**Software Defect /Fault Prediction using code metrics**

**Course No.: MBAZG622T**

**Course Title: Project Work**

**Mid-Semester 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.**

**(September-2023)**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Section No.** | **Section Name** | **Page No.** |
| 1 | Requirement Statement / Problem Statement | 3 |
| 2 | Project Objectives | 5 |
| 3 | Project Scope and Limitations | 6 |
| 4 | Software Metrics | 8 |
| 5 | Methodology | 14 |
| 6 | Data Preprocessing and Feature Selection | 16 |
| 7 | Architecture Of Software Fault/Defect Prediction | 19 |
| 8 | Components of Software Fault/Defect Prediction | 20 |
| 9 | Model Selection, Development and Performance Metrics | 23 |
| 10 | Model Development, Model Training, and Evaluation (Cross-Validations) | 30 |
| 12 | Results, Visualizations and Analysis | 42 |
| 13 | Resource Requirements and Plan for their availability | 61 |
| 14 | Risks and Mitigation Plan | 62 |
| 15 | Issues and Resolutions | 63 |
| 16 | Conclusions and Recommendations | 64 |
| Annexure-1 | Computer Programs / Code | 66 |
|  | References | 68 |
|  | Glossary | 69 |

# Project Type:

# [ ] Work-related

# [√] Industry-related

# [ ] Academic

# [ ] Others If Others, please specify the other project type: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Project Area:

# *Predictive Analytics*

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Requirement Statement / Problem Statement

The process of predicting parts that are fault prone in software is called SDP.

The idea behind SDP is to use measurements extracted from e.g., the source code and the development process, among others, to find out if these measurements can provide information about defects. When I had been working as software developer had experienced and studied in may forums and where it is prevalent that testing related activities consume between 50% to 80% of the total development time. As this is a substantial part of the development process, it is important to focus on testing those parts where defects are likely to occur due to its cost saving potential. SDP when I studied while making this project learnt that where simple equations, with measurements from source code as variables, was used as prediction techniques. Since then, the focus for prediction techniques has shifted towards statistical analysis, expert estimations and ML.

Taking about ML, ML is a branch of artificial intelligence concerning computer programs learning from data. ML aims at imitating the human learning process with computers, and is basically about observing a phenomenon and generalizing from the observations.

ML has broadly two categories: supervised and unsupervised learning.   
There is a large variety of ML algorithms (also known as classifiers) for supervised learning including decision trees, classification rules, neural networks and probabilistic classifiers.

In the dynamic landscape of software development, defects or bugs in software applications can be costly and disruptive. Early detection and prevention of defects are essential to ensure high software quality and customer satisfaction. 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.

# 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.

# 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.

The development of a tool for data analysis, as this is done with the existing  
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.

# 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:

1. Product Metrics
2. Process Metrics.

These categories are further classified into subcategories. However, sometimes, each

category is not mutually exclusive and there are some metrics, which act to be a part of more than one category.

**Product metrics:** This is the set of metrics calculated from the end software product. These metrics are generally used to estimate the overall quality of the software such that whether software product confirms norms of ISO-9126 standard or not, etc. Product metrics are further

classified as traditional metrics, object-oriented metrics, and dynamic metrics.

**Traditional metrics / Static Code Metrics:** are metrics that can be directly extracted from the source code, such as *Lines of code (LOC)* and cyclomatic complexity. This set of metrics mainly contained the following metrics:

Source Lines Of Code (SLOC) are a family of metrics centred around the count of lines in source files. The SLOC family contains, among others: Physical LOC (SLOCP), the total line count, Blank LOC (BLOC), empty lines, Comment LOC (CLOC), lines with comments, Logical LOC (SLOC-L) and lines with executing statements.

Size metrics: “Function Points (FP), Source lines of code (SLOC), Kilo-SLOC (KSLOC).”

Quality metrics: “Defects per FP after delivery, Defects per SLOC (KSLOC) after delivery.”

System complexity metrics: “Cyclomatic Complexity, McCabe Complexity, and Structural complexity”.

The Cyclomatic Complexity Number (CCN), also known as McCabe metric, is a measure of the complexity of a module’s decision structure, introduced by Thomas McCabe.

CCN is equal to the number of linearly independent paths and is calculated as follows: starting from zero, the CCN is incremented by one whenever the control flow of a method splits, i.e. when if, for, while, case, catch, &&, || or ? is encountered in the source code.  
Other static code metrics include compiler instruction counts and data declaration counts.

Halstead metrics: “number of distinct operators (nl), number of distinct operands (n2), total number of operators (N1), total number of operands (N2), Program vocabulary (), volume (v), program length (N), Difficulty (D), effort (E), number of delivered bugs (B), time required to program (T)” (Halstead 1977).

**Object-oriented metrics:** This set of metrics are corresponding to the various features of the software developed using object-oriented (OO) methodology such as coupling, cohesion, inheritance. Many OO metrics suites have been proposed capturing different OO features.

A subcategory of static code metrics are object-oriented metrics, since they are also  
metrics derived from the source code itself. The *Chidamber-Kemerer (CK) Object-oriented (OO) Metric Suite* used in my project consists of eight different  
metrics: six from the original CK metric set and two additional metrics, which follows  
below.

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)”.

Coupling Between Objects (CBO) counts the number of other classes to which the  
class is coupled, i.e. the usage of methods declared in other classes

Weighted Methods per Class (WMC) counts the number of methods in a class.  
Depth of Inheritance Tree (DIT) counts length of the maximum path from class to root class.

Number of Children (NOC) counts the number of subclasses for one class.

Lack of Cohesion in Methods (LCOM) counts the number of methods in a class that does not share usage of some member variables.  
The additional OO metrics that are used in the thesis are:  
Afferent couplings (Ca) counts how many other classes that use the class (cf. CBO).  
Number of Public Methods (NPM) counts how many methods in the class that is declared as "public".

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”.

Briand metrics suite: “IFCAIC, ACAIC, OCAIC, FCAEC, DCAEC, OCAEC, IFCMIC, ACMIC, OCMIC, FCMEC, DCMEC, OCMEC, IFMMIC, AMMIC, OM-MIC, FMMEC, DMMEC, OMMEC

**Dynamic metrics:** Dynamic metrics are used to capture the dynamic behaviour of the software project. This set of metrics are calculated from a running program and are used to identify the objects that are the most run-time coupled and complex during execution. These metrics give different indication on the quality of the software design.

Yacoub metrics suite: “Export Object Coupling (EOC) and Import Object Coupling (I0C)”.

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 matrices, the division of the workload of different components3 may be decided in the design phase of software development. This division of tasks can impact the quality of the software. Dependency mapping between source code files could therefore be a possible  
predictor of fault prone components to detect fault prone source files and packages.

In this project , efforts are done to collect for each source code file or package, the import statements and compared these imports to failures in components to perform dependency mapping.

**Process metrics :** Process metrics capture and measure the features of software development life cycle. These metrics are used to estimate the process of software development.

Further, these metrics can be used to make strategic decisions about the software development process.

Generally, software managers and developers have used these metrics to measure software process and services that lead to long-term software process improvement. Process metrics can be further classified into different subcategories as follows:

Code delta metrics: “Delta of LOC, Delta of changes”.

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

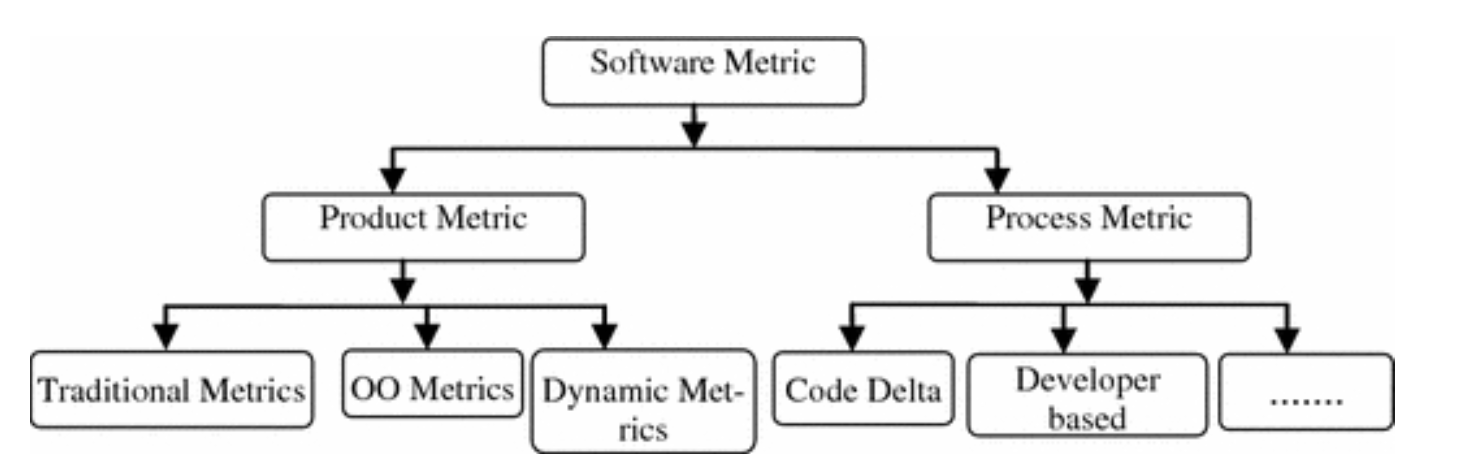
Change metrics: “Revisions, Refactoring’s, Bugfixes, Authors, LOC added, Max LOC Added, Ave LOC Added, LOC Deleted, Max LOC Deleted, Ave LOC Deleted, Codechurn, Max Codechurn, Ave Codechurn, Max Changeset, Ave Changeset and Age”.

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**Test code metrics can be made up of the same set of metrics that has been previously  
described, but for the source code of the tests. In addition to this, Nagappan [36]  
introduced a test metric suite called Software Testing and Reliability Early Warning  
metric suite (STREW) for finding software defects. STREW is constituted by nine  
metrics, in three different families.

STREW provides an estimate of the software’s quality in an early stage of the development as well as identifying fault prone module.

# Methodology

**Software Fault Dataset**

Typically, the idea of software fault prediction is based on this underlying process. It makes the use of historic fault dataset of the earlier software projects stored in the repositories to predict fault-proneness in the currently developing software project.

Software fault dataset is used as training dataset to train the fault prediction model and as testing dataset to assess the performance of fault prediction model.

Mainly, software fault dataset has three sets of information:

* Set of software metrics,
* Fault information such as faulty or non-faulty software module,
* Number of faults per module, and
* Meta information about the software project.

**Software fault information:** This information tells about the fault-proneness of the given software artifact or component. Depending upon the software development methodology, component can be a class, a function, a module, or a package. When, during the development or

testing, any fault is detected in any software component that have faults reported in the software repository along with the details of the component where it has found. The fault information can tell that whether a given component is faulty or non-faulty, how many faults are found in the component, what is the severity of the faults, etc.

Generally depending on the availability of the repository and the type of software project for which data is collected, fault data repositories can be classified in to 3 types i.e. (1) Private /Commercial Datasets repository, (2) Partially Public / Freeware Dataset repository, and (3) Public Datasets Repository.

Here in this project, I am using public fault/defect data repository. So, in this case software metric values and corresponding fault information both are publicly available.

In this project I am using PROMISE data repository.

<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 in these repositories are generally corresponding to open-source software projects. The studies performed using datasets from these repositories can be repeatable.

Not all the available fault dataset contained all types of fault information. Some of the fault datasets have information of only faulty and non-faulty software modules. Some other fault datasets have both number of faults and severity of fault information for software modules.

# Data Preprocessing and Feature Selection:

The datasets available in public repositories like PROMISE and ECLIPSE that I used in this project are of high quality and are from any problem. However, this is not always held for all the available datasets.

There are some quality issues associated with the datasets available in the public domain repositories that need proper handling / cleaning before using them for building software fault prediction model.

**Outlier:** In fault dataset, outliers are the observations that significantly deviated from the general observations of the dataset. Outliers violate the mechanism that is used to generate the data points. Careful detection and removal of outliers are particularly important in the fault prediction since there is a gray area between the outlier objects

and normal objects.

Additionally, sometimes outlier observations contained fault information and removing them can cause significant loss of fault information.

**Missing value:** Missing values are the set of values left blank in the dataset. Due to the human error who entered the values or unavailability of the information, sometimes some values are not entered in the dataset. Some of the fault prediction models ignore the observations corresponding to the missing values and learn from the rest of the observation, while some other fault prediction models employ some autocorrective measures to deal with the missing value.

**Repeated value:** Repeated attributes represent a situation in the dataset where two attributes/features have the same identical values for each observation. This particular case occurs in the dataset when a single attribute is over-described. For building accurate fault prediction model, it is required to remove one of such attribute, so that each attribute is being represented only once.

**Redundant and irrelevant value:** Data redundancy is a condition is the dataset where same attribute/feature is held in two or more observations with the same class label. Such data points are problematic for the effective learning of fault prediction since this could lead to the overfitting of the prediction model and could result in the

model performance decrement.

**Class imbalance:** Class imbalance occurs when the number of positive class observations (minor class) is less than the number of negative class observations(major class).

It represents a situation where a class of observations is rarely presented in the dataset compared to the other types of observations. In this case, observations of major class dominate the dataset as opposed to the observations of the minor class. This imbalance in the distribution of observations can lead to the biased learning of prediction model toward the observations of major class. The prediction model can produce poor results for the minor class observations.

**Data shifting problem:** In fault dataset, data shifting problem occurs when the training

dataset and testing dataset are not following the same joint distribution. In the literature, data shifting also refers to concept shift or concept drift. When data shifting happens, the prediction model having previous knowledge about the dataset is in no use as the new dataset has the change in the class distribution. Data shifting problem can have a serious effect on the performance of the prediction models.Therefore, it is required to handle the data shifting problem for building accurate and effective fault prediction model.

**High data dimensionality:** Dimensionality refers to the number of attributes/features in the given fault dataset. When the number of attributes is higher, then it is known as high data dimensionality. Sometimes high data dimensionality is defined as the situation where number of attributes (P) is greater than the number of observations (m) in the dataset (p > m). Earlier, it was found that high data dimensionality could decrease the performance a fault prediction model and could lead to the higher misclassification errors. Higher data dimensionality is also resulted in the higher computational cost and memory usage when building the fault prediction model.

**Feature (or attribute) selection**, is a method for handling large metric sets, to identify  
which metrics contribute to the SDP performance. By using feature selection, redundant, and hence non-independent, attributes are removed from the data set.

There are two approaches to feature selection: wrappers and filters. Wrappers use  
the ML algorithm itself to select attributes to evaluate the usefulness of different features. Filters use heuristics based on the characteristics of the data to evaluate the  
features. The positive effects of using filters over wrappers is that they operate faster, and are hence more appropriate for the selection in large feature sets.

Another positive effect of using filters is that they can be used together with any ML algorithm. However, in most cases they require the classification problem to be discrete,  
whereas wrappers can be put into use with any classification problem. As wrappers  
uses the same algorithm for feature selection and classification, the feature selection  
process must be done for every algorithm used for prediction.

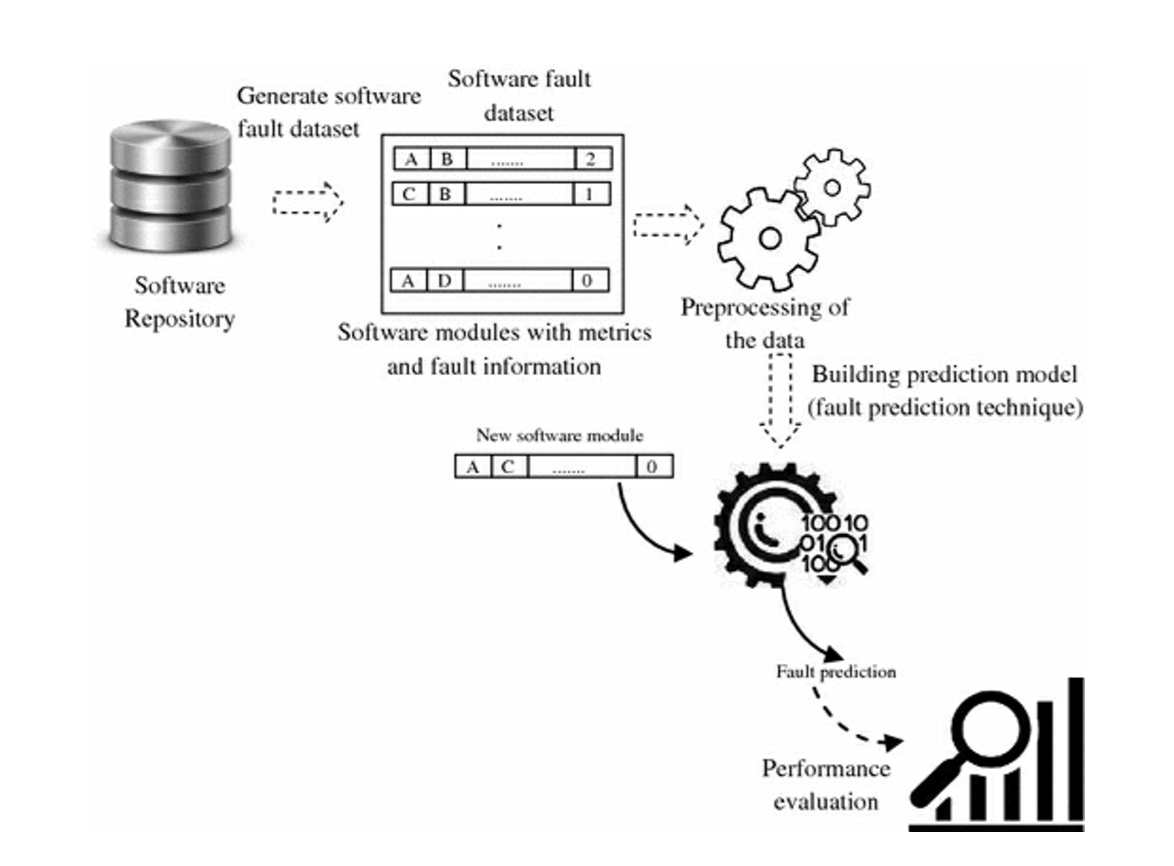
A filter-based feature selection approach called Correlation based Feature Selection (CFS).

Unlike other filter-based feature selection algorithms that evaluate individual features, CFS  
evaluates subsets of features by taking both the merit of individual features and their  
intercorrelations into account. The merit of a feature subset is calculated using Pearson’s correlation, by dividing how predictive a feature group is with the redundancy between the features in the subset. If a feature is highly correlated with one or more of the other features in the feature set, that feature will not be considered to be included in the reduced feature set.

# Architecture Of Software Fault/Defect Prediction

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, here are some details on components of software defect/fault prediction.

Main components of software defect prediction are software defect/fault dataset (metrics and fault information) , defect / fault prediction techniques , and performance evaluation measures.

# Components of Software Fault/Defect Prediction

Meta information about the project: Meta information refers to basic information about software project, which can provide the additional information and can make working with data easier. It may have the following set of information such as the domain for which software is

developed, the programming language in which software is written, the number of releases/versions of the software.The transferability of fault prediction model between different contexts may affect the prediction results. Meta information of software project consists the following variables/factors that can be used when building fault prediction models, as given below

Source of data: Source of the data shows the domain of the software project which is used. For example, a software project can be of open-source or commercial, can be Web browser or IDE, etc. Sometimes, it also shows that whether fault dataset of the project is publicly available or not. The source of the fault dataset can have an effect on the performance

of the fault prediction models. Fault prediction models are suffered from the data shifting issue, where training and testing have been performed on different datasets.

Maturity of the system: It shows how old software system is. The maturity of system is measured in terms of number of versions/releases over which software system evolved. Each new release of the software is generally corresponding to the changes occurred in the software or to add new functionality in the software. The mature the system is, the lower the chance of error in the system. Performance of fault prediction model can get influenced by the maturity of the system.

Size: Size of the software project is estimated using non-commented lines of code (LOC) attribute. Generally, it is represented in kilo lines of code (KLOC). It is generally analogy that the higher the value of LOC, the higher the faults in the software. Size of the software

influences the performance of fault prediction model. Fault prediction model based on large software project generally produced better performance.

Application domain: It presents the environment and the process under which a software project is developed. Therefore, sometimes a fault prediction model built for one application domain may not perform better when application domain changes.

The granularity of prediction: The granularity of a fault dataset shows the level for which data has been collected. For example, fault dataset collected corresponding to the class level for object-oriented software or function level for procedural software, or at the module level, etc.

Binary Class Classification of Software Faults & Confusion Matrix importance:

Fault prediction models based on the binary class classification classify the given software modules into faulty or non-faulty classes only. When a fault prediction model predicts more than one fault in the software module then that module is marked as faulty. When a fault prediction model predicts zero faults in the software module then that module is marked as non- faulty.

These studies have used various machine learning and statistical techniques including logistic regression, linear regression, naive Bayes, decision tree to build the fault prediction models for this context. Some semi-supervised learning such as EM algorithm and unsupervised learning techniques such as clustering has also been used to build the fault prediction models for the binary class classification of faults.

In this type of fault prediction, confusion matrix-based performance evaluation measures have been used to assess the performance of the built fault prediction models. Accuracy, precision, recall, and f-measure have primarily used to assess the performance of the prediction models. Some of the studies have also used AUC (area under ROC curve) and g-means to assess the performance of the prediction models.

Additionally, class level software metrics can help in building better fault prediction models as compared to method level software metrics. Eleven different techniques such as linear regression, pace regression, support vector regression, neural network, naive Bayes, instance-based learning, etc. has been evaluated. From the results found that combination of 1-Rule and instance-based learning techniques gives better prediction accuracy compared to the other used techniques.

**Performance evaluation measures:** Performance evaluation measures are used to assess the performance of a fault prediction model. A fault prediction model is trained for some fault dataset and then this trained model is applied to unknown testing dataset to assess its performance.

Various performance evaluation measures have been used earlier in the context of software fault prediction to assess the model performance. Generally, these performance measures can be grouped into two classes:

numeric measures and

graphical measures.

Numeric performance evaluation measures belong to the class of measures provide a numeric description about the performance of prediction models. It mainly contains accuracy, precision, recall, f-measure, false positive rate, G-means, false negative rate, J-coefficient, specificity,

average absolute error, and average relative error.

Graphical performance evaluation measures display information in easy and quickly understandable manner.

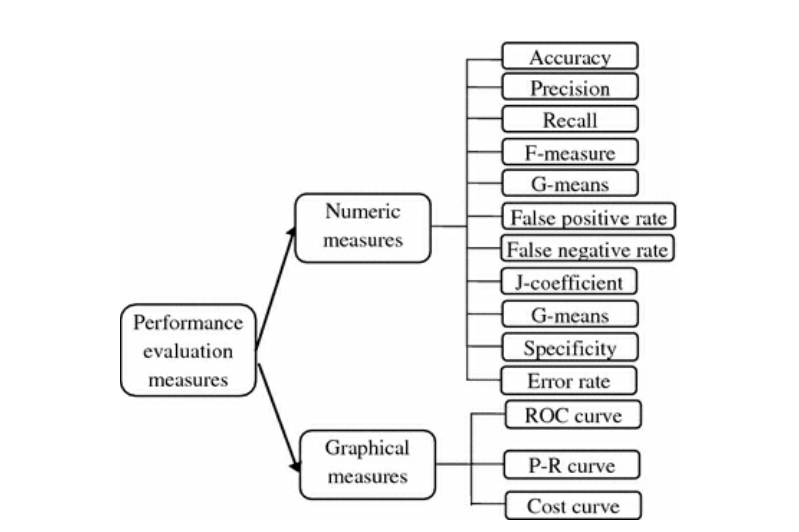
It mainly contains ROC curve, Precision-Recall (P-R) curve, and Cost curve.

# Model Selection, Development and Performance Metrics

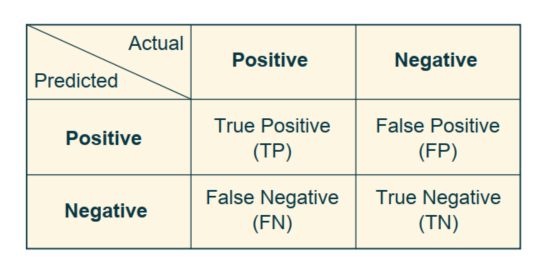
This project was modelled to test the discriminatory power of different metric sets, the ML algorithms included were used with their standard settings in the WEKA environment. The algorithms used to build the prediction models in this bug prediction project.

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 root of all the numeric evaluation measures is the **confusion matrix**. A confusion matrix contains information about actual and predicted classifications done by a fault prediction technique. It provides four different information.



The confusion matrix is a fundamental tool for evaluating and fine-tuning software defect prediction models. It helps you 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.

1. True positive (TP) = a faulty module is correctly predicted as faulty.

2. True negative (TN) = a non-faulty module is correctly predicted as non-faulty.

3. False positive (FP) = a non-faulty module is incorrectly predicted as faulty.

4. False negative (FN) = a faulty module is incorrectly predicted as non-faulty.

**Accuracy:** It shows the probability of correctly predicted faulty and non-faulty modules for a fault prediction model. However, accuracy does not provide any information about the incorrect prediction of faulty and non-faulty modules. It does not capture the misclassification cost of the prediction model. Therefore, accuracy is not a suitable measure, if anyone is interested in estimating misclassification cost of faulty and non-faulty modules or if fault dataset is imbalance.

Accuracy = (TN+TP) / (TP+TN+FP+FN)

False positive rate (FPR) and False negative rate (FNR):

FPR is calculated as the ratio of non-faulty modules

predicted incorrectly as faulty module to the all non-

faulty modules in the dataset. It

is also known as false alarm rate or type 1 error.

FPR = FP / (TN+FP)

FNR is calculated as the ratio of faulty modules

predicted incorrectly as non-faulty module to the all

faulty modules in the dataset (Lewis and Gale 1994). It is

also known as type 2 error.

FNR = FN/ (FN+TP)

Precision, Recall, and Specificity:

Precision is used to measure the relevancy of the results.

It shows the amount of modules predicted correctly as

faulty out of all faulty predicted modules.

Precision = TP/( FP+TP)

Recall is used to measure how many are true out of

the relevant results. It shows the amount of modules

predicted correctly as faulty out of all the faulty modules

in the given fault dataset (Conte et al. 1986). It is also

known as probability of detection (PD).

Recall = TP/ (FN+TP)

Precision and recall are useful when dataset is

imbalance and accuracy measure cannot be used. Using

them together can provide useful information about the

fault prediction model.

Specificity is calculated as the portion of negative (non-

faulty modules) that bare correctly predicted by the fault prediction model.

Specificity = TN/ (FP+TN)

F-measure: It is used to show the trade-off between

precision and recall. It is defined as the harmonic mean

of precision and recall values of a fault prediction model.

F-Measure= 2\* (Precision \* Recall) / Precision + Recall

G-means & J-coefficient:

In fault prediction environment, sometimes a prediction

model is achieving higher prediction accuracy by

correctly predicting the non-faulty modules. However, the

same prediction model is incorrectly predicting the faulty modules.

In this scenario, G-means and J-coefficient can be used to capture the actual prediction performance of the model.

G-Means 1 =

G-mean-1 is calculated as the square root of the precision and recall. G-mean-2 is calculated as the square root of the product of recall and specificity

J-Coeff =

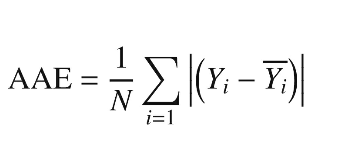
J-coefficient (J-coeff): It provides the information about the fault prediction model performance by combining the results of recall and specificity.

J-Coeff = Recall +Specificity -1

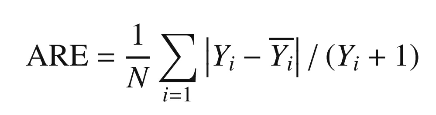
The value of J-coeff = 0 indicates that the chance of identifying a faulty is equal to the false alarm rate and model is not useful to do fault prediction. The value of J-coeff  > 0 indicates that model is useful to do fault prediction . The J-coeff = 1 shows perfect fault prediction model, and the J-coeff = −1 shows the worst fault prediction model.

**Error rate:** It is used to calculate the difference between the predicted values of the faults to the actual value of the faults in a given software module. Generally, two different measures are used to calculate error rate.

**Average absolute error (AAE):** AAE calculates the absolute difference between the predicted values of the faults to the actual value of the faults in a given module.

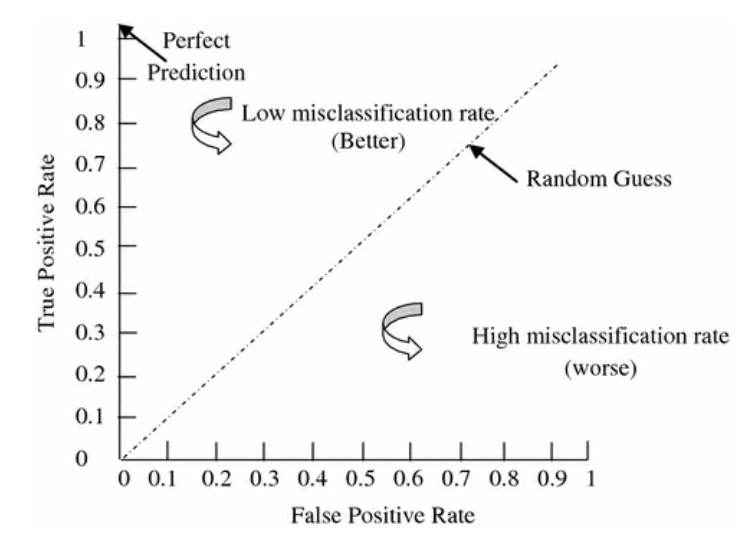


**Absolute relative error (ARE):** ARE estimates the magnitude of the absolute error in comparison with the total size of the object measured.



**Graphical Measures:** Graphical measures are used to visually depict the performance of a fault prediction model. These measures are also derived from the confusion matrix.

**ROC Curve:**



ROC curve provides a performance estimation by calculating the ratio of the faulty modules predicted correctly to non-faulty modules predicted incorrectly. It gives an estimation about

the prediction model’s performance by considering the cost of misclassification of faulty and non-faulty modules, if there is an unbalance in the class distribution. ROC curve is represented by a graph with a diagonal line.

This diagonal line shows a random model, which has no prediction capability.

ROC curve starts from the bottom left side and evolves toward the upper right side. The closer the curve to the upper left side, the accurate the prediction model is.

Generally, area under the ROC curve (AUC) is used to measure the prediction capability of the model. It is generally calculated by choosing a threshold value. The value AUC = 1 represents 100% prediction capability of the model. The value AUC = 0 represents the 0% prediction capability of the model.

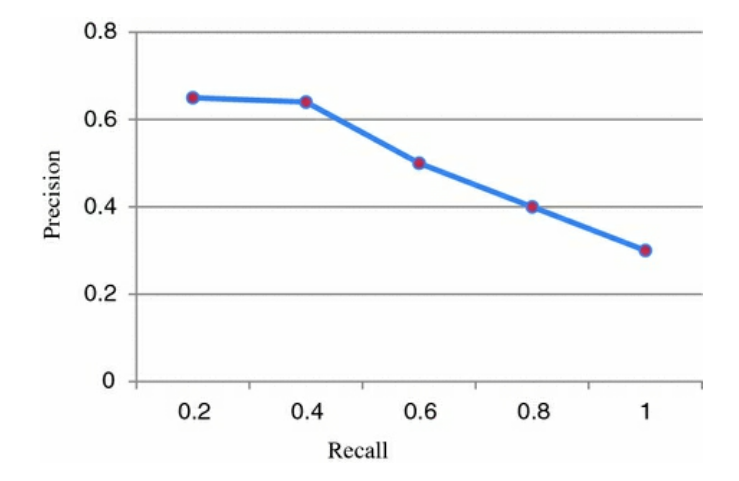
**PR Curve:**

A PR curve shows the trade-off between precision and recall. It is plotted by showing recall value at x-axis and precision value at y-axis.

In PR curve, high value of recall and high value of precision show the best performance.

For a prediction model to be perfect, its PR curve must pass through the upper right corner having 100% precision and 100% recall.

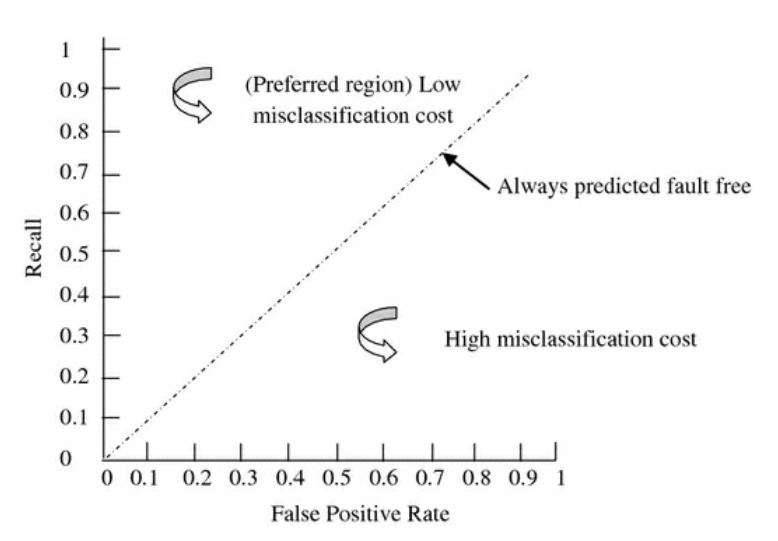
The closer the PR curve to the upper right corner, the better the prediction model is.



**Cost Curve:**

A cost curve is used to depict 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. The horizontal line between (0,0) and (0,1) shows that prediction model predicted correctly all the modules. Generally, range of (0,0.5) is sufficient for model evaluation.



# Model Development, Model Training, and Evaluation (Cross-Validations)

**Fault Dataset Repositories**

PROMISE (Predictor Models In Software Engineering,

http://openscience.us/repo/): PROMISE software dataset repository is one of the most widely used repositories in the software fault prediction community. It contains software fault and other datasets such as effort estimation, refactoring, etc. of various open-source and proprietary software systems. The repository hosts datasets of a large number of applications and software systems such as Apache software project, including jEdit, Lucene, Xerces, and many others, Eclipse , Mozilla.

Currently, PROMISE data repository contained fault dataset of 61 different software projects and their different releases. The size of the repository is growing rapidly with many more software datasets are added day

by day.

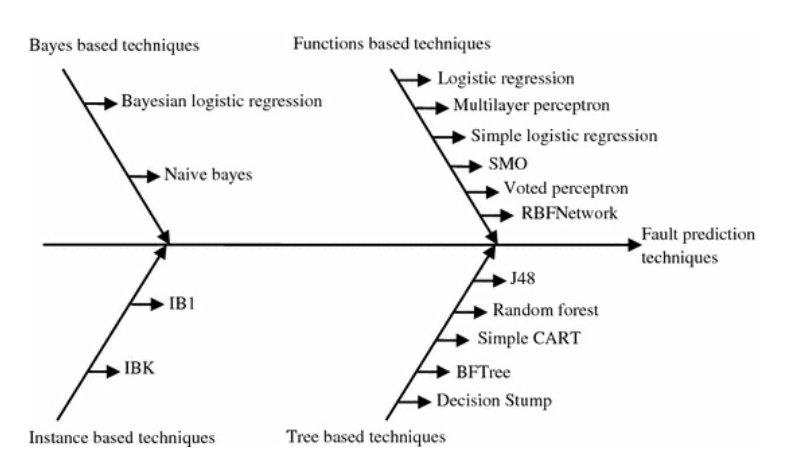
Eclipse Bug Data Repository (https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/): Eclipse bug data repository has been primarily design to collect and store fault dataset of Eclipse project. This repository contained the fault datasets of Eclipse project and its components.

Currently, repository hosts the dataset of three versions of Eclipse project, Eclipse-2.0, Eclipse-2.1, and Eclipse-3.0.

For each version of Eclipse, its provide the information of various source code and structural metrics and pre-release and post-release faults.

**Fault Prediction Techniques**

It includes several machine learning techniques such as Bayesian-based techniques, tree-based techniques, instance-based techniques and several statistical techniques such as logistic regression, linear regression.



The used fault prediction techniques have been classified into four major categories: Bayesian-based techniques, tree-based techniques, instance-based techniques, and function-based techniques.

Each category consists of various fault prediction techniques. All the used fault prediction techniques are supervised learning techniques. A technique is called supervised technique when the technique operates under the supervision provided with the actual outcome for each of the training examples. The supervised technique requires known fault measurement data (i.e., the number of faults, fault density, or faulty or non-faulty) for training data.

Usually, fault measurement data from previous versions, pre-release, or similar project can act as training data to predict new projects.

In tree-based techniques, a tree type (hierarchical) of structure is created that classifies the given testing example into one of the output classes (faulty or non-faulty) by deducing some learning rules from the given features (software metrics).

The root node is selected by determining a feature that best splits the training data. Similarly, other features are selected as intermediate nodes and leaves. I have used five different tree-based fault prediction techniques.

They are—random forest, Simple CART, J48, BF Tree, and decision stump.

Function-based techniques are the one that can be written down as mathematical equations. Have used six different function-based fault prediction techniques.

They are—logistic regression, multilayer perceptron, simple logistic, SMO, RBFNetwork, and voted perceptron.

Logistic regression fits a regression model that minimizes the squared root error between the actual and the predicted value. It is used to obtain the output using regression if the output class is nominal.

SMO technique is an implementation of sequential minimal optimization algorithm for training a support vector machine. It is used polynomial or Gaussian kernels to build the support vectors.

Simple logistic is an implementation of logistic regression that uses a LogitBoost with simple regression functions as base learners. It determines the number of iterations to be performed using cross-validation, which supports automatic attribute selection.

Voted perceptron is the perceptron algorithm that uses a modified basic perceptron to multiplicative updates.

RBFNetwork is an implementation of Gaussian radial basis function. It uses K-means algorithm

to derive the centres and widths of hidden units and uses logistic regression to combine the outputs obtained from the hidden layer.

Multilayer perceptron consists a series of interconnected processing elements in the form of layers govern by some weights. During the training phase, the intermediate layers are updated iteratively until the difference between the actual value and predicted value becomes below certain threshold value.

Instance-based fault prediction techniques are the one that learn from the similar examples available in the training dataset to classify a given testing example. These techniques do not generate any explicit knowledge from the training dataset. Based on the value of the similar

examples in the training dataset, they classify the given testing example into one of the output classes. Have used two different instance-based fault prediction

techniques. They are—IB1 and IBK (K = 5).

Bayesian-based techniques are the techniques that use a statistical inference based on Bayes’ theorem to update the probability of a hypothesis as more supporting evidence become available. I have used two different Bayesian-based fault prediction techniques.

They are— Bayesian logistic regression and naive Bayes. Naive Bayes technique uses Bayes’ theorem with the assumption that each pair of attributes (software metrics) is independent

of each other.

Naive Bayes technique assumes that the presence of a particular feature in the dataset does not affect the presence of any other feature. Bayesian logistic regression uses Bayesian

inference to determine the logistic regression parameters. It first calculates the likelihood function of the data, and then it calculates the prior distribution of all the unknown parameters. Last, it applies Bayes’ theorem to determine the posterior distribution of all parameters.

In this predictive analytics fault detection project, I have used Bayed Based techniques that are:

1. Bayesian Logistic regression
2. Naïve Bayes.

Function based Techniques used in this project are:

1. Logistic Regression
2. Simple Logistic Regression

Instance based techniques used in this Predictive analytics project are :

1. IBI
2. IBK

Tree based Technique that I used in this project are:

1. Random Forest
2. J48

**Practical Setup**

**Performance Evaluation Measures**

I have built the fault prediction models for the binary class classification of software faults. Each fault prediction model has labelled the given software modules as faulty or non-faulty.

To assess the performance of built fault prediction models, have used accuracy, precision, recall, F-measure, and AUC (area under ROC) curve performance measures. The root of all these performance measures is the confusion matrix.

Mostly binary class classification of software faults have used some or all of these performance measures to evaluate the performance of the fault prediction techniques. For this reason, I have used these measures to carry out and perform overall assessment of the used fault prediction techniques.

**Software Fault Datasets Used**

In this work, I have used the software fault datasets available in the PROMISE data repository to evaluate the performance used fault prediction techniques. PROMISE data repository contained the fault datasets of various open-source software projects and others. Currently, it has contained fault datasets of sixty-five different software projects and their different releases. The size of the available fault datasets varies, and some datasets have only few software modules, whereas some other datasets have thousands of software modules. All the available datasets contained the

same twenty object-oriented software metrics. They are—

weighted method count (WMC),

coupling between objects (CBO),

response for a class (RFC),

depth of inheritance tree(DIT),

number of children (NOC), inheritance coupling(IC),

coupling between methods (CBM),

afferent coupling(CA),

efferent coupling (CE),

measure of functional abstraction (MFA),

lack of cohesion of methods (LCOM),

lack of cohesion of methods 3 (LCOM3),

cohesion among methods (CAM),

measure of aggregation (MOA),

number of public methods in a class (NPM),

data access metric (DAM),

average method complexity (AMC),

lines of code (LOC),

cyclomatic complexity (CC).

In the presented work project , I have used forty-six different software fault datasets out of total sixty-five available fault datasets having more than 100 software modules. I have discarded nineteen fault datasets having less than 100 software modules. The rationale behind discarding these datasets is that it is difficult to trust the results of datasets of very small size due to the requirement of some fault prediction techniques of the larger dataset for the training.

The description of the used software fault datasets I had mentioned in the table. For each given software project, I used multiple releases to build and evaluate the fault prediction model.

I have made a table depicting the metrics with abbreviations and full-forms of the names as the same are there in downloaded .csv files for bug-data which I had used in this project from PROMISE repo.

|  |  |
| --- | --- |
| **Short name of Columns in Metrics** | **The name of the module.** |
| version | The version of the software. |
| wmc | Weighted Methods per Class. |
| dit | Depth of Inheritance Tree. |
| noc | Number of Children. |
| cbo | Coupling between Objects. |
| rfc | Response for a Class. |
| lcom | Lack of Cohesion in Methods. |
| ca | Afferent Couplings. |
| ce | Efferent Couplings. |
| npm | Number of Public Methods. |
| lcom3 | Lack of Cohesion in Methods 3. |
| loc | Lines of Code. |
| dam | Data Access Metric. |
| moa | Measure of Aggregation. |
| mfa | Measure of Functional Abstraction. |
| cam | Cohesion Among Methods in Class. |
| ic | Inheritance Cohesion. |
| cbm | Coupling Between Methods. |
| amc | Average Method Complexity. |
| max\_cc | Maximum McCabe's Cyclomatic Complexity. |
| avg\_cc | Average McCabe's Cyclomatic Complexity. |
| bug | A column related to bugs. |

In this, for various software fault datasets tested from PROMISE repository, I used # non-commented -LOC, Total number of modules, total number of faulty modules and calculated % percentage of faulty modules using this formula (Number of Faulty Modules / Total Number of Modules) \* 100%.

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 | 208KLOC | 745 | 166 | 22.25 |
| 2 | Camel | Camel-1.0 | 33KLOC | 340 | 13 | 3.82 |
| 3 |  | Camel-1.2 | 66KLOC | 609 | 126 | 35.47 |
| 4 |  | Camel-1.4 | 98KLOC | 873 | 145 | 16.61 |
| 5 |  | Camel-1.6 | 113KLOC | 966 | 188 | 19.46 |
| 6 | Ivy | Ivy-1.1 | 27KLOC | 112 | 63 | 56.25 |
| 7 |  | Ivy-1.4 | 59KLOC | 242 | 16 | 6.61 |
| 8 |  | Ivy-2.0 | 87KLOC | 353 | 40 | 11.33 |
| 9 | Jedit | Jedit-3.2 | 128KLOC | 273 | 90 | 32.97 |
| 10 |  | Jedit-4.0 | 144KLOC | 307 | 75 | 24.43 |
| 11 |  | Jedit-4.1 | 153KLOC | 313 | 79 | 25.24 |
| 12 |  | Jedit-4.2 | 170KLOC | 368 | 48 | 13.04 |
| 13 |  | Jedit-4.3 | 202KLOC | 493 | 11 | 2.23 |
| 14 | Log4 j | Log4 j-1.0 | 21KLOC | 136 | 34 | 25 |
| 15 |  | Log4 j-1.1 | 19KLOC | 110 | 37 | 33.64 |
| 16 |  | Log4 j-1.2 | 38KLOC | 206 | 189 | 91.75 |
| 17 | Lucene | Lucene-2.0 | 50KLOC | 196 | 91 | 46.43 |
| 18 |  | Lucene-2.2 | 63KLOC | 248 | 144 | 58.06 |
| 19 |  | Lucene-2.4 | 102KLOC | 341 | 203 | 59.53 |
| 20 | Poi | Poi-1.5 | 55KLOC | 238 | 141 | 59.24 |
| 21 |  | Poi-2.0 | 93KLOC | 315 | 37 | 11.75 |
| 22 |  | Poi-2.5 | 119KLOC | 386 | 248 | 64.25 |
| 23 |  | Poi-3.0 | 129KLOC | 443 | 281 | 63.43 |
| 24 | Synapse | Synapse-1.0 | 28KLOC | 157 | 16 | 10.19 |
| 25 |  | Synapse-1.1 | 42KLOC | 223 | 60 | 26.91 |
| 26 |  | Synapse-1.2 | 53KLOC | 257 | 87 | 33.85 |
| 27 | Velocity | Velocity-1.4 | 51KLOC | 197 | 147 | 74.62 |
| 28 |  | Velocity-1.5 | 53KLOC | 215 | 142 | 66.05 |
| 29 |  | Velocity-1.6 | 57KLOC | 230 | 78 | 33.91 |
| 30 | Xalan | Xalan-2.4 | 225KLOC | 724 | 111 | 15.33 |
| 31 |  | Xalan-2.5 | 304KLOC | 804 | 387 | 48.13 |
| 32 |  | Xalan-2.6 | 411KLOC | 886 | 411 | 46.39 |
| 33 |  | Xalan-2.7 | 428KLOC | 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 | 202KLOC | 493 | 11 | 2.23 | **CATEGORY 1 (Less than 10 % of faulty modules)** |
| 2 | Camel | Camel-1.0 | 33KLOC | 340 | 13 | 3.82 |
| 7 |  | Ivy-1.4 | 59KLOC | 242 | 16 | 6.61 |
| 24 | Synapse | Synapse-1.0 | 28KLOC | 157 | 16 | 10.19 | **CATEGORY 2 (10% - 20% of faulty modules)** |
| 8 |  | Ivy-2.0 | 87KLOC | 353 | 40 | 11.33 |
| 21 |  | Poi-2.0 | 93KLOC | 315 | 37 | 11.75 |
| 12 |  | Jedit-4.2 | 170KLOC | 368 | 48 | 13.04 |
| 30 | Xalan | Xalan-2.4 | 225KLOC | 724 | 111 | 15.33 |
| 4 |  | Camel-1.4 | 98KLOC | 873 | 145 | 16.61 |
| 5 |  | Camel-1.6 | 113KLOC | 966 | 188 | 19.46 |
| 1 | Ant | Ant-1.7 | 208KLOC | 745 | 166 | 22.25 | **CATEGORY 3 (20% - 30% of Faulty Modules)** |
| 10 |  | Jedit-4.0 | 144KLOC | 307 | 75 | 24.43 |
| 14 | Log4 j | Log4 j-1.0 | 21KLOC | 136 | 34 | 25 |
| 11 |  | Jedit-4.1 | 153KLOC | 313 | 79 | 25.24 |
| 25 |  | Synapse-1.1 | 42KLOC | 223 | 60 | 26.91 |
| 9 | Jedit | Jedit-3.2 | 128KLOC | 273 | 90 | 32.97 | **CATEGORY 4 (Above 30% )** |
| 15 |  | Log4 j-1.1 | 19KLOC | 110 | 37 | 33.64 |
| 26 |  | Synapse-1.2 | 53KLOC | 257 | 87 | 33.85 |
| 29 |  | Velocity-1.6 | 57KLOC | 230 | 78 | 33.91 |
| 3 |  | Camel-1.2 | 66KLOC | 609 | 126 | 35.47 |
| 32 |  | Xalan-2.6 | 411KLOC | 886 | 411 | 46.39 |
| 17 | Lucene | Lucene-2.0 | 50KLOC | 196 | 91 | 46.43 |
| 31 |  | Xalan-2.5 | 304KLOC | 804 | 387 | 48.13 |
| 6 | Ivy | Ivy-1.1 | 27KLOC | 112 | 63 | 56.25 |
| 18 |  | Lucene-2.2 | 63KLOC | 248 | 144 | 58.06 |
| 20 | Poi | Poi-1.5 | 55KLOC | 238 | 141 | 59.24 |
| 19 |  | Lucene-2.4 | 102KLOC | 341 | 203 | 59.53 |
| 23 |  | Poi-3.0 | 129KLOC | 443 | 281 | 63.43 |
| 22 |  | Poi-2.5 | 119KLOC | 386 | 248 | 64.25 |
| 28 |  | Velocity-1.5 | 53KLOC | 215 | 142 | 66.05 |
| 27 | Velocity | Velocity-1.4 | 51KLOC | 197 | 147 | 74.62 |
| 16 |  | Log4 j-1.2 | 38KLOC | 206 | 189 | 91.75 |
| 33 |  | Xalan-2.7 | 428KLOC | 910 | 989 | 98.68 |

This to mention that, I have discarded nineteen fault datasets having less than 100 software modules. The rationale behind discarding these datasets is that it is difficult to trust the results of datasets of very small size due to the requirement of some fault prediction techniques of the larger dataset for the training.

I have pre-processed the datasets and removed the unique ID and path to the software module fields from the datasets. In this, focused on the binary class classification of software faults.

Therefore, I have transformed the number of faults information into faulty and non-faulty information. If a software module has one or more faults, then labelled that module as faulty otherwise non-faulty. Applied the same preprocess and transformation for all the used fault datasets.

The performance of fault prediction techniques may vary with respect to the percentage of faulty modules in the fault datasets. Therefore, categorized the used fault datasets into four different categories based the percentage of the faulty module and evaluated all the fault prediction techniques for each category i.e category 1 ,2 and 3 as in the above table last column with color highlighted rows.

If a fault dataset has less than 10% of the faulty software modules, then categorized it into category-1. If a fault dataset has faulty software modules between 10 and 20%, then categorized it into category-2. If a fault dataset has faulty software modules between 20 and 30%, then I categorized it into category-3. If a fault dataset has more than 30% faulty software modules, then I categorized it into category-4. This analysis has helped me to find out that whether the performance of fault prediction techniques has been affected by the percentage of faulty modules or not.

**Test Execution**

I have used **WEKA** machine learning tool in this practical setup execution to build and evaluate different Fault Prediction Models. All the reported practical tests utilizing different fault prediction techniques have been implemented using Weka tool by using the default values of various parameters of used techniques as given in Weka.

Note: have used a ten-fold cross-validation scheme to build and evaluate the fault prediction models.

Internal working : The original fault datasets has been partitioned into ten different parts. Each time nine parts are used as training dataset, and rest one part is used as testing dataset. This particular process has been repeated for ten times, and results are averaged for all the iterations.

# 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 that I created below corresponds to one category of software fault datasets. Each table shows the value of all five performance evaluation measures—accuracy, precision, recall, F-measure, and AUC.

**Results of Category -1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Technique** | **Metric** | **Min.** | **Average** | **Max.** |
|  |  |  |  |  |
| Bayesian Logistic | Accuracy | 90 | 93.12 | 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 | 0.5 | 0.5 | 0.51 |
| Naive Bayes | Accuracy | 80.15 | 87.94 | 93.69 |
|  | Precision | 0.86 | 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 | 0.84 | 0.9 | 0.96 |
|  | 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 | 0.86 | 0.9 | 0.97 |
|  | 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 | 0.85 | 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 | 0.85 | 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 |
| 148 | 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 |
| RBFNetwork | 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, I found that the Bayesian-based classifiers and tree-based classifiers have performed better compared to other used fault prediction techniques for this category-1 for all five performance evaluation measures.

**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 | 09 |
|  | 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 | 09 |
|  | 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 |
| RBFNetwork | 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 |

Category-2 - found that tree-based classifiers have performed better compared to the other used techniques for this category-2 for all five performance evaluation measures.

**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 |
| RBFNetwork | 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 |

Based on Results of Category -3, found that naive Bayes technique have performed better compared to the other used techniques for this category for all five performance evaluation measures.

**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 |
| RBFNetwork | 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 |

Based on Results of Category -4, found that random forest technique has performed better compared to the other used techniques for this category for all five performance evaluation measures.

Overall, based on the analysis of the results for all the categories, I found that when a fault dataset has lesser number of faulty software modules **Bayesian-based fault prediction techniques have performed the best followed by tree-based techniques**.

The results of the analysis showed that simple fault prediction techniques such naive Bayes, Bayesian logistic regression, and tree-based techniques outperformed other used fault prediction techniques for most of the cases.

Performed the operation for the fault datasets corresponding to the open-source software project to facilitate the reproduction of the experimental findings and to generalize the results.

These results can be used to select the appropriate fault prediction technique to predict the fault-proneness in the early phases of the software development.

**Fault Prediction Models Evaluation**

**Evaluation of the Techniques for the Prediction of Number of Faults**

The results of the analysis showed that simple fault prediction techniques such naive Bayes, Bayesian logistic regression, and tree-based techniques outperformed other used fault prediction techniques for most of the cases. In the evaluation for the fault datasets corresponding to the open-source software project to facilitate the reproduction of the experimental findings and to generalize the results. The results of the presented can be used in enterprise projects and by software practitioners to select the appropriate fault prediction technique to predict the fault-proneness in the early phases of the software development.

Have used three different fault datasets available in the Eclipse bug data repository to build the fault prediction models and to evaluate the performance of used fault prediction techniques for the prediction of number of faults. Eclipse bug data repository contains the fault information of three successive releases of Eclipse project available at the file-level. They are Eclipse-metric-files-2.0, Eclipse-metric-files-2.1, and Eclipse-metric-files-3.0. Each fault dataset contains various source code and structural software metrics and the information about the number of faults in each software module.

The software metrics are:

pre, NOM\_sum, NSM\_avg, ArrayCreation, Arraylnitializer, ArrayType, CharacterLiteral, ConditionalExpression, ContinueStatement, DoStatement, FieldAccess, Javadoc, LabeledStatement, ParenthesizedExpression, PrefixExpression, QualifiedName, ReturnStatement, SuperMethodInvocation, SwitchStatement, ThisExpression, ThrowStatement. In this project, I have selected 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-metric- 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 used to estimate the expected value of count models (NBR and ZIP) in terms of fault occurrences for each software module.

To estimate the parameters of count models, the maximum likelihood function is used.

Some of the software metrics have relatively larger values. To mitigate this issue, a square root transformation of software metrics and logarithmic transformation of LOC metric has been performed. The construction of count models has been done using **STATA** tool.

**Modelling Of Linear Regression, Multilayer Perceptron, and Decision Tree Regression**

I had implemented Linear Regression and Multilayer Perceptron using the Weka Machine Learning tool in previous stage. To estimate the parameters of the Linear Regression, used least square method, and rest of the parameter values of Linear regression have been initialized to their default values as per available in Weka tool.

Also, have used back-propagation algorithm and sigmoid function to train the Multilayer Perceptron.

For the rest of the parameters, default values as defined in Weka are used. Weka implementation of decision tree regression (as known as M5P) has been used to build the fault prediction model.

**Results And Analysis 2 of 4**

Results obtained from the practical execution of model analysis with respect to the AAE and ARE measures have been created and elaborated in the table below.

AAE analysis of 4 Fault Prediction techniques for all used datasets

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AAE Analysis Of 4 Prediction Techniques** | | | | |
| **Dataset** | **Linear Regression** | **Multilayer Perceptron** | **Decision Tree Regression** | **Negative -Binomial Regression** |
| 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 |

ARE analysis of 4 Fault Prediction techniques for all used 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-metric-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 values of AAE and ARE show the better performance of fault prediction techniques.

Note :

1. With respect to the AAE measure, linear regression and decision tree regression have produced better values as compared to the other used fault prediction techniques. For most of the techniques, the AAE value is below 0.50 except negative binomial regression, where the value is relatively higher.
2. With respect to the ARE measure, multilayer perceptron has produced better values as compared to the other used fault prediction techniques followed by decision tree regression and linear regression.
3. Again, negative binomial regression produced relatively higher value for the used performance measure.

**Results And Analysis 3 of 4**

Results obtained from the practical execution of model analysis with respect to the AAE and ARE measures have been created and elaborated in the table below.

AAE analysis of 4 Fault Prediction techniques for all used datasets

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AAE Analysis Of 4 Prediction Techniques** | | | | |
| **Dataset** | **Linear Regression** | **Multilayer Perceptron** | **Decision Tree Regression** | **Negative -Binomial Regression** |
| 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 |

ARE analysis of 4 Fault Prediction techniques for all used 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-metric-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 values of AAE and ARE show the better performance of fault prediction techniques.

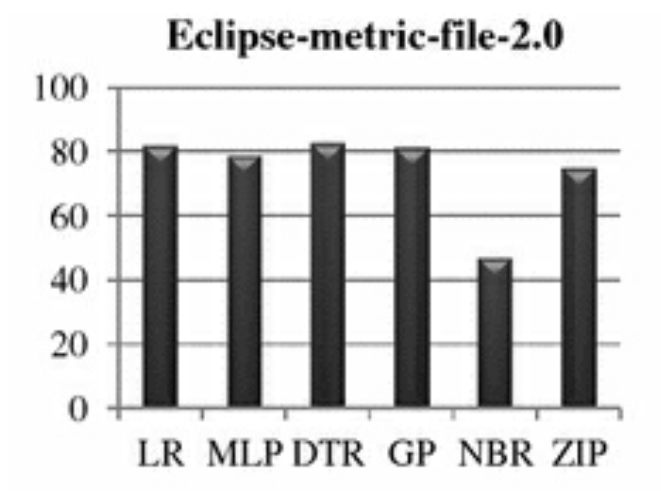
Note:

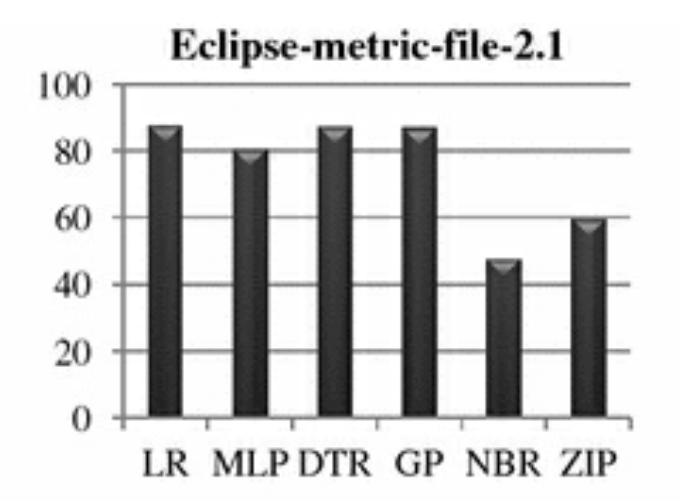
1. With respect to the AAE measure, linear regression and decision tree regression have produced better values as compared to the other used fault prediction techniques. For most of the techniques, the AAE value is below 0.50 except negative binomial regression, where the value is relatively higher.
2. With respect to the ARE measure, multilayer perceptron has produced better values as compared to the other used fault prediction techniques followed by decision tree regression and linear regression.
3. Again, negative binomial regression produced relatively higher value for the used performance measure.

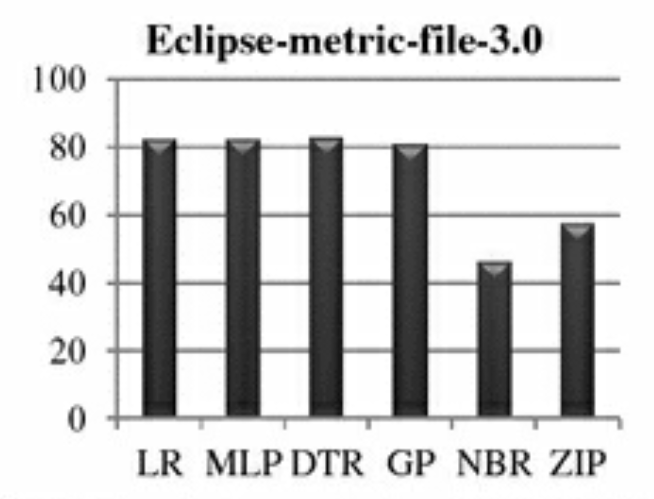
**Results And Analysis 4 of 4**

**Visualization and Interpretation:**

In the screenshots below, the results of the used fault prediction techniques for pred(0.3) analysis shown here.







In each graph, x-axis shows the used fault prediction techniques and y-axis shows the value of pred(0.3) analysis of each technique. From figure, it is observed that for most of the cases, fault prediction techniques have achieved pred(0.3) value greater than 80%. Exception has been found for negative binomial regression, where the value is lower than 50% of used fault datasets.

# 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.

# 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.

**To estimate the parameters of count models, the maximum likelihood function is used. Some of the software metrics have relatively larger values. To mitigate this issue, a square root transformation of software metrics and logarithmic transformation of LOC metric has been performed**

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

# 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.

# 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 that are more likely to contain defects, 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.

# Annexure 1

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

# 

Pseudo code describing the building and evaluation of the 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*](https://www.mdpi.com/2071-1050/15/6/5517)
13. [*https://www.sciencedirect.com/science/article/pii/S1319157818313077*](https://www.sciencedirect.com/science/article/pii/S1319157818313077)
14. [*https://core.ac.uk/download/pdf/185520882.pdf*](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 used 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. |
| Feature Selection | - | The process of selecting the most relevant and informative code metrics to be used as input features for the defect prediction model. |
| Ensemble Methods | - | Machine learning techniques that combine multiple models to improve prediction accuracy and reduce overfitting. Examples include Random Forest and Gradient Boosting. |
| 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. |

**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 | 4th |
| 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 , currently a research scholar pursuing PhD in Management. | MCA |
| 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

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **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***](mailto:project@wilp.bits-pilani.ac.in)*for any queries / clarifications*