault prediction model and are essential for choosing the most appropriate model.

9. Model Selection, Development and Performance Metrics

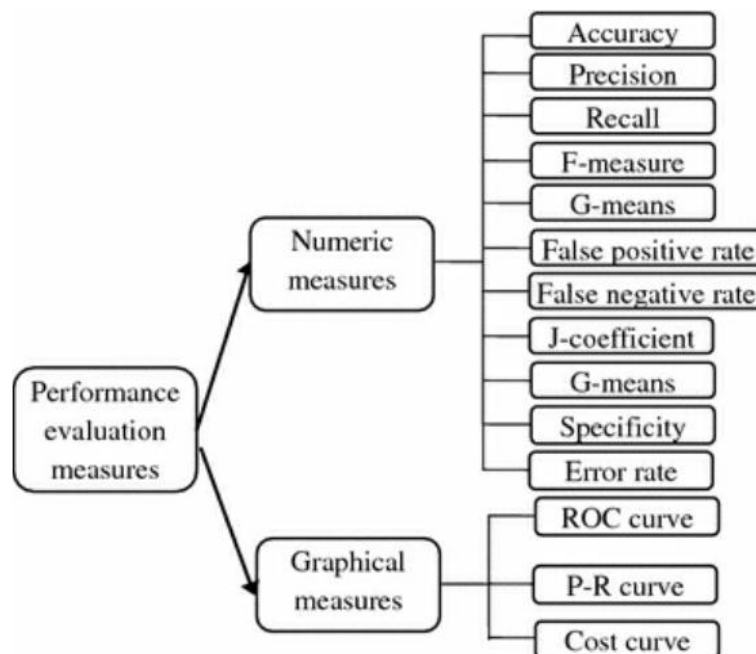
To assess the discriminative capabilities of various metric sets in my project, I employed a range of machine learning (ML) algorithms, each running with their default settings within the WEKA

environment. WEKA, a versatile tool for data mining and machine learning, provided a comprehensive suite of algorithms and tools for my analysis.

The selection of ML algorithms for constructing prediction models in this bug prediction project was made with careful consideration of their suitability for the task. These algorithms were executed with their standard configurations, and they were utilized to create models aimed at identifying and predicting software bugs or faults. My objective was to evaluate and compare the effectiveness of different metric sets and the associated algorithms in predicting software defects, aligning with the goals of my project.

Performance Evaluation Measures

Below table is data related to various metrics with fault information for different modules within the projects of the PROMISE repository software fault datasets:



Numeric Measures : The foundation of numeric evaluation measures in fault prediction relies on the confusion matrix. As per the below chart , confusion matrix provides four different information.

Predicted \ Actual	Positive	Negative
	True Positive (TP)	False Positive (FP)
Positive		
Negative	False Negative (FN)	True Negative (TN)

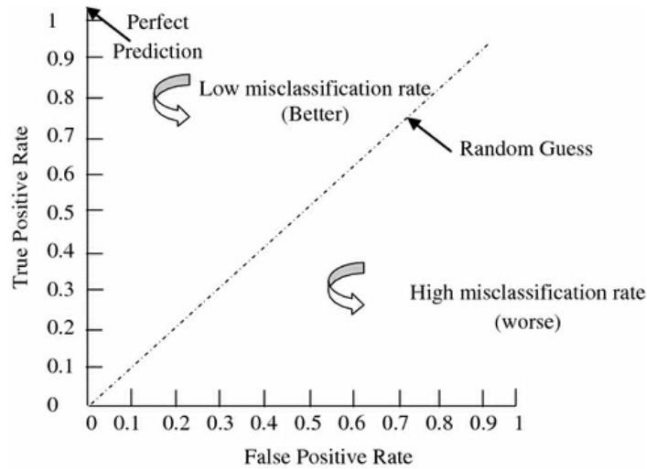
The confusion matrix is a tool for evaluating and fine-tuning software defect prediction models. It helps to understand the model's performance in terms of false positives, false negatives, and correct predictions, ultimately aiding in better decision-making for defect detection and software quality improvement.

In addition to metrics like precision and recall, the confusion matrix provides a detailed view of the model's behaviour, making it easier to explain model predictions to stakeholders.

The confusion matrix is a valuable tool for selecting the most suitable machine learning model and tuning its parameters. For example, you might choose a model that minimizes false negatives if missing a defect is very costly in your context.

Graphical Measures: Graphical measures serve the purpose of visual illustration. These visual representations are calculated using the confusion matrix.

THE ROC Curve :



With this ROC curve graph, I calculated the ratio of the faulty modules predicted correctly to non-faulty modules predicted incorrectly for performance estimation.

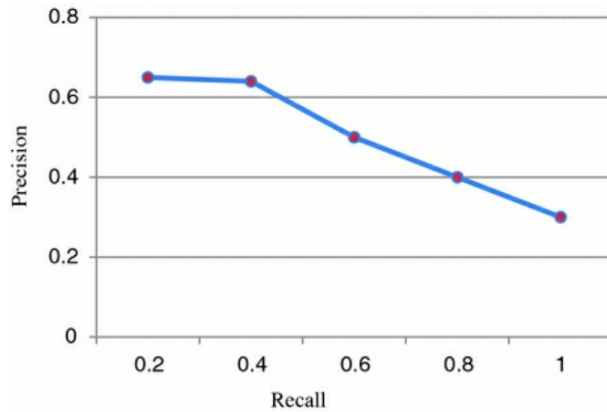
Ideally, The area under the ROC curve (AUC) serves as a measure to assess a model's predictive capability. It is determined by selecting a specific threshold value. An AUC value of 1 signifies that the model has a 100% prediction capability, whereas an AUC value of 0 indicates that the model has a 0% prediction capability.

PR Curve :

Also known as a Precision-Recall curve, provides insights into the trade-off between precision and recall in a prediction model. The graph illustrates the recall value on the x-axis and precision value on the y-axis.

In a PR curve, optimal performance is indicated by high values of both recall and precision. An ideal prediction model would have a PR curve passing through the upper right corner, indicating 100% precision and 100% recall.

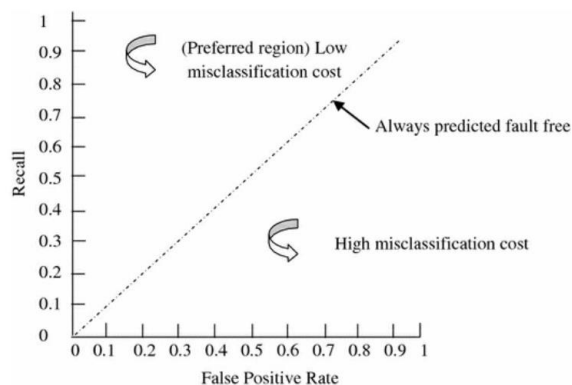
The proximity of the PR curve to the upper right corner signifies the quality of the prediction model, with models closer to this corner considered superior in performance.



Cost Curve:

Cost curve depicts the cost of misclassification of software modules of a fault prediction model. Y-axis of cost curve plots the probability of detection (PD), also known as recall and x-axis plots the probability of false alarm.

It shows the difference between the maximum and minimum values of cost for misclassifying the faulty modules, so this is an example of cost curve. The diagonal line between (0,0) and (1,1) shows that prediction model predicted all the modules as fault-free. The diagonal line between (0,1) and (1,0) shows that prediction model predicted all the modules as faulty. The horizontal line between (0,1) and (1,1) shows that prediction model misclassified all the modules.



10. Model Development, Model Training, and Evaluation (Cross-Validations)

Fault Dataset Repositories

In the course of my project work, I made use of two significant repositories to access essential datasets for software fault prediction:

PROMISE (Predictor Models In Software Engineering): The PROMISE software dataset repository, accessible at <http://openscience.us/repo/> , served as a valuable resource. This repository contains a wide range of datasets related to software faults and various other aspects, including efforts estimation and refactoring.

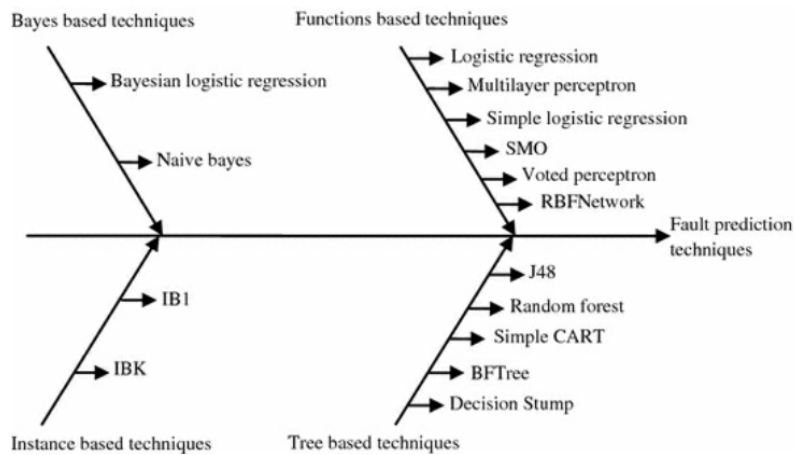
Eclipse Bug Data Repository: This public repo for bug(s) data, available at <https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse> , was primarily designed to collect and maintain fault datasets from the Eclipse project. Within this repository, I found datasets that pertain to three different versions of the Eclipse project, namely Eclipse-2.0, Eclipse-2.1, and Eclipse-3.0. All these versions with Eclipse bug data provide valuable data on a variety of source code and structural metrics. These repositories have played a pivotal role in my project, providing access to the critical datasets required for the analysis and modeling of software faults.

Fault Prediction Techniques

Within the scope of my project, I have explored a range of machine learning techniques and statistical methods to analyze and model software faults. These techniques encompass:

- Machine Learning Techniques
- Statistical Techniques

These methods were employed to build predictive models and assess their effectiveness in predicting software faults. Each technique offers unique advantages and may be suitable for different aspects of the project.



In my project, I've employed a variety of fault prediction techniques.

Practical Setup

Performance Evaluation Measures

In my project, the primary objective is to construct models classify instances to assess the performance of these models, I employ a set of standard performance measures. The performance measures used in the project, such as accuracy, precision, recall, F-measure, and AUC, are derived from the confusion matrix. These measures are vital for assessing the effectiveness of the fault prediction techniques and quantifying their performance in classifying software modules as faulty or non-faulty.

Accuracy: a significant metric in my project, serves to quantify the balance between precision and recall. It calculates a harmonic mean of these two performance measures, offering valuable insights into the effectiveness of the fault prediction models.

Precision: This measure evaluates the relevancy of the results by assessing the number of correctly predicted faulty instances out of all instances predicted as faulty.

Recall: Also known as the probability of detection, recall measures how many true faulty instances were correctly predicted out of all faulty instances in the dataset.

F-Measure: It quantifies providing a harmonic mean between precision and recall.

AUC (Area Under ROC Curve): This performance measure assesses the model's ability to distinguish between the two classes (faulty and non-faulty) by analyzing the Receiver Operating Characteristic (ROC) curve.

Fault Datasets Used in this Project

I utilized the PROMISE data repository that contains a wide range of defects in the datasets , making it a valuable resource for my project. It currently hosts fault datasets from 65 different software projects and their releases.

PROMISE data repository datasets come in various sizes, ranging from datasets with only a few software modules to those with thousands of modules. Despite the differences in size, all these datasets share a common set of software metrics. This consistency in metrics allows for a standardized approach to evaluating and comparing fault prediction techniques.

To evaluate the faults by building model for prediction, I considered multiple releases of each software project, ensuring a comprehensive assessment of the techniques' performance across different versions.

In my project, I conducted an analysis of various fault datasets obtained from PROMISE repository.

By calculating this percentage, I was able to assess the proportion of modules affected by faults in each dataset, providing valuable information for my project's analysis and evaluation.

Table of Software Fault Datasets Used

No. Of Fault datasets	Dataset Name	Release	# Non-commented LOC	Total No. Of Modules	Total No. Of Faulty Modules	% of Faulty Modules (%)
1	Ant	Ant-1.7	208 KLOC	745	166	22.25
2	Camel	Camel-1.0	33 KLOC	340	13	3.81

3		Camel-1.2	66 KLOC	609	126	35.47
4		Camel-1.4	98 KLOC	873	145	16.61
5		Camel-1.6	113 KLOC	966	188	19.46
6	Ivy	Ivy-1.1	27 KLOC	112	63	56.25
7		Ivy-1.4	59 KLOC	242	16	6.61
8		Ivy-2.0	87 KLOC	353	40	11.33
9	Jedit	Jedit-3.2	128 KLOC	273	90	32.97
10		Jedit-4.0	144 KLOC	307	75	24.43
11		Jedit-4.1	153 KLOC	313	79	25.24
12		Jedit-4.2	170 KLOC	368	48	13.04
13		Jedit-4.3	202 KLOC	493	11	2.23
14	Log4 j	Log4 j-1.0	21 KLOC	136	34	25
15		Log4 j-1.1	19 KLOC	110	37	33.64
16		Log4 j-1.2	38 KLOC	206	189	91.75
17	Lucene	Lucene-2.0	50 KLOC	196	91	46.43
18		Lucene-2.2	63 KLOC	248	144	58.06
19		Lucene-2.4	102 KLOC	341	203	59.53
20	Poi	Poi-1.5	55 KLOC	238	141	59.24
21		Poi-2.0	93 KLOC	315	37	11.75
22		Poi-2.5	119 KLOC	386	248	64.25
23		Poi-3.0	129 KLOC	443	281	63.43
24	Synapse	Synapse-1.0.	28 KLOC	157	16	10.19

25		Synapse-1.1	42 KLOC	223	60	26.91
26		Synapse-1.2	53 KLOC	257	87	33.85
27	Velocity1	Velocity1-1.4.0	51 KLOC	198	146	75.63
28		Velocity1-1.5.	53 KLOC	215	142	66.04
29		Velocity-1.6	57 KLOC	230	78	33.91
30	Xalan	Xalan-2.4	225 KLOC	724	111	15.33
31		Xalan-2.5	304 KLOC	804	387	48.13
32		Xalan-2.6	411 KLOC	886	411	46.39
33		Xalan-2.7	428 KLOC	910	989	98.68

No. Of Fault datasets	Dataset Name	Release	# Non-commented LOC	Total No. Of Modules	Total No. Of Faulty Modules	% of Faulty Modules (%)	
13		Jedit-4.3	202 KLOC	493	11	2.23	CATEGORY 1 (Less than 10 % of faulty modules)
2	Camel	Camel-1.0	33 KLOC	340	13	3.81	
7		Ivy-1.4	59 KLOC	242	16	6.61	
24	Synapse	Synapse-1.0	28 KLOC	157	16	10.19	CATEGORY 2 (10% - 20% of faulty modules)
8		Ivy-2.0	87 KLOC	353	40	11.33	
21		Poi-2.0	93 KLOC	315	37	11.75	
12		Jedit-4.2	170 KLOC	368	48	13.04	
30	Xalan	Xalan-2.4	225 KLOC	724	111	15.33	
4		Camel-1.4	98 KLOC	873	145	16.61	

5		Camel-1.6	113 KLOC	966	188	19.46	
1	Ant	Ant-1.7	208 KLOC	745	166	22.25	CATEGORY 3 (20% - 30% of Faulty Modules)
10		Jedit-4.0	144 KLOC	307	75	24.43	
14	Log4 j	Log4 j-1.0	21 KLOC	136	34	25	
11		Jedit-4.1	153 KLOC	313	79	25.24	
25		Synapse-1.1	42 KLOC	223	60	26.91	
9	Jedit	Jedit-3.2	128 KLOC	273	90	32.97	CATEGORY 4 (Above 30%)
15		Log4 j-1.1	19 KLOC	110	37	33.64	
26		Synapse-1.2	53 KLOC	257	87	33.85	
29		Velocity-1.6	57 KLOC	230	78	33.91	
3		Camel-1.2	66 KLOC	609	126	35.47	
32		Xalan-2.6	411 KLOC	886	411	46.39	
17	Lucene	Lucene-2.0	50 KLOC	196	91	46.43	
31		Xalan-2.5	304 KLOC	804	387	48.13	
6	Ivy	Ivy-1.1	27 KLOC	112	63	56.25	
18		Lucene-2.2	63 KLOC	248	144	58.06	
20	Poi	Poi-1.5	55 KLOC	238	141	59.24	
19		Lucene-2.4	102 KLOC	341	203	59.53	
23		Poi-3.0	129 KLOC	443	281	63.43	
22		Poi-2.5	119 KLOC	386	248	64.25	
28		Velocity-1.5	53 KLOC	215	142	66.05	
27	Velocity	Velocity-1.4	51 KLOC	197	147	74.62	
16		Log4 j-1.2	38 KLOC	206	189	91.75	
33		Xalan-2.7	428 KLOC	910	989	98.68	

I'd like to emphasize that I excluded nineteen fault datasets that contained fewer than 100 software modules from my analysis. The reason for this exclusion was the challenge of obtaining reliable results with datasets of very small size, particularly because some fault prediction techniques require larger datasets for effective training.

To ensure the quality of the datasets and streamline the analysis, I performed pre-processing. My primary focus throughout this project has been classification of software faults, allowing me to concentrate on the specific task at hand.

In my project, I performed a transformation, distinguishing if a module contains one or more faults, it is labelled as “faulty” or else , categorized as “non-faulty”. This transformation is consistently applied to all the fault datasets used in the analysis.

I recognized that techniques might be influenced by the proportion of faulty modules within the datasets. To gain a comprehensive understanding of this impact, I categorized the datasets into 4 distinct categories. The categorization criteria were as follows:

- Category 1: Fault datasets with less than 10 % as faulty.
- Category 2: with faulty modules between 10 % and 20% of the total.
- Category 3: with faulty modules representing 20 % to 30% of the total.
- Category 4: with more than 30 % of the modules classified as faulty.

This categorization allowed me to explore prediction techniques was influenced by the varying proportions in datasets. This analysis helped me draw valuable insights into the relationship between fault dataset characteristics and prediction model performance.

Test Execution

In my practical setup, I employed the WEKA machine learning tool for building and evaluating various Fault Prediction Models. These tests were executed using default parameter values for the different fault prediction techniques available within WEKA.

To ensure robustness of the evaluation, I adopted a ten fold cross validation scheme. Here's how the internal working of this scheme functioned:

1. The fault datasets were divided into ten separate parts.
2. During each iteration, nine of these parts were utilized as training dataset.
3. The remaining one part served as test dataset.
4. Same process is repeated ten times, with each part taking a turn as the testing dataset.
5. The results from all ten iterations were then averaged, providing assessment of the model(s)

performance.

This approach helped to ensure that the evaluation was robust and that the fault prediction models were rigorously assessed using cross-validation.

11. Results, Visualizations and Analysis

Results and Analysis 1 of 4

For the results obtained from the practical test execution and analysis , I have drafted in excel files (sort of reports).

Each table presented below corresponds to a specific category of software fault datasets. These measures serve as valuable indicators of the model's performance and effectiveness in predicting software faults within each category of datasets.

Results of Category -1

Technique	Metric	Min.	Average	Max.
Bayesian Logistic	Accuracy	90	93.12	97.76
	Precision	0.82	0.89	0.95
	Recall	0.9	0.94	0.98
	F Measure	0.85	0.91	0.97
	AUC	0.5	0.5	0.52
Naive Bayes	Accuracy	80.15	87.94	93.69
	Precision	0.87	0.9	0.97
	Recall	0.8	0.88	0.94
	F Measure	0.83	0.89	0.95
	AUC	0.59	0.69	0.81
Logistic Regression	Accuracy	88.93	92.2	96.54
	Precision	0.83	0.9	0.97
	Recall	0.89	0.92	0.97
	F Measure	0.86	0.91	0.96
	AUC	0.48	0.65	0.8
Multilayer Perceptron	Accuracy	88.48	92.26	97.35
	Precision	0.85	0.9	0.96
	Recall	0.87	0.92	0.97
	F Measure	0.87	0.9	0.96
	AUC	0.38	0.62	0.77
Simple Logistic	Accuracy	89.84	93.2	97.15
	Precision	0.81	0.9	0.96
	Recall	0.9	0.93	0.97
	F Measure	0.86	0.91	0.96

	AUC	0.59	0.69	0.8
SMO	Accuracy	90	93.11	97.76
	Precision	0.81	0.87	0.96
	Recall	0.9	0.93	0.98
	F Measure	0.86	0.91	0.97
	AUC	0.5	0.5	0.5
Voted Perceptron	Accuracy	86.72	91.32	96.95
	Precision	0.81	0.88	0.96
	Recall	0.81	0.88	0.96
	F Measure	0.49	0.5	0.53
IB1	Accuracy	84.54	89.67	95.93
	Precision	0.86	0.9	0.96
	Recall	0.85	0.9	0.96
	F Measure	0.85	0.9	0.96
	AUC	0.5	0.58	0.64
IBK (k = 5)	Accuracy	87.42	92.38	97.76
	Precision	0.82	0.88	0.96
	Recall	0.87	0.92	0.98
	F Measure	0.84	0.91	0.97
	AUC	0.55	0.69	0.75
J48	Accuracy	89.24	92.56	97.76
	Precision	0.85	0.89	0.96
	Recall	0.89	0.93	0.98
	F Measure	0.86	0.91	0.97
	AUC	0.47	0.55	0.66
Decision Stump	Accuracy	90	93.11	97.76
	Precision	0.81	0.87	0.96
	Recall	0.9	0.93	0.98
	F Measure	0.85	0.91	0.97
	AUC	0.49	0.59	0.7
Random Forest	Accuracy	86.21	91.15	96.95
	Precision	0.86	0.89	0.96
	Recall	0.86	0.91	0.97
	F Measure	0.86	0.9	0.96
	AUC	0.54	0.68	0.8
Simple Cart	Accuracy	90	93.15	97.74
	Precision	0.81	0.89	0.96
	Recall	0.9	0.93	0.98
	F Measure	0.85	0.91	0.97
	AUC	0.41	0.5	0.62
BF Tree	Accuracy	90	92.99	97.76

	Precision	0.81	0.89	0.96
	Recall	0.9	0.93	0.98
	F Measure	0.85	0.91	0.97
	AUC	0.46	0.55	0.71
RBF Network	Accuracy	90	93.04	97.76
	Precision	0.81	0.87	0.96
	Recall	0.9	0.93	0.98
	F Measure	0.85	0.91	0.97
	AUC	0.48	0.63	0.74

In this case 1, In my analysis, I observed Bayesian based classifiers and the tree based classifiers consistently outperformed the other utilized fault prediction techniques in category-1. These classifiers exhibited superior performance across all five performance evaluation measures, underscoring their effectiveness in predicting software faults within this specific category of datasets.

Results Of Category -2

Technique	Metric	Min.	Average	Max.
Bayesian Logistic	Accuracy	64.41	84.96	89.8
	Precision	0.42	0.75	0.84
	Recall	0.64	0.84	0.9
	F Measure	0.51	0.78	0.85
	AUC	0.5	0.51	0.53
Naive Bayes	Accuracy	57.92	80.07	85.55
	Precision	0.69	0.81	0.88
	Recall	0.58	0.8	0.86
	F Measure	0.58	0.8	0.85
	AUC	0.57	0.71	0.82

Logistic Regression	Accuracy	76.36	85.43	89.8
	Precision	0.76	0.82	0.89
	Recall	0.76	0.85	0.9
	F Measure	0.76	0.82	0.9
	AUC	0.63	0.73	0.83
Multilayer Perceptron	Accuracy	78.44	84.71	89.61
	Precision	0.75	0.82	0.87
	Recall	0.78	0.85	0.9
	F Measure	0.76	0.83	0.87
	AUC	0.61	0.72	0.85
Simple Logistic	Accuracy	77.66	85.75	89.77
	Precision	0.76	0.82	0.89
	Recall	0.78	0.86	0.9
	F Measure	0.74	0.82	0.89
	AUC	0.55	0.73	0.83
SMO	Accuracy	57.92	80.07	85.55
	Precision	0.69	0.81	0.88
	Recall	0.58	0.8	0.86
	F Measure	0.58	0.8	0.85
	AUC	0.57	0.71	0.82
Voted Perceptron	Accuracy	66.23	83.32	89.17
	Precision	0.7	0.77	0.83
	Recall	0.66	0.83	0.89
	F Measure	0.55	0.78	0.85
	AUC	0.49	0.52	0.61
1B1	Accuracy	74.81	81.98	87.78
	Precision	0.74	0.82	0.87
	Recall	0.75	0.82	0.88

	F Measure	0.75	0.82	0.87
	AUC	0.55	0.64	0.79
IBK (k = 5)	Accuracy	77.51	85.46	89.64
	Precision	0.73	0.82	0.88
	Recall	0.78	0.85	0.9
	F Measure	0.74	0.83	0.88
	AUC	0.65	0.75	0.88
J48	Accuracy	76.37	85.19	90.01
	Precision	0.74	0.83	0.89
	Recall	0.76	0.85	0.9
	F Measure	0.75	0.83	0.89
	AUC	0.47	0.64	0.78
Decision Stump	Accuracy	67.27	85.02	90.34
	Precision	0.7	0.76	0.9
	Recall	0.67	0.85	0.9
	F-Measure	0.67	0.8	0.89
	AUC	0.55	0.65	0.76
Random Forest	Accuracy	77.39	83.92	89.63
	Precision	0.75	0.82	0.88
	Recall	0.77	0.84	0.89
	F Measure	0.76	0.83	0.88
	AUC	0.63	0.74	0.89
Simple Cart	Accuracy	78.7	85.93	90.34
	Precision	0.72	0.8	0.9
	Recall	0.79	0.86	0.9
	F Measure	0.73	0.82	0.89
	AUC	0.42	0.59	0.82
BF Tree	Accuracy	79.27	85.67	89.57

	Precision	0.71	0.82	0.88
	Recall	0.79	0.86	0.9
	F Measure	0.73	0.82	0.88
	AUC	0.41	0.62	0.81
RBF Network	Accuracy	73.76	85.11	89.43
	Precision	0.7	0.78	0.86
	Recall	0.74	0.85	0.89
	F Measure	0.74	0.8	0.86
	AUC	0.62	0.69	0.78

In the case of category-2 datasets, my findings indicated that tree-based classifiers outperformed the other utilized techniques across all five performance evaluation measures. These tree-based classifiers demonstrated superior performance and proved to be the most effective choice for predicting software faults within this particular category of datasets.

Results Of Category -3

Technique	Metric	Min.	Average	Max.
Bayesian Logistic	Accuracy	72.97	76.35	80.53
	Precision	0.64	0.72	0.78
	Recall	0.73	0.76	0.81
	F Measure	0.64	0.71	0.77
	AUC	0.5	0.57	0.65
Naive Bayes	Accuracy	72.52	78.61	84.44
	Precision	0.74	0.78	0.84
	Recall	0.73	0.79	0.84
	F Measure	0.73	0.78	0.83
	AUC	0.71	0.77	0.84

Logistic Regression	Accuracy	78.82	81.1	83.33
	Precision	0.77	0.8	0.83
	Recall	0.79	0.81	0.83
	F Measure	0.77	0.8	0.83
	AUC	0.72	0.78	0.82
Multilayer Perceptron	Accuracy	73.87	79.52	81.73
	Precision	0.74	0.79	0.81
	Recall	0.74	0.8	0.82
	F Measure	0.74	0.79	0.81
	AUC	0.74	0.76	0.79
Simple Logistic	Accuracy	77.92	81.52	83.7
	Precision	0.8	0.83	0.78
	Recall	0.82	0.84	0.76
	F Measure	0.76	0.8	0.83
	AUC	0.72	0.79	0.82
SMO	Accuracy	78.82	81.1	83.33
	Precision	0.77	0.8	0.83
	Recall	0.79	0.81	0.83
	F Measure	0.77	0.8	0.83
	AUC	0.72	0.78	0.82
Voted Perceptron	Accuracy	68.26	71.47	76.77
	Precision	0.64	0.7	0.77
	Recall	0.68	0.71	0.77
	F Measure	0.65	0.77	0.57
	AUC	0.63	0.7	0.69
IB1	Accuracy	72.97	76.17	79.41
	Precision	0.73	0.76	0.8
	Recall	0.73	0.76	0.79

	F Measure	0.73	0.76	0.8
	AUC	0.66	0.68	0.72
IBK (k = 5)	Accuracy	74.77	78.79	80.71
	Precision	0.72	0.77	0.79
	Recall	0.75	0.79	0.81
	F Measure	0.72	0.77	0.79
	AUC	0.73	0.77	0.81
J48	Accuracy	72.52	76.31	79.06
	Precision	0.72	0.75	0.78
	Recall	0.73	0.76	0.79
	F Measure	0.72	0.76	0.78
	AUC	0.62	0.66	0.68
Decision Stump	Accuracy	69.36	76.79	83.35
	Precision	0.67	0.76	0.83
	Recall	0.69	0.77	0.83
	F-Measure	0.68	0.76	0.83
	AUC	0.62	0.66	0.7
Random Forest	Accuracy	73.87	78.62	80.93
	Precision	0.74	0.78	0.8
	Recall	0.74	0.79	0.81
	F Measure	0.74	0.78	0.81
	AUC	0.76	0.79	0.81
Simple Cart	Accuracy	75.67	79.68	83.08
	Precision	0.74	0.78	0.83
	Recall	0.76	0.8	0.83
	F Measure	0.74	0.79	0.83
	AUC	0.65	0.68	0.7
BF Tree	Accuracy	73.87	78.88	82.14

	Precision	0.72	0.77	0.81
	Recall	0.74	0.79	0.82
	F Measure	0.73	0.78	0.82
	AUC	0.65	0.66	0.69
RBF Network	Accuracy	75.16	78.48	81.48
	Precision	0.73	0.77	0.81
	Recall	0.75	0.78	0.82
	F Measure	0.74	0.78	0.81
	AUC	0.63	0.72	0.76

In my analysis of category-3 datasets, I observed that the naive Bayes technique surpassed the other employed techniques in terms. This indicates that, this specific category of datasets, naive Bayes proved to be the most effective method for predicting software faults, consistently outperforming other techniques.

Results Of Category -4

Technique	Metric	Min.	Average	Max.
Bayesian Logistic	Accuracy	49.23	67.16	98.78
	Precision	0.32	0.59	0.98
	Recall	0.49	0.67	0.99
	F Measure	0.4	0.59	0.98
	AUC	0.5	0.54	0.69
Naive Bayes	Accuracy	49.06	64.94	83.93
	Precision	0.58	0.72	0.99
	Recall	0.49	0.65	0.84
	F Measure	0.47	0.64	0.9
	AUC	0.57	0.73	0.85
Logistic Regression	Accuracy	60.32	75.21	98.12
	Precision	0.6	0.75	0.99
	Recall	0.6	0.75	0.98
	F Measure	0.6	0.75	0.98
	AUC	0.62	0.75	0.92

Multilayer Perceptron	Accuracy	61.13	75.23	98.78
	Precision	0.62	0.75	0.98
	Recall	0.61	0.75	0.99
	F Measure	0.61	0.75	0.98
	AUC	0.66	0.76	0.9
Simple Logistic	Accuracy	62.39	76.37	98.67
	Precision	0.63	0.76	0.9
	Recall	0.73	0.76	0.9
SMO	Accuracy	59.4	73.76	98.78
	Precision	0.6	0.73	0.98
	Recall	0.59	0.73	0.99
	F Measure	0.54	0.72	0.98
	AUC	0.5	0.65	0.77
Voted Perceptron	Accuracy	48.97	64.25	98.56
	Precision	0.57	0.69	0.98
	Recall	0.49	0.64	0.99
	F Measure	0.46	0.6	0.98
	AUC	0.5	0.59	0.7
1B1	Accuracy	61.13	74.15	98.99
	Precision	0.61	0.74	0.99
	Recall	0.61	0.74	0.99
	F Measure	0.61	0.74	0.99
	AUC	0.6	0.7	0.87
IBK (k = 5)	Accuracy	59.1	74.87	99.39
	Precision	0.6	0.74	0.99
	Recall	0.59	0.75	0.99
	F Measure	0.59	0.74	0.99
	AUC	0.62	0.76	0.9
J48	Accuracy	61.53	75	99.11
	Precision	0.61	0.75	0.99
	Recall	0.62	0.75	0.99
	F Measure	0.61	0.75	0.99
	AUC	0.61	0.7	0.91
Decision Stump	Accuracy	55.58	72.56	98.78
	Precision	0.34	0.73	0.98
	Recall	0.56	0.73	0.99
	F-Measure	0.43	0.7	0.98
	AUC	0.66	0.86	
Random Forest	Accuracy	60.32	76.43	99.44
	Precision	0.59	0.76	1
	Recall	0.6	0.76	0.99
	F Measure	0.59	0.76	0.99

	AUC	0.65	0.79	0.93
Simple Cart	Accuracy	57.08	76.11	99.11
	Precision	0.56	0.75	0.99
	Recall	0.57	0.76	0.99
	F Measure	0.57	0.75	0.99
	AUC	0.42	0.7	0.88
BF Tree	Accuracy	59.1	75.62	99.11
	Precision	0.58	0.75	0.99
	Recall	0.59	0.76	0.99
	F Measure	0.58	0.75	0.99
	AUC	0.57	0.71	0.9
RBF Network	Accuracy	56.53	72.46	98.78
	Precision	0.57	0.72	0.98
	Recall	0.57	0.72	0.99
	F Measure	0.55	0.71	0.98
	AUC	0.54	0.71	0.82

In the case of category-4 datasets, my analysis revealed that the random forest technique delivered the best performance across all five performance evaluation measures, outperforming the other utilized techniques.

Overall in-depth analysis across categories, it became evident that Bayesian-based fault prediction techniques excelled when dealing with datasets containing a smaller portion of faulty modules, closely followed by tree based techniques.

Fault Prediction Models Evaluation

Evaluation of Techniques for the Prediction of Number of Faults

Here analysis consistently showing straightforward techniques / methods performed better. These findings are valuable not only for enterprise projects but also for software practitioners, helping

them choose the most suitable fault prediction technique for forecasting software defects at early stages.

In addition, I utilized three distinct fault datasets from to construct defect prediction models to evaluate in predicting the number of faults.

In this project, I focused on predicting the number of faults in a software module as the dependent variable.

Dataset	Total Number Of Modules	Non-Commented LOC	No. Of Faulty Modules	Distribution Of Fault in %
Eclipse-metrics- file-2.0.	6730	796KLOC	976	14.5
Eclipse-metric-file-2.1	7889	987KLOC	855	10.83
Eclipse-metric-file-3.0	10594	1305KLOC	1569	14.81

Modelling of Count Models

Probability distribution functions are pivotal in estimating expected values for count model, with maximum likelihood function playing a crucial role in estimating the parameters of these models. This aids in determining the most likely values for the model's parameters based on the observed data.

In this project, transformations have been applied to certain software metrics. Specifically, a square root transformation has been utilized for metrics with relatively larger values, while a logarithmic transformation has been implemented for the lines of code (LOC) metric. These transformations aim to render the data more suitable for modeling and analysis.

Using the STATA tool, a widely adopted statistical software package for data analysis and modeling. These models are instrumental in predicting and comprehending fault occurrences in software modules, offering valuable insights into the relationships between software metrics and fault proneness.

Modelling Of Linear Regression, Multilayer Perceptron, and DecisionTree Regression Techniques

I employed various methodologies within “Weka” ML tool. For the estimation of parameters, I utilized the least square method. The default values for the remaining parameters in Linear Regression were set according to the Weka tool specifications.

Results And Analysis 2 of 4

Results obtained from the practical execution of model analysis w.r.t. the AAE and ARE measures I have created and elaborated in the table below.

Analysis of Absolute Average Error (AAE) for Four Fault Prediction Techniques on the Utilized Datasets

AAE Analysis Of 4 Prediction Techniques				
Dataset	Linear1 Regression1	Multilayer1 Perceptron1	Decision Tree1 Regression1	Negative - Binomial1 Regression1
Eclipse-metric- file- 2.0	0.24	0.29	0.24	0.64
Eclipse-metric-file- 2.1	0.15	0.16	0.16	0.57
Eclipse-metric-file- 3.0	0.23	0.23	0.24	0.62

Analysis of Absolute Relative Error (ARE) for Four Fault Prediction Techniques on the Utilized Datasets

ARE Anslsysis of 4 Fault Prediction Techniques				
Dataset	Linear Regression	Multilayer Perceptron	Decision Tree Regression	Negative - Binomial Regression

Eclipse-metric- file- 2.0	0.14	0.14	0.13	0.53
Eclipse-metrics-file- 2.1.	0.08	0.07	0.1	0.52
Eclipse-metric-file- 3.0	0.13	0.11	0.19	0.52

If the values are lower in case of AAE and ARE it indicates performance is better for the used prediction techniques.

Specifically:

- i. In terms of AAE, demonstrating AAE values mostly below 0.50. However, -ve binomial regression showed relatively higher AAE values.
- ii. In the context of ARE, the multilayer perceptron demonstrated superior performance among the fault prediction techniques.
- iii. Negative binomial regression exhibited high values for the performance, suggesting lower performance compared to other techniques in this specific context.

Results And Analysis 3 of 4

Results obtained from the practical execution of model analysis created and elaborated in the table below.

The average absolute error (AAE) analysis was performed for four fault prediction techniques. Resulting AAE analysis provide insights into the accuracy of these techniques in predicting the number of faults. A lower AAE indicates better predictive accuracy.

AAE Analysis Of 4 Prediction Techniques

Dataset	Linear Regression2	Multilayer Perceptron2	Decision Tree Regression2	Negative - Binomial Regression2
Eclipse-metric- file- 2.0	0.24	0.29	0.24	0.64
Eclipse-metrics-file- 2.1.	0.15	0.16	0.16	0.57
Eclipse-metric-file- 3.0	0.23	0.23	0.24	0.62

The analysis of the average relative error (ARE) for four fault prediction techniques across all used datasets provides valuable insights into the performance of these techniques.

A lower ARE indicates better predictive accuracy and reliability in estimating the number of faults.

ARE Anslsys of 4 Fault Prediction Techniques				
Dataset	Linear Regression	Multilayer Perceptron	Decision Tree Regression	Negative - Binomial Regression
Eclipse-metric- file- 2.0	0.14	0.14	0.13	0.53
Eclipse-metrics-file- 2.1.	0.08	0.07	0.1	0.52
Eclipse-metric-file- 3.0	0.13	0.11	0.19	0.52

Lower AAE and ARE depicts better performance in FP techniques here.

To elaborate:

i. In the context of the analysis for the Average Absolute Error (AAE), it evidently shown superior performance. Many AAE values for these two techniques were below 0.50, indicating their effectiveness in predicting the number of faults. On the other hand, negative binomial regression

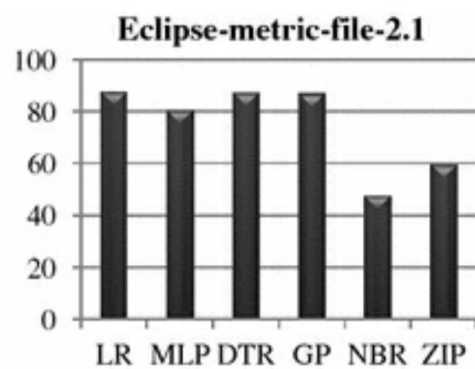
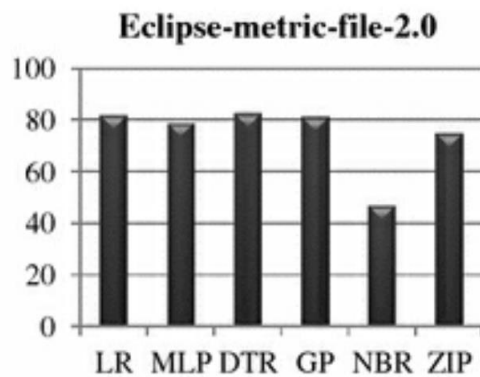
exhibited relatively higher AAE values, suggesting lower predictive performance for this specific technique.

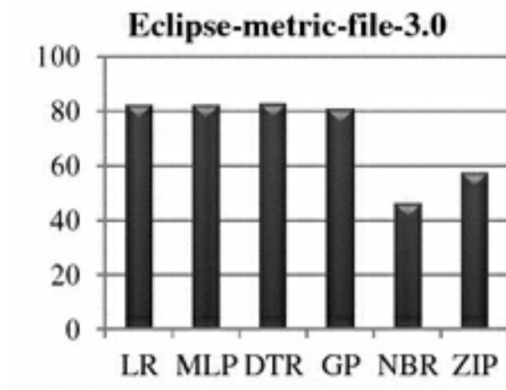
ii. Negative binomial regression consistently produced higher ARE values, highlighting its lower performance compared to the other techniques.

Results And Analysis 4 of 4

Visualization and Interpretation:

In the screenshots below, Results of `pred(0.3)` analysis are shown below.





The figures reveal that, in majority scenarios, the fault prediction has achieved a $\text{pred}(0.3) > 80\%$. However, there is an exception with negative binomial regression, $\text{pred}(0.3)$ value falls below 50% for the utilized fault datasets. This suggests that negative binomial regression may not perform as well as the other techniques in these specific cases.

12. Resource Requirements and Plan for their availability

As a MBA Student in MBA -Business Analytics, I am the resource for making this project under guidance from my supervisor and review from additional examiner.

I'm using my Windows 10 machine / laptop with .net framework, Python, Weka, STATA, Jupyter, and Visual Studio Code.

All the resources are installed.

13. Risks and Mitigation Plan

Data Quality and Availability:

- Risk: Insufficient or poor-quality data.
- Mitigation: Careful data selection, cleaning, and preprocessing.

Model Selection and Performance:

- Risk: Inappropriate model choice.
- Mitigation: Evaluate multiple models, cross-validation.

Overfitting:

- Risk: Overfitting on training data.
- Mitigation: Feature selection, regularization, and tuning.

I had used maximum likelihood estimation in count models/ regression models to estimate parameters. These transformations can be beneficial in addressing issues such as heteroscedasticity and skewed distributions (where metrics have larger values).

Addressing these risks will ensure a smoother execution of the project and successful achievement of its objectives.

14. Issues and Resolution

It is essential to iterate and refine the methodology based on the project's specific requirements and domain expertise. Additionally, considering domain knowledge and expert input in the process can lead to more effective defection prediction models tailored to the software development context.

Using Weka and STATA tool doesn't output direct results in tabular format as shown in the project work. So , in this case , lot of excel pivoting was done to show in a proper manner for lots of data from public datasets.

15. Conclusions and Recommendations

The implementation of a software defect prediction tool using code metrics can bring several significant benefits to the software development process and the overall quality of the software product. Some of the expected benefits include:

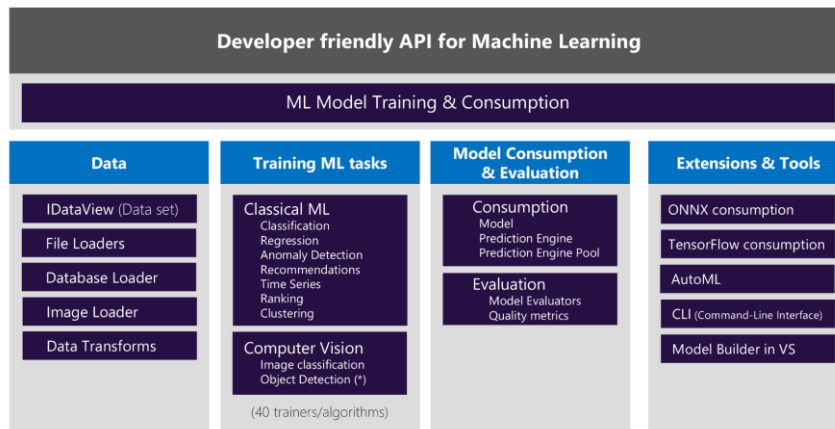
1. **Early Detection of Defects:** The defect prediction tool can identify potential defects early in the development lifecycle, allowing developers to address issues before they escalate. This early detection helps prevent defects from reaching production and reduces the cost and effort required for defect fixing.
2. **Improved Software Quality:** By proactively identifying defect-prone areas in the codebase, the tool enables software development teams to focus on high-risk components, resulting in improved overall software quality. Developers can concentrate their efforts on code areas , leading to a more reliable and robust software product.
3. **Cost Savings:** Detecting and fixing defects during the development phase is less costly than dealing with defects in production. The defect prediction tool helps reduce the cost of defect fixing, as issues are addressed early, minimizing the impact on the project budget.
4. **Enhanced Developer Productivity:** With the tool's guidance, developers can prioritize their code review and testing efforts more effectively. This leads to increased productivity as they can concentrate on the most critical areas of the codebase, rather than spending time on less defect-prone components.
5. **Reduced Software Maintenance Efforts:** Proactively addressing defect-prone areas helps reduce the need for extensive maintenance activities in the future. As the tool aids in producing higher-quality code, the software requires fewer maintenance efforts and results in a more sustainable codebase.
6. **Better Resource Allocation:** By predicting defect-prone areas, the development team can allocate their resources more efficiently. They can focus on code areas that require the most attention, leading to optimal resource utilization and project management.
7. **Increased Software Reliability:** The defect prediction tool contributes to the development of a more reliable software product. By targeting defect-prone regions, developers can significantly reduce the occurrence of defects in the final product.
8. **Data-Driven Decision Making:** The use of code metrics and machine learning in defect prediction

empowers software development teams with data-driven insights. It promotes a culture of evidence-based decision-making, where decisions are grounded in objective metrics and historical data.

9. **Improved Customer Satisfaction:** With a higher-quality software product, customers are more likely to have a positive experience with the software. Reduced defects and improved reliability lead to increased customer satisfaction and loyalty.
10. **Competitive Advantage:** Companies that invest in defect prediction tools and embrace proactive defect management gain a competitive advantage. They can deliver more stable and reliable software faster, giving them an edge in the market.

Actually, being a veteran .net developer, I had also mentioned below the ML.NET for SDP purpose.

ML.NET components



Pseudo code showing skeleton outline in building and evaluate models:

```
classifierType = {  
    Naive bayes ,  
    Random Forest ,  
    J48 ,  
    ...  
}  
10.times {  
    randomize instance order  
    prepare stratified 10-fold cross validation  
    fold.each{  
        get training set  
        get testing set  
        create feature selection set from training set  
        record selected features  
        classifierType.each{  
            build classifier  
            test classifier against testing set
```

```
record results
```

```
}
```

```
}
```

```
}
```

References

1. <https://dl.acm.org/doi/abs/10.1145/3555776.3577809>
2. Software Fault Prediction: A Road Map (SpringerBriefs in Computer Science) Kumar, Sandeep; Rathore, Santosh.
3. <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8058420>
4. https://www.researchgate.net/publication/323536716_Software_Bug_Prediction_using_Machine_Learning_Approach
5. Manjula.C.M. Prasad, Lilly Florence, and Arti Arya. 2015. A Study on Software Metrics based Software Defect Prediction using Data Mining and Machine Learning Techniques. International Journal of Database Theory and Application 7 (06 2015), 179--190.
6. Erik Arisholm, Lionel C. Briand, and Eivind B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software, 83(1):2–17, January 2010
7. G. Mauša, T. G. Grbac, and B. D. Bašić, “Software defect prediction with Bug-Code analyzer—A data collection tool demo,” in Proc. IEEE 22nd Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM), Sep. 2014, pp. 425–426.
8. R. Malhotra and R. Raje, “An empirical comparison of machine learning techniques for software defect prediction,” in Proc. 8th Int. Conf. Bioinspired Inf. Commun. Technol., 2014, pp. 320–327.
9. N.E. Fenton and Martin Neil. A critique of software defect prediction models. IEEE Transactions on Software Engineering, 25(5):675–689, 1999.
10. Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering 2008; 34(4):485–496.
11. Menzies, T., Stefano, J.S.D., Orrego, A., Chapman, R.: ‘Assessing predictors of software defects’. Proc. Workshop on Predictive Software Models, 2004
12. <https://www.mdpi.com/2071-1050/15/6/5517>
13. <https://www.sciencedirect.com/science/article/pii/S1319157818313077>
14. <https://core.ac.uk/download/pdf/185520882.pdf>

Glossary

Term	Abbreviation	Definition
Software Defect Prediction	SDP	The process of forecasting potential defects or bugs in software code using historical data, code metrics, and machine learning models.
Code Metrics	CM	Quantitative measurements or metrics that capture various aspects of software code, such as complexity, size, coupling, cohesion, and maintainability.
Machine Learning	ML	An artificial intelligence technique where computer systems learn patterns and make predictions from data without being explicitly programmed.
Training Data	-	The historical dataset used to train the machine learning models for defect prediction. It includes code metrics and corresponding defect labels.
Testing Data	-	A separate subset of the dataset used to evaluate the performance of the trained machine learning models.
Model Evaluation Metrics	-	Performance metrics to assess the accuracy and effectiveness of the defect prediction models, such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC).
Overfitting	-	A situation where a machine learning model performs well on the training data but poorly on unseen data due to excessive memorization of training examples.
Interpretability	-	The degree to which a machine learning model's prediction can be understood and explained in human-readable terms.
Cross-Validation	-	A resampling technique used to assess a model's performance by splitting the dataset into multiple subsets for training and testing, helping to avoid overfitting.
Data Preprocessing	-	The process of cleaning, transforming, and preparing the dataset for analysis, including handling missing values and outliers.

ROC Curve	-	Receiver Operating Characteristic curve, a graphical representation of the true positive rate against the false positive rate, used to evaluate the performance of binary classifiers.
Hyperparameter Tuning	-	The process of selecting the optimal hyperparameters of machine learning models to achieve the best performance.
Ethical Considerations	-	The awareness and adherence to ethical guidelines, privacy regulations, and data protection when handling sensitive software development data.
Model Deployment	-	The implementation of the trained defect prediction model in a production environment to predict defects in new code changes.
Scope Creep	-	The uncontrolled expansion of the project scope beyond its original objectives, potentially leading to resource and timeline overruns.
Continuous Monitoring	-	The practice of regularly assessing the performance of the defect prediction model and updating it with new data to maintain accuracy.
Proactive Defect Management	-	The approach of identifying and addressing defects early in the software development lifecycle to prevent them from reaching production.
Software Quality	-	The degree to which software meets specified requirements and user expectations, including reliability, maintainability, and usability.
Data Privacy	-	The protection and confidentiality of personal or sensitive data collected and used during the project, ensuring compliance with privacy regulations.
Machine Learning Model	-	A mathematical representation that learns patterns from data and makes predictions based on new inputs. In the context of

		this project, it predicts software defects based on code metrics.
Feature Engineering	-	The process of creating new informative features or transforming existing features to improve the performance of machine learning models.
Local Interpretable Model-agnostic Explanations	LIME	A method for explaining individual predictions of machine learning models in a human-interpretable way.
Software Development Repository	-	A version control system or database used to store and manage software code and its history during the development process.

Summary of how the feedback for Project Outline and Mid-Semester Project Report have been addressed

As per feedback from mentor in Project Outline evaluation, I had elaborated Data Methodologies and also shown the Secondary data i.e. datasets from public repositories. I had depicted the pre-processing techniques and analysis done. I had updated Mid Project Report and carried in Final

Report as per the feedback after evaluation from my mentor. Below is the evaluation screenshot from Bits portal at various stages.

Dissertation Evaluation Progress and Status									
Submission	Soft Submission Date	Hard Submission Date	Submission Done	Resubmission Required	Resubmission Date	Resubmission Done	Evaluation Status	Grade	Evaluator
FINAL REPORT	10/11/2023	17/11/2023	Yes	No	-	-	Not Started	-	Rajesh Prabhakar Kaila . (rajesh.prabhakar@wilp.bits-pilani.ac.in)
MID SEMESTER REPORT	20/09/2023	29/09/2023	Yes	No	-	-	Completed	GOOD	Rajesh Prabhakar Kaila . (rajesh.prabhakar@wilp.bits-pilani.ac.in)
ABSTRACT	26/07/2023	15/08/2023	Yes	No	-	-	Completed	GOOD	Rajesh Prabhakar Kaila . (rajesh.prabhakar@wilp.bits-pilani.ac.in)

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
Work Integrated Learning Programmes Division (WILPD)

Particulars of Student

Student Id	2021mb21621
Student Name	Harsimrat Singh
Student E-Mail Address	2021mb21621@wilp.bits-pilani.ac.in
Employing Organization and Location	ASM Technologies Limited, Bangalore
Programme Name	MBA in Business Analytics
Semester	4 th
Project Title	Software Defection Prediction using code metrics

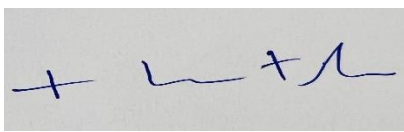
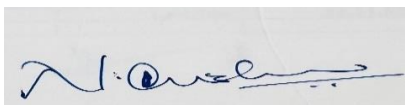
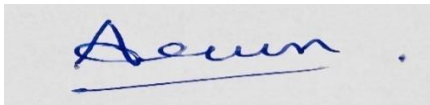
Particulars of the Supervisor and Additional Examiner

	Supervisor	Additional Examiner
Name	Krishnan Narayana	Arunachalam Palaniappan
Qualification	B.E. in Industrial & Production Engg. , MBA- Marketing ,	MCA

	currently a research scholar pursuing PhD in Management.	
Designation	Senior Vice President	Sr. Engineering Manager
Employing Organization and Location	ASM Technologies Limited, Bangalore	ASM Technologies Limited, Bangalore
Phone No (with STD code)	+91 80 669 62301	+917406273399
Email Address	krishnan@asmltd.com	arunachalam@asmltd.com

Remarks of the Supervisor on Project Outline

Go ahead, and keep up the good work.

		
Signature of Student	Signature of Supervisor	Signature of Additional Examiner
Name: Harsimrat Singh	Name: Krishnan Narayana	Name: Arunachalam Palaniappan

Please write to **WILP Project** at project@wilp.bits-pilani.ac.in for any queries / clarifications

