A project report on

SOFTWARE FAULT PREDICTION

Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology (IT)

Under the supervision of

Mr. Varun Goel
Assistant Professor

Mr. Sachin Garg
Assistant Professor

Submitted by

Harshit Saini

(20214803119)

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

SECTOR-22, ROHINI, DELHI –110086

December 2022

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Mr. Varun Goel (Assistant Professor),** Department of Information Technology, MAIT, for their guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr. M.L. Sharma**, Head of the Department, Information Technology, for providing us with the required facilities for the completion of the project work. We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

Harshit Saini
20214803119

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this project report titled, **"SOFTWARE FAULT PREDICTION"** submitted by me in the partial fulfilment of the requirement of the award of the degree of **Bachelor of Technology (B.Tech.)** Submitted in the Department of **Information Technology,** Maharaja Agrasen Institute of Technology is an authentic record of my project work carried out under the guidance of **Mr. Varun Goel** and **Mr. Sachin Garg**. The matter presented in this project report has not been submitted either in part or full to any university or Institute for award of any degree.

Date :

Place: Delhi

Harshit Saini

Roll No.: 20214803119

SUPERVISOR'S CERTIFICATE

It is to certify that the Project entitled **"SOFTWARE FAULT PREDICTION"** which is being submitted by **Mr. Sachin Garg** and **Mr. Varun Goel** to the Maharaja Agrasen Institute of Technology, Rohini in the fulfilment of the requirement for the award of the degree of **Bachelor of Technology (B.Tech.)**, is a record of bonafide project work carried out by him/her under my/ our guidance and supervision.

Mr. Sachin Garg
Assistant Professor
Department of IT

Mr. Varun Goel
Assistant Professor
Department of IT

Prof. (Dr.) M.L. Sharma
H.O.D.
Department of IT

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Software fault prediction aims to identify fault-prone software modules by using some underlying properties of the software project before the actual testing process begins. It helps in obtaining desired software quality with optimized cost and effort. Initially, this paper provides an overview of the software fault prediction process. Next, different dimensions of software fault prediction process are explored and discussed. This review aims to help with the understanding of various elements associated with fault prediction process and to explore various issues involved in the software fault prediction. We search through various digital libraries and identify all the relevant papers published since 1993. The review of these papers are grouped into three classes: software metrics, fault prediction techniques, and data quality issues. For each of the class, taxonomical classification of different techniques and our observations have also been presented. The review and summarization in the tabular form are also given. At the end of the paper, the statistical analysis, observations, challenges, and future directions of software fault prediction have been discussed.

CHAPTER 1: INTRODUCTION

1.1 Need of the study

The IT and software industry has grown tremendously over the past few years, creating an increasing impact on the lives of people and on society as a whole. Consequently, we must make the software and applications more accurate, free of major errors, and more reliable. Therefore, predicting software flaws could be very useful in the IT field and will have a profound impact on society at large.

We can predict system faults and bugs by analyzing software systems, which include databases and local state machines. We take a very methodical approach to finding bugs, optimizing the program for efficiency and quality. This may lead to improved customer satisfaction, increased customer retention and save money.

1.2 Scope of the study

Machine learning techniques have been around us and have been compared and used for analysis for many kinds of data science applications. The major motivation behind this research-based project was to explore the feature selection methods, data preparation, and processing behind the training models in machine learning. With first- hand models and libraries, the challenge we face today is data were beside their abundance, and in our cooked models, the accuracy we see during training, testing, and actual validation has a higher variance. Hence this project is carried out with the motivation to explore behind the models, and further implement Logistic Regression model to train the obtained data.

Software fault prediction aims to predict fault-prone software modules by using some under- lying properties of the software project. It is typically performed by training a prediction model using project properties augmented with fault information for a known project, and subsequently using the prediction model to predict faults for unknown projects. Software fault prediction is based on the understanding that if a project developed in an environment leads to faults, then any module developed in the similar environment with similar project characteristics will ends to be faulty. The early detection of faulty modules can be useful to streamline the efforts to be applied in the later phases of software development by better focusing quality assurance efforts to those modules.

1.3 Objectives of the study

The main objectives of developing this project are:

- To determine algorithms using machine learning that can perform better in predicting the faults in software than the existing ones.
- To improve the results with the help of current datasets available on the internet using our algorithms.
- To use different evaluation benchmarks to evaluate the performance of our model.
- The main objective of the project is to create an automated software fault predicting model using machine learning techniques that will make the software more efficient and reliable to use.

CHAPTER 2: LITERATURE REVIEW

In this sub-section, we discuss the previously published review papers on defect prediction. Catal and Diri analyzed software defect prediction articles with respect to different software metrics, datasets, and approaches [16]. Malhotra and Jain analyzed the prior publications and published a review paper on defect prediction [17]. Malhotra reviewed publications from 1991 to 2013 that apply machine learning methods for software defect prediction [18].

Radjenovic et al. analyzed defect prediction papers published from 1991s to 2011 and reported that machine learning methods and object-oriented metrics were widely applied for fault detection in the literature [19]. Misirli et al. analyzed 38 publications using machine learning methods and presented a systematic mapping study. They reported that machine learning algorithms such as Bayesian networks were used in 70% of studies [20].

Morera et al. reviewed studies on software defect prediction from 2002 to 2014, selected 40 studies, and presented a systematic mapping study on software defect prediction. They discussed the performance of machine learning methods such as Random Forest, Naïve Bayes, Logistic Regression, and Decision Trees [21].

Ozakıncı et al. reviewed publications published between 2000 and 2016 and selected 52 publications. They investigated the aim, development, progress, advantages, and components of models and presented a systematic review [22]. Son et al. performed a systematic mapping study of software defect prediction studies using 156 articles and reported that very few studies described cross-project defect prediction [23].

In addition, several systematic literature reviews (SLR) and systematic mapping stud- ies (SMS) have been published in the software engineering discipline so far. Najm et al. analyzed studies published until 2017, which used a Decision Trees (DT) algorithm for software development effort estimation in their SMS study. The selected publications are categorized based on publication platform, analysis model, research strategy approaches applied in organizations [24].

Alsolai et al. analyzed publications related to the software maintainability prediction and presented an SLR study. They reported that the authors used some private datasets in some papers, evaluated their models using k-fold cross-validation approaches, and applied named regression algorithms [25].

CHAPTER 3: IMPLEMENTATION OF PROPOSED SYSTEM

3.1 System Requirements

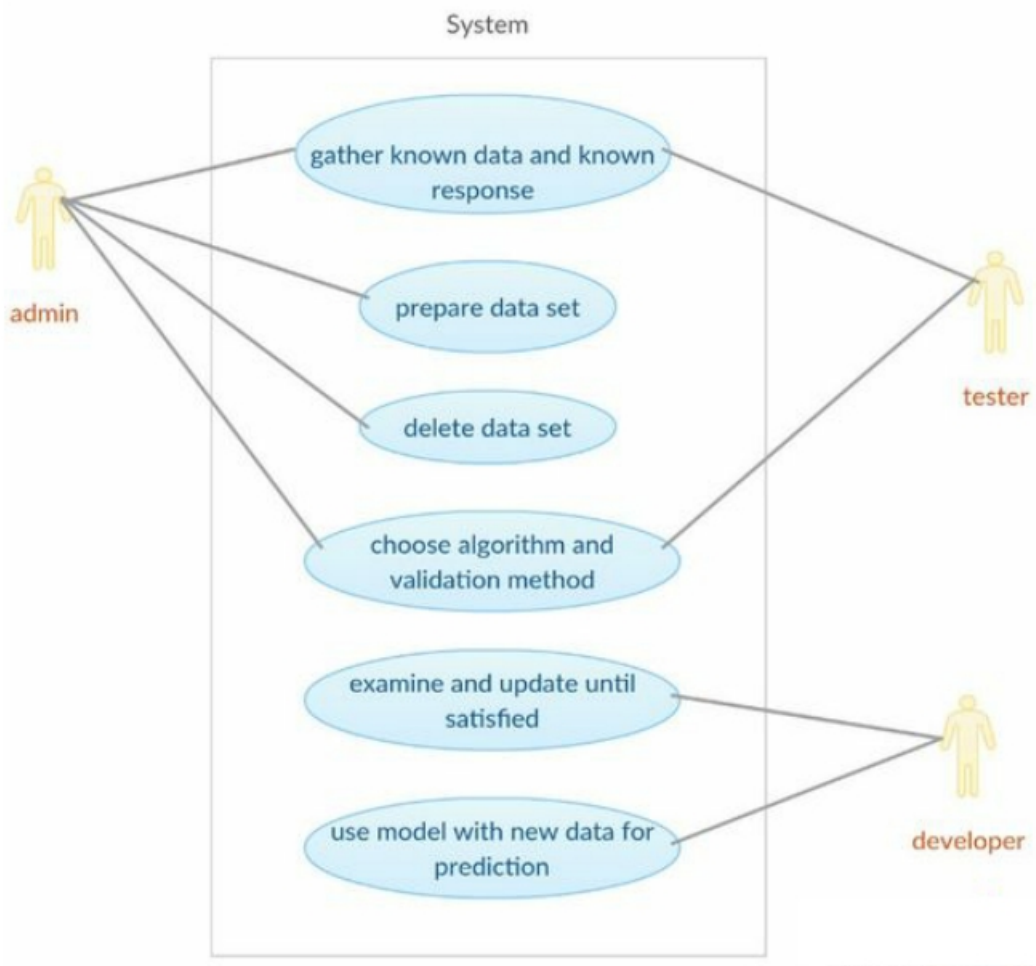
3.1.1 Hardware requirements:

Processer	:	Any Update Processer
Ram	:	Min 4GB
Hard Disk	:	100GB Free space

3.1.2 Software requirements:

Operating System	:	Windows Family Technology (Python3.7)
IDE	:	Jupyter notebook

3.2 Use Case Diagram



3.3 Flow Chart

3.4 Proposed System

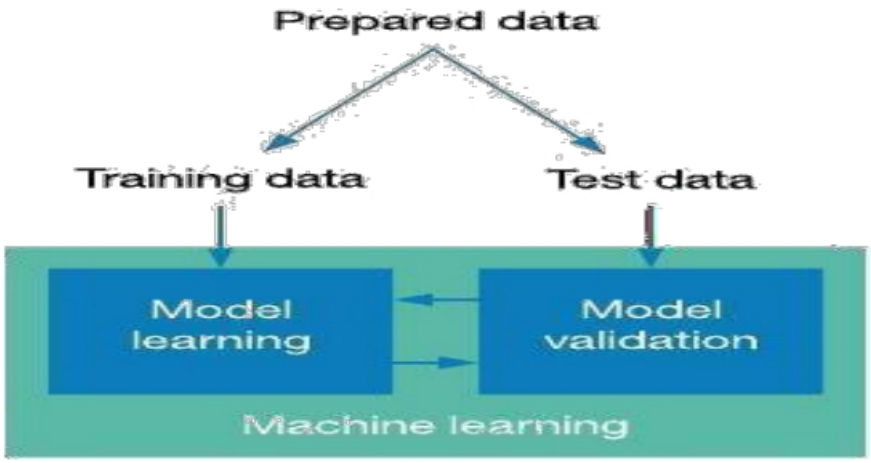
The working of the system starts with the collection of data and selecting the important attributes. Then the required data is preprocessed into the required format. The data is then divided into two parts training and testing data. The algorithms are applied and the model is trained using the training data. The accuracy of the system is obtained by testing the system using the testing data. This system is implemented using the following modules.

1. Collection of Dataset
2. Selection of attributes
3. Data Pre-Processing
4. Balancing of Data
5. Fault Prediction

3.4.1 Collection of Dataset

Initially, we collect a dataset for our software fault prediction system. After the collection of the dataset, we split the dataset into training data and testing data. The training dataset is used for prediction model learning and testing data is used for evaluating the prediction model. For this

project, 70% of training data is used and 30% of data is used for testing. The dataset used for this project is JM1 an CM1. The dataset consists of 22 attributes and we are using all of them.



3.4.2 Selection of attributes

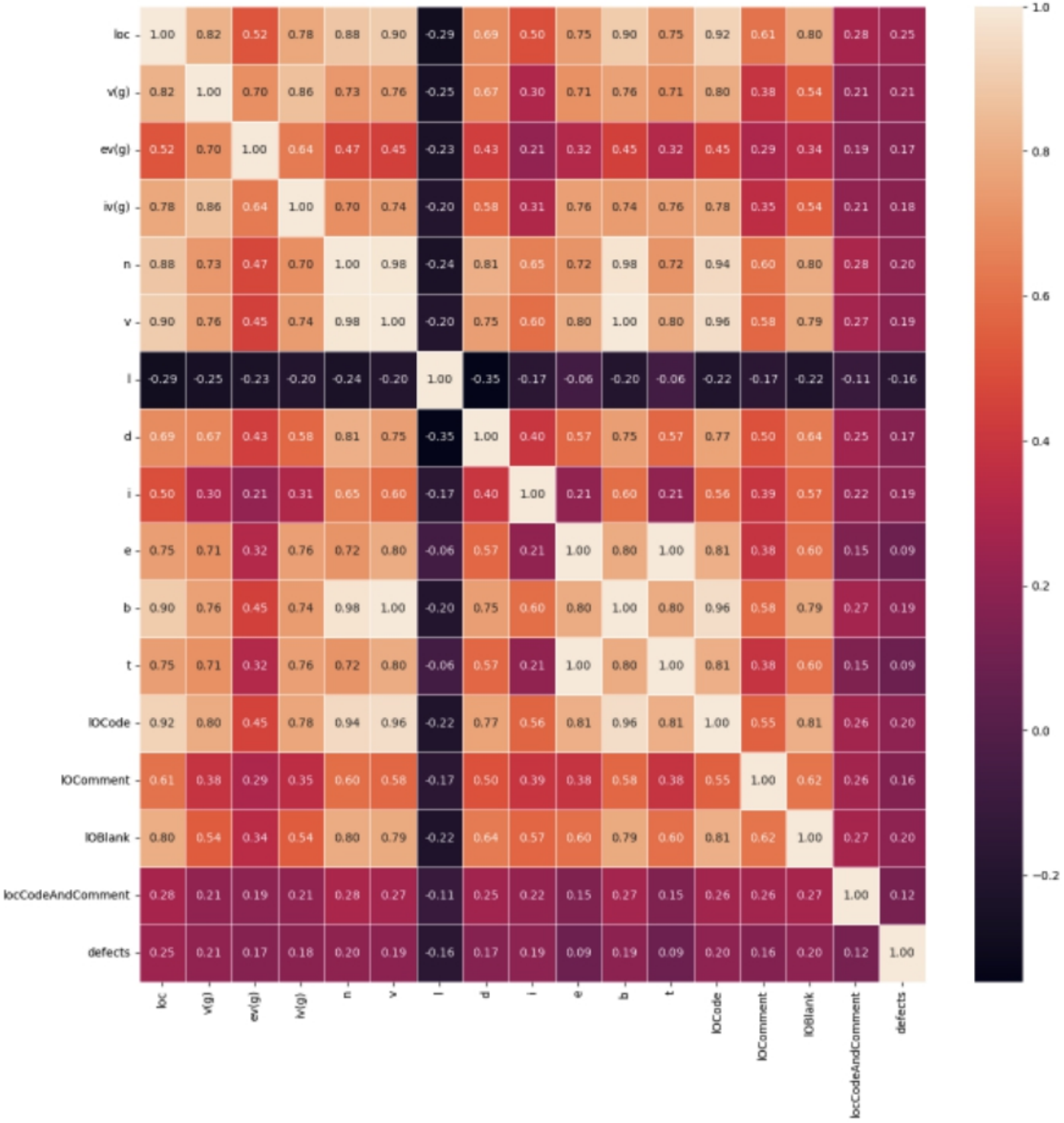
Attribute or Feature selection includes the selection of appropriate attributes for the prediction system. This is used to increase the efficiency of the system. Various attributes of the software like loc, branch count, defects, unique operands, etc. are selected for the prediction. The Correlation matrix is used for attribute selection for this model.

3.4.3 Data Pre-Processing

Data pre-processing is an important step for the creation of a machine learning model. Initially, data may not be clean or in the required format for the model which can cause misleading outcomes. In pre-processing of data, we transform data into our required format. It is used to deal with noises, duplicates, and missing values of the dataset. Data pre-processing has the activities like importing datasets, splitting datasets, attribute scaling, etc. Preprocessing of data is required for improving the efficiency and accuracy of the model / system.

3.4.4 Balancing of Data

Imbalanced datasets can be balanced in two ways. They are Under Sampling and Over Sampling (a) (a) Under Sampling: In Under Sampling, dataset balance is done by the reduction of the size of the ample class. This process is considered when the amount of data is adequate.



(b) Over Sampling: In Over Sampling, dataset balance is done by increasing the size of the scarce samples. This process is considered when the amount of data is inadequate.

3.4.5 Fault Prediction

Various machine learning algorithms like Linear Regression, Naive Bayes, Decision Tree, Random Tree, K-nearest neighbor (KNN), etc are used for classification and prediction. Comparative analysis is performed among algorithms and the algorithm that gives the highest accuracy is used for software fault prediction.

CHAPTER 4: EXPERIMENTAL ANALYSIS

4.1 Machine Learning

In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.

- **Supervised Learning**

Supervised learning is the type of machine learning in which machines are trained using well "labeled" training data, and on the basis of that data, machines predict the output. The labeled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

- **Unsupervised learning**

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar to how a human learns to think by their own experiences, which makes it closer to the real AI.

- **Reinforcement learning**

Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

4.2 Algorithms

4.2.1 Naive Bayes

Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset.

Naive Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

The Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

The Naive Bayes algorithm is comprised of two words Naive and Bayes, Which can be described as:

- **Naive**: It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the basis of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' theorem:

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability. The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Types of Naive Bayes model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.

- **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.

- **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

4.2.2 Decision Tree

Decision Tree is a supervised learning technique that can be used for both classification and regression problems, but mostly it is preferred for solving classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision Tree, there are two nodes, which are the Decision Node and Leaf Node.

Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a Decision Tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm. A Decision Tree simply asks a question, and based on the answer (Yes/No), it further split the tree into sub trees.

The Decision Tree Algorithm belongs to the family of supervised machine learning algorithms. It can be used for both a classification problem as well as for a regression problem.

The goal of this algorithm is to create a model that predicts the value of a target variable, for which the decision tree uses the tree representation to solve the problem in which the leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision Tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

In Decision Tree the major challenge is to identify the attribute for the root node in each level. This process is known as attribute selection. We have two popular attribute selection measures:

1. Information Gain:

When we use a node in a Decision Tree to partition the training instances into smaller subsets, the entropy changes. Information gain is a measure of this change in entropy.

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy the more the information content.

2. Gini Index:

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified. It means an attribute with lower Gini index should be preferred. Sklearn supports "Gini" criteria for Gini Index and by default, it takes "gini" value.

The most notable types of Decision Tree algorithms are:-

1. **IDichotomiser 3 (ID3):** This algorithm uses Information Gain to decide which attribute is to be used to classify the current subset of the data. For each level of the tree, information gain is calculated for the remaining data recursively.

2. **C4.5:** This algorithm is the successor of the ID3 algorithm. This algorithm uses either Information gain or Gain ratio to decide upon the classifying attribute. It is a direct improvement from the ID3 algorithm as it can handle both continuous and missing attribute values.

3. **Classification and Regression Tree (CART):** It is a dynamic learning algorithm which can produce a regression tree as well as a classification tree depending upon the dependent variable.

Working:

In a Decision Tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of the root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contains possible values for the best attributes.
- Step-4: Generate the Decision Tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.

4.2.3 Support Vector Machine (SVM)

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyper plane. SVM chooses the extreme points/ vectors that help in creating the hyper plane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine.

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In the 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

The followings are important concepts in SVM -

- Support Vectors - Data Points that are closest to the hyper plane are called support vectors. Separating line will be defined with the help of these data points.
- Hyper plane - As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.

- Margin - It may be defined as the gap between two lines on the closest data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

Types of SVM:

SVM can be of two types:

- Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

- Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

The objective of the support vector machine algorithm is to find a hyperplane in an N- dimensional space (N - the number of features) that distinctly classifies the data points.

The advantages of support vector machines are:

- Effective in high dimensional spaces.

- Still effective in cases where the number of dimensions is greater than the number of samples.

- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

- Versatile: different kernel functions can be specified for the decision function.

- Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over- fitting in choosing Kernel functions and regularization term is crucial.

- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

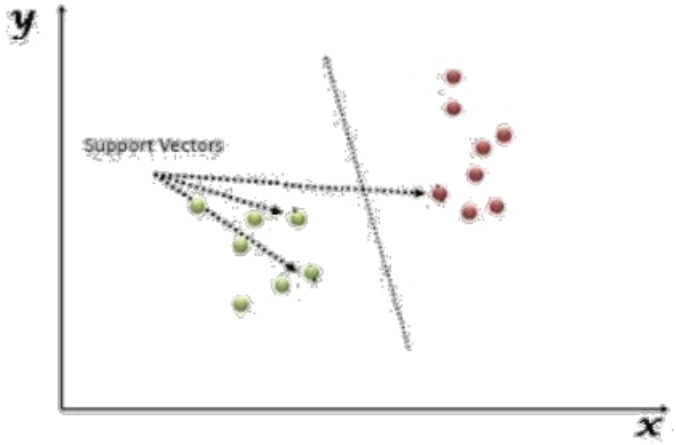


Figure: Support Vector Machine

4.2.4 Random Forest Algorithm

Random Forest is a supervised learning algorithm. It is an extension of machine learning classifiers which include the bagging to improve the performance of Decision Tree. It combines tree predictors, and trees are dependent on a random vector which is independently sampled. The distribution of all trees are the same. Random Forests splits nodes using the best among of a predictor subset that are randomly chosen from the node itself, instead of splitting nodes based on the variables. The time complexity of the worst case of learning with Random Forests is O(M(dnlogn)) , where M is the number of growing trees, n is the number of instances, and d is the data dimension.

It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest consists of trees. It is said that the more trees it has, the more robust a forest is. Random Forests create Decision Trees on randomly selected data samples, get predictions from each tree and select the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

Random Forests have a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of over fitting.

Assumptions:

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Advantages:

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages:

Although Random Forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

4.2.5 K-Nearest Neighbors (KNN)

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

Regression problems use a similar concept as classification problem, but in this case, the average the k nearest neighbors is taken to make a prediction about a classification.

The main distinction here is that classification is used for discrete values, whereas regression is used with continuous ones. However, before a classification can be made, the distance must be defined. Euclidean distance is most commonly used, which we'll delve into more below.

It's also worth noting that the KNN algorithm is also part of a family of "lazy learning" models, meaning that it only stores a training dataset versus undergoing a training stage. This also means that all the computation occurs when a classification or prediction is being made. Since it heavily relies on memory to store all its training data, it is also referred to as an instance-based or memory-based learning method.

Compute KNN: defining k

The k value in the k-NN algorithm defines how many neighbors will be checked to determine the classification of a specific query point. For example, if k=1, the instance will be assigned to the same class as its single nearest neighbor. Defining k can be a balancing act as different values can lead to overfitting or underfitting. Lower values of k can have high variance, but low bias, and larger values of k may lead to high bias and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k. Overall, it is recommended to have an odd number for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset.

Advantages

- **Easy to implement:** Given the algorithm's simplicity and accuracy, it is one of the first classifiers that a new data scientist will learn.

- **Adapts easily:** As new training samples are added, the algorithm adjusts to account for any new data since all training data is stored into memory.

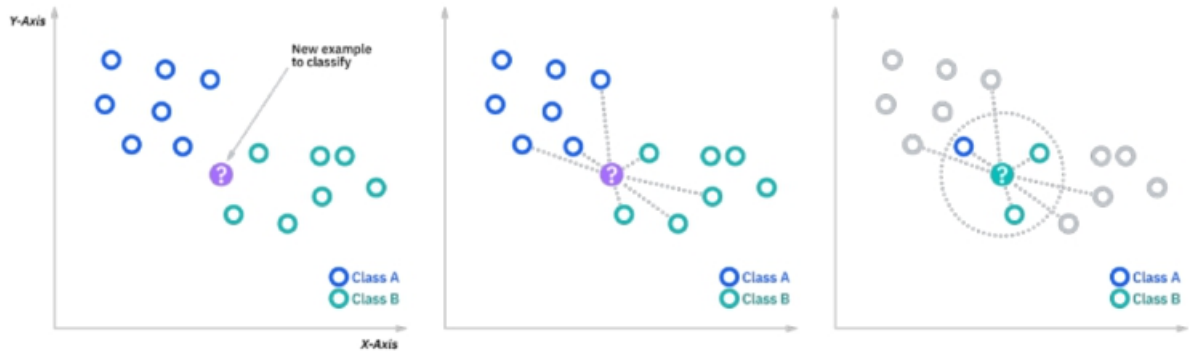
- **Few hyper parameters:** KNN only requires a k value and a distance metric, which is low when compared to other machine learning algorithms.

Disadvantages

- **Does not scale well**: Since KNN is a lazy algorithm, it takes up more memory and data storage compared to other classifiers. This can be costly from both a time and money perspective. More memory and storage will drive up business expenses and more data can take longer to compute. While different data structures, such as Ball-Tree, have been created to address the computational inefficiencies, a different classifier may be ideal depending on the business problem.

- **Curse of dimensionality:** The KNN algorithm tends to fall victim to the curse of dimensionality, which means that it doesn't perform well with high-dimensional data inputs. This is sometimes also referred to as peaking phenomenon, where after the algorithm attains the optimal number of features, additional features increases the amount of classification errors, especially when the sample size is smaller.

- **Prone to overfitting:** Due to the "curse of dimensionality", KNN is also more prone to overfitting. While feature selection and dimensionality reduction techniques are leveraged to prevent this from occurring, the value of k can also impact the model's behavior. Lower values of k can overfit whereas higher values of k tend to "smooth out" the prediction values since it is averaging the values over a greater area, or neighborhood. However, if the value of k is too high, then it can underfit the data.



4.2.6 Logistic Regression Algorithm

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas logistic regression is used for solving the classification problems.

In Logistic regression, instead of fitting a regression line, we fit an "S"shaped logistic function, which predicts two maximum values (0 or 1).

The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.

Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

Advantages:

- Logistic Regression is one of the simplest machine learning algorithms and is easy to implement yet provides great training efficiency in some cases. Also due to these reasons, training a model with this algorithm doesn't require high computation power.

- The predicted parameters (trained weights) give inference about the importance of each feature. The direction of association i.e. positive or negative is also given. So we can use Logistic Regression to find out the relationship between the features.

- This algorithm allows models to be updated easily to reflect new data, unlike Decision Tree or Support Vector Machine. The update can be done using stochastic gradient descent.

- Logistic Regression outputs well-calibrated probabilities along with classification results. This is an advantage over models that only give the final classification as results. If a training example has a 95% probability for a class, and another has a 55% probability for the same class, we get an inference about which training examples are more accurate for the formulated problem.

Disadvantages:

- Logistic Regression is a statistical analysis model that attempts to predict precise probabilistic outcomes based on independent features. On high dimensional datasets, this may lead to the model being over-fit on the training set, which means overstating the accuracy of predictions on the training set and thus the model may not be able to predict accurate results on the test set. This usually happens in the case when the model is trained on little training data with lots of features. So on high dimensional datasets, Regularization techniques should be considered to avoid over- fitting (but this makes the model complex). Very high regularization factors may even lead to the model being under-fit on the training data.

- Non linear problems can't be solved with logistic regression since it has a linear decision surface. Linearly separable data is rarely found in real world scenarios. So the transformation of non linear features is required which can be done by increasing the number of features such that the data becomes linearly separable in higher dimensions.

- **Non-Linearly Separable Data:** It is difficult to capture complex relationships using logistic regression. More powerful and complex algorithms such as Neural Networks can easily outperform this algorithm

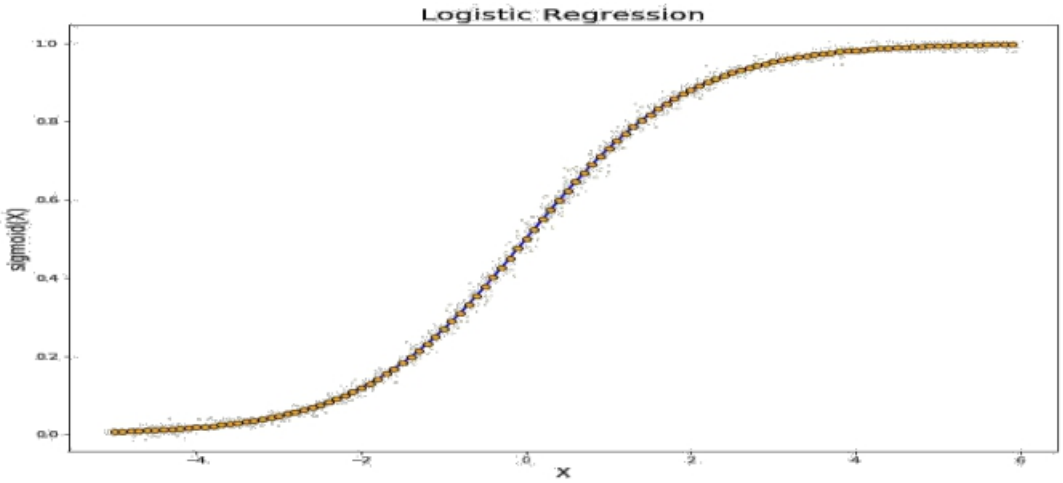


Figure: Logistic Regression

4.3 Dataset Details

In this project, we have used 3 open source publicly available data from PROMISE Software Engineering Database. These datasets Tim Menzies et al. have been used in their research paper [1]. In another study, Jureczko et al. [2] have been assembled a software fault prediction model to predict the software defects using machine learning algorithms. They have discussed in their paper about 8 projects (PROMISE Repository) data and by taking 19 CK metrics and McCabe metrics for constructed a predictive model. In our study, we have used 22 attributes for building our automated fault predict model. Table 1 shows 22 different attributes from software defect datasets including 21 independent metrics and one is outcome information. i.e. which is faulty and no-fault. We are using JM1, CM1, PC1 datasets which were implemented in C language.

Table 1. List of the metrics

No	Metrics name	Type
1	Line of code	McCabe
2	Cyclomatic complexity	McCabe
3	Essential complexity	McCabe
4	Design complexity	McCabe
5	Halstead operators and operands	Halstead
6	Halstead volume	Halstead
7	Halstead program length	Halstead
8	Halstead difficulty	Halstead
9	Halstead intelligence	Halstead
10	Halstead effort	Halstead
11	Halstead time estimator	Halstead
12	Halstead line count	Halstead
13	Halstead comments count	Halstead
14	Halstead blank line count	Halstead
15	IO code and comments	Miscellaneous
16	Unique operators	Miscellaneous
17	Unique operands	Miscellaneous
18	Total operators	Miscellaneous
19	Total operands	Miscellaneous
20	Branch count	Miscellaneous
21	b: numeric	Halstead
22	Defects	False or true

We are using JM1, CM1, PC1 datasets which were implemented in C language. Table 2 depicted details about detail of all datasets with their features.

Table 2: Details about datasets

No	Dataset	Missing attribute	Instance	Class distribution	
				True	False
1	JM1	None	10885	8779 (80.65%)	2106 (19.35%)
2	CM1	None	498	49 (9.83%)	449 (90.16%)
3	PC1	None	1109	1032 (93.05%)	77 (6.94%)

4.4 Performance Analysis

Once the predictive model has been built, it can be applied to perform a test to predict the fault modules inside the software fault datasets. In this work, we examined the ML prediction models, utilizing six classification algorithms, based on different statistical techniques such as confusion matrix (True Positive = TP, True Negative = TN, False Positive = FP, False Negative = FN), recall, precision, F1 measure, etc. Table 3 shows a quality measure of predictive model based on confusion matrix as below

Table 3. Performance measurement criteria

Metrics	Mathematical formula
Accuracy	$\frac{(TP+TN)}{(TP+FP+TN+FN)}$
Precision	$\frac{TP}{(TP+FP)}$
Recall = TPR	$\frac{TP}{(TP+FN)}$
F1 measure	$\frac{2*(Recall*Precision)}{(Recall+Precision)}$
Specificity = TNR	$\frac{TN}{(TN+FP)}$

4.5 Experimental Setup

The proposed SDPD model and procedure of the experiment are determined based on the following aspects. First, the integrated software development process and its uses of the implementation as an input when the requirement analysis given into the predictive model for software development. Second, the high-level diagram and detail level diagram are mandatory to build a scalable SDPD model. Third, SDPD development analyzes defect modules to provide knowledge-based automated fault recovery of the software systems. And it is lead to predict defects and contain all the required data about faulty modules of application. Figure 1. SDPD shown the main components of proposed SDPD approach.

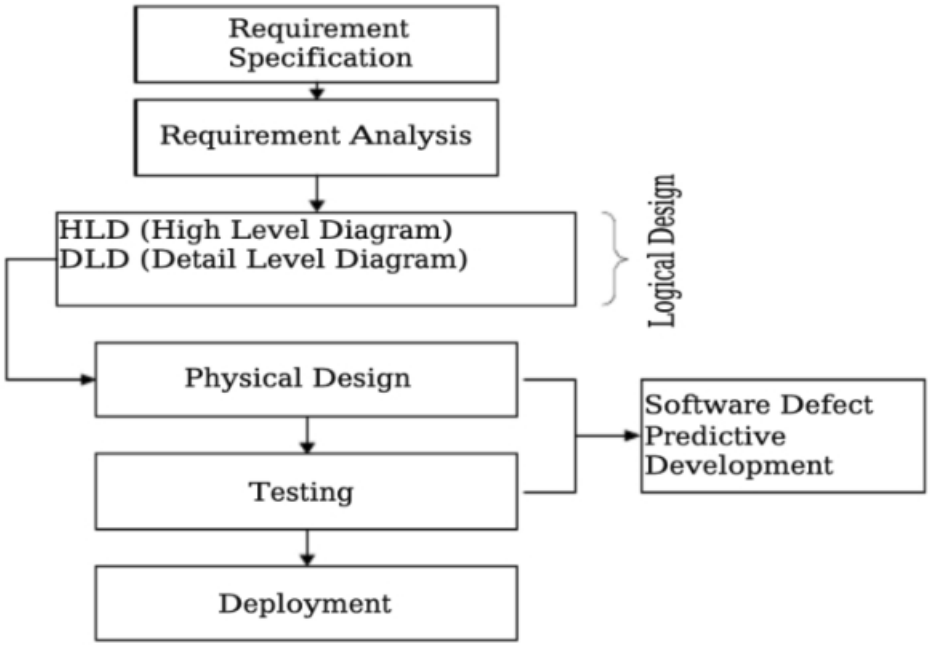


Figure 1. Proposed Software Defect Predictive Development Model

Our goal is to build a defect model that significantly increases the prediction accuracy with less number of features. To achieve this goal, we applied data preprocessing techniques such as Covariance Analysis, Feature Extraction and Min-Max Normalization on our data to find high correlation inside data, identify missing values and outlier in dataset by removing them from training data. Finally, we used SVD on our new dataset for feature extraction process. Details workflow of our defect model for SDPD based application is presented in Figure 2.

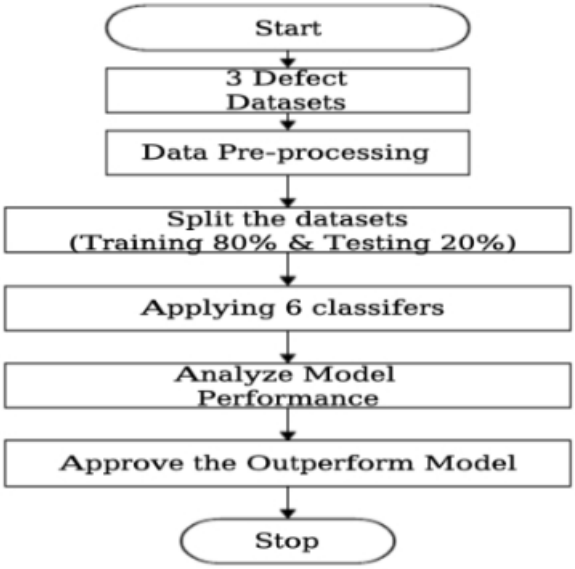


Figure 2. Workflow of the defect model

4.6 Result

We used the machine learning techniques to predict defects in software. In this study, we focused on automated fault recovery inside software through a predictive model, besides we also observed Requirement Specification Requirement Analysis Physical Design HLD (High Level Diagram) DLD (Detail Level Diagram) Deployment Software Defect Performance 3 Defect Datasets Split the datasets (Training 80% & Testing 20%) Approve the Outperform Model. Table 4 shows the performance evaluation of six supervised classification techniques for software fault prediction.

Table 4. Classification performance of ml techniques

Algorithms	Performance Measurement	Datasets		
		JM1	CM1	PC1
DT	Precision	1.0	1.0	.97
	Recall	0.99	1.0	1.0
	Accuracy	0.99	1.0	.99
	F1	0.99	1.0	.98
NB	Precision	0.94	1.0	.91
	Recall	0.93	1.0	.85
	Accuracy	0.98	1.0	.98
	F1	0.93	1.0	.87
RF	Precision	.99	1.0	.97
	Recall	1.0	1.0	.97
	Accuracy	0.99	1.0	.99
	F1	0.99	1.0	.97
KNN	Precision	0.95	.95	.86

CHAPTER 5: CONCLUSION AND FUTURE SCOPE

In this study, we proposed an automated software engineering approach for defect prediction model development (SDPD) on software development life cycle. After that, the main objective of our study was to evaluate the abilities of 4 supervised based machine learning classifications techniques to predict the software defect modules using 3 NASA datasets (JM1, CM1, PC1). The results (i.e. accuracy: 90 - 98%) of the experiment with different attributes showed the capability and efficiency of SDPD model to identify the fault and improve software quality.

In addition, this SDPD model can be able to early detection of software faults by collecting real-time software development data from the target applications. The proposed approach can be used for software fault recovery inside a system and enhanced by applying machine learning techniques.

For future work, we will implement more classification algorithms, such as hybrid or ensemble model to verify the performance of the software fault prediction.

ANNEXURE 1 : Code

[35]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
%pip install -q seaborn
import seaborn as sns # statistical data visualization

#-- plotly
%pip install -q chart-studio
import micropip
await micropip.install("ssl")
import chart_studio.plotly as py
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
import plotly.graph_objs as go
#--

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the

import os
#print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

[36]:
```
data = pd.read_csv('./data/jm1.csv')
```

About this Software Defect Prediction Dataset

```
This is a Promise data set made publicly available in order to encourage repeatable, verifiable,
refutable, and/or improvable predictive models of software engineering.
```

Attribute Information:

1. loc : numeric % McCabe's line count of code
2. v(g) : numeric % McCabe "cyclomatic complexity"
3. ev(g) : numeric % McCabe "essential complexity"
4. iv(g) : numeric % McCabe "design complexity"
5. n : numeric % Halstead total operators + operands
6. v : numeric % Halstead "volume"
7. l : numeric % Halstead "program length"
8. d : numeric % Halstead "difficulty"
9. i : numeric % Halstead "intelligence"
10. e : numeric % Halstead "effort"
11. b : numeric % Halstead
12. t : numeric % Halstead's time estimator
13. lOCode : numeric % Halstead's line count
14. lOComment : numeric % Halstead's count of lines of comments
15. lOBlank : numeric % Halstead's count of blank lines

16. lOCodeAndComment : numeric
17. uniq_Op : numeric % unique operators
18. uniq_Opnd : numeric % unique operands
19. total_Op : numeric % total operators
20. total_Opnd : numeric % total operands
21. branchCount : numeric % of the flow graph
22. defects : {false,true} % module has/has not one or more reported defects

[38]:
```
data.head() #shows first 5 rows
```

[38]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6

5 rows × 22 columns

[39]:
```
data.tail() #shows last 5 rows
```

[39]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank
10880	18.0	4.0	1.0	4.0	52.0	241.48	0.14	7.33	32.93	1770.86	...	13	0	2
10881	9.0	2.0	1.0	2.0	30.0	129.66	0.12	8.25	15.72	1069.68	...	5	0	2
10882	42.0	4.0	1.0	2.0	103.0	519.57	0.04	26.40	19.68	13716.72	...	29	1	10
10883	10.0	1.0	1.0	1.0	36.0	147.15	0.12	8.44	17.44	1241.57	...	6	0	2
10884	19.0	3.0	1.0	1.0	58.0	272.63	0.09	11.57	23.56	3154.67	...	13	0	2

5 rows × 22 columns

[40]:
```
data.sample(10) #shows random rows (sample(number_of_rows))
```

[40]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBl
8620	26.0	2.0	1.0	2.0	89.0	456.51	0.12	8.02	56.92	3661.56	...	18	0	
5410	51.0	7.0	1.0	4.0	150.0	792.81	0.04	26.27	30.18	20829.29	...	46	0	
2611	9.0	1.0	1.0	1.0	0.0	0.00	0.00	0.00	0.00	0.00	...	0	0	
4611	32.0	6.0	1.0	5.0	75.0	343.87	0.10	9.90	34.73	3404.33	...	27	0	
4617	38.0	5.0	1.0	5.0	66.0	313.82	0.08	12.92	24.28	4055.55	...	30	0	
2874	7.0	1.0	1.0	1.0	23.0	82.45	0.25	4.00	20.61	329.82	...	5	0	
596	23.0	3.0	1.0	3.0	96.0	461.51	0.11	9.00	51.28	4153.55	...	16	0	
6931	35.0	9.0	1.0	9.0	0.0	0.00	0.00	0.00	0.00	0.00	...	0	0	
3891	19.0	6.0	1.0	6.0	58.0	232.00	0.06	16.67	13.92	3866.67	...	12	0	
3196	168.0	44.0	42.0	34.0	540.0	3384.06	0.02	40.68	83.20	137649.97	...	120	15	

10 rows × 22 columns

[41]:
```
data.shape #shows the number of rows and columns
```

[41]:
```
(10885, 22)
```

[42]:
```
data.describe() #shows simple statistics (min, max, mean, etc.)
```

[42]:

	loc	v(g)	ev(g)	iv(g)	n	v	l
count	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000
mean	42.016178	6.348590	3.401047	4.001599	114.389738	673.758017	0.135335
std	76.593332	13.019695	6.771869	9.116889	249.502091	1938.856196	0.160538
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000
25%	11.000000	2.000000	1.000000	1.000000	14.000000	48.430000	0.030000
50%	23.000000	3.000000	1.000000	2.000000	49.000000	217.130000	0.080000
75%	46.000000	7.000000	3.000000	4.000000	119.000000	621.480000	0.160000
max	3442.000000	470.000000	165.000000	402.000000	8441.000000	80843.080000	1.300000

[43]:
```
defects_true_false = data.groupby('defects')['b'].apply(lambda x: x.count()) #defect rates (true/
print('False : ' , defects_true_false[0])
print('True : ' , defects_true_false[1])
```

```
False :  8779
True :  2106
```

-> Histogram

[ ]:
```
%pip install --upgrade nbformat
#%pip install micropip
#micropip.install(..., keep_going=True)
trace = go.Histogram(
    x = data.defects,
    opacity = 0.75,
    name = "Defects",
    marker = dict(color = 'green'))

hist_data = [trace]
hist_layout = go.Layout(barmode='overlay',
                   title = 'Defects',
                   xaxis = dict(title = 'True - False'),
                   yaxis = dict(title = 'Frequency'),
)
fig = go.Figure(data = hist_data, layout = hist_layout)
iplot(fig)
```

-> Covariance

Covariance is a measure of the directional relationship between the returns on two risky assets. A positive covariance means that asset returns move together while a negative covariance means returns move inversely.

[44]:
```
data.corr() #shows coveriance matrix
```

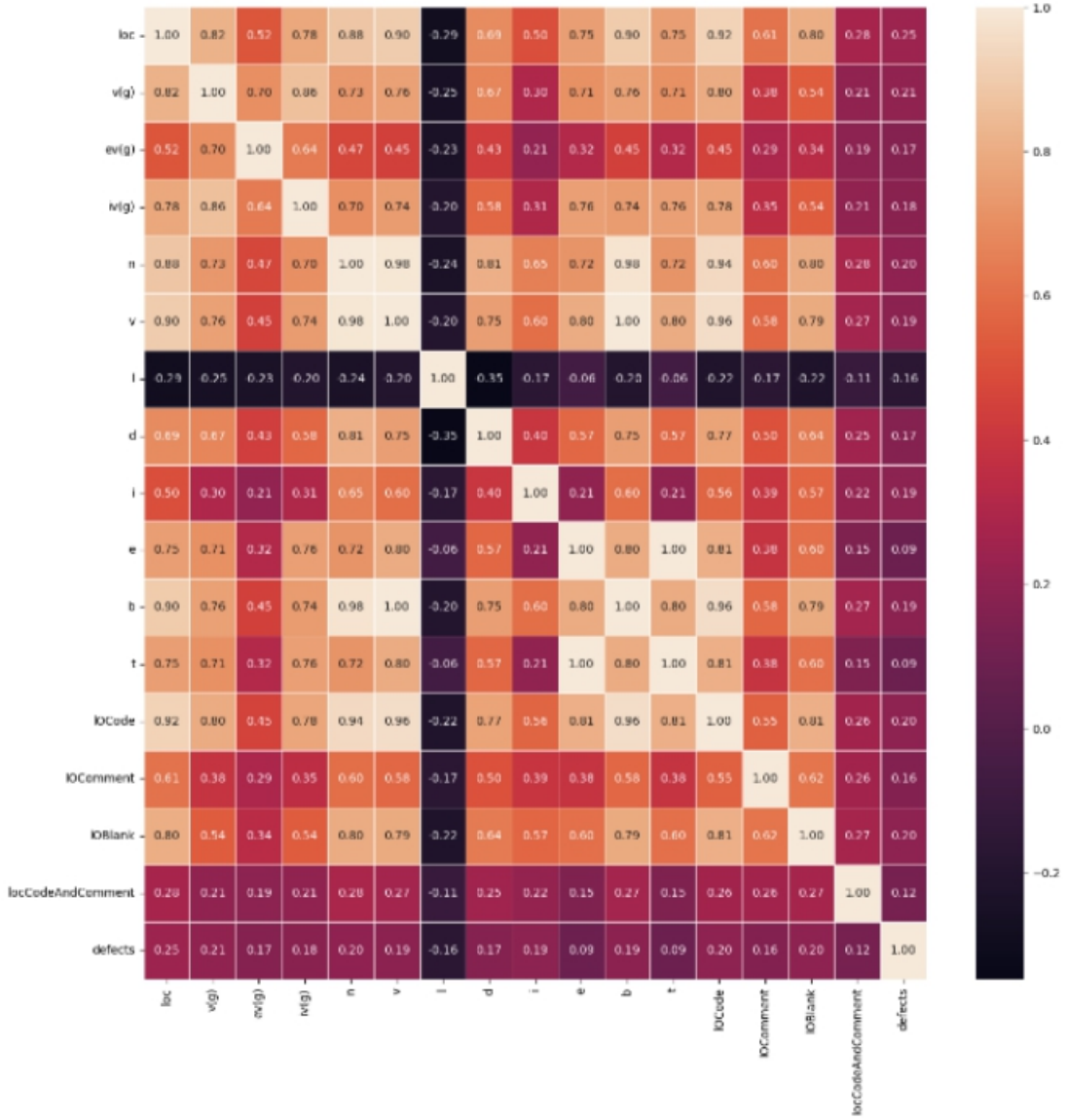
[44]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d
loc	1.000000	0.817757	0.517551	0.784057	0.881795	0.900293	-0.286587	0.689543
v(g)	0.817757	1.000000	0.701710	0.859590	0.730781	0.759881	-0.252902	0.669057
ev(g)	0.517551	0.701710	1.000000	0.639574	0.465992	0.445902	-0.233982	0.434009
iv(g)	0.784057	0.859590	0.639574	1.000000	0.702415	0.743193	-0.197736	0.575369
n	0.881795	0.730781	0.465992	0.702415	1.000000	0.984276	-0.240749	0.808113
v	0.900293	0.759881	0.445902	0.743193	0.984276	1.000000	-0.198104	0.752206
l	-0.286587	-0.252902	-0.233982	-0.197736	-0.240749	-0.198104	1.000000	-0.347215
d	0.689543	0.669057	0.434009	0.575369	0.808113	0.752206	-0.347215	1.000000
i	0.499946	0.303031	0.213211	0.309717	0.651209	0.598743	-0.166801	0.398162
e	0.750564	0.709501	0.315538	0.757702	0.716536	0.800000	-0.062026	0.574298
b	0.899965	0.759635	0.445693	0.743013	0.983938	0.999696	-0.196147	0.751835
t	0.750564	0.709501	0.315538	0.757702	0.716536	0.800000	-0.062026	0.574298
lOCode	0.921918	0.799915	0.454604	0.775873	0.944383	0.962078	-0.218373	0.768188
lOComment	0.612858	0.384506	0.294208	0.351583	0.596374	0.576844	-0.165885	0.502121
lOBlank	0.803573	0.538366	0.338243	0.541296	0.798561	0.792330	-0.223670	0.637211
locCodeAndComment	0.278119	0.209811	0.190911	0.207028	0.284391	0.266537	-0.106117	0.253793
defects	0.245388	0.208644	0.172973	0.181984	0.204143	0.189136	-0.164917	0.169629

-> Heatmap

[45]:

```
f,ax = plt.subplots(figsize = (15, 15))
sns.heatmap(data.corr(), annot = True, linewidths = .5, fmt = '.2f')
plt.show()
```



The light color in the heat map indicates that the covariance is high. (Ex. "v-b" , "v-n", etc.)

The dark color in the heat map indicates that the covariance is low. (Ex. "loc-l" , "l-d", etc.)

-> Scatter Plot

[ ]:
```
trace = go.Scatter(
    x = data.v,
    y = data.b,
    mode = "markers",
    name = "Volume - Bug",
    marker = dict(color = 'darkblue'),
    text = "Bug (b)")

scatter_data = [trace]
scatter_layout = dict(title = 'Volume - Bug',
              xaxis = dict(title = 'Volume', ticklen = 5),
              yaxis = dict(title = 'Bug' , ticklen = 5),
             )
fig = dict(data = scatter_data, layout = scatter_layout)
iplot(fig)

#two attributes with high correlation v-b > just about 1
```

-> Data Preprocessing

[46]:
```
data.isnull().sum() #shows how many of the null
```

[46]:
```
loc                 0
v(g)                0
ev(g)               0
iv(g)               0
n                   0
v                   0
l                   0
d                   0
i                   0
e                   0
b                   0
t                   0
lOCode              0
lOComment           0
lOBlank             0
locCodeAndComment   0
uniq_Op             0
uniq_Opnd           0
total_Op            0
total_Opnd          0
branchCount         0
defects             0
dtype: int64
```

No missing value.

No data cleaning needed because the data is all important.

-> Outlier Detection (Box Plot)

[ ]:
```
trace1 = go.Box(
    x = data.uniq_Op,
    name = 'Unique Operators',
    marker = dict(color = 'blue')
    )
box_data = [trace1]
iplot(box_data)
```

Showing all information when clicking on plot (min, max, q1, q2, etc.).

-> Feature Extraction

[47]:
```
def evaluation_control(data):
    evaluation = (data.n < 300) & (data.v < 1000 ) & (data.d < 50) & (data.e < 500000) & (data.t
    data['complexityEvaluation'] = pd.DataFrame(evaluation)
    data['complexityEvaluation'] = ['Succesful' if evaluation == True else 'Redesign' for evaluat
```

[48]:
```
evaluation_control(data)
data
```

[48]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOComment	lOBlank	loc
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	10	8	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	29	28	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	1	6	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10880	18.0	4.0	1.0	4.0	52.0	241.48	0.14	7.33	32.93	1770.86	...	0	2	
10881	9.0	2.0	1.0	2.0	30.0	129.66	0.12	8.25	15.72	1069.68	...	0	2	
10882	42.0	4.0	1.0	2.0	103.0	519.57	0.04	26.40	19.68	13716.72	...	1	10	
10883	10.0	1.0	1.0	1.0	36.0	147.15	0.12	8.44	17.44	1241.57	...	0	2	
10884	19.0	3.0	1.0	1.0	58.0	272.63	0.09	11.57	23.56	3154.67	...	0	2	

10885 rows × 23 columns

[49]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   loc                   10885 non-null  float64
 1   v(g)                  10885 non-null  float64
 2   ev(g)                 10885 non-null  float64
 3   iv(g)                 10885 non-null  float64
 4   n                     10885 non-null  float64
 5   v                     10885 non-null  float64
 6   l                     10885 non-null  float64
 7   d                     10885 non-null  float64
 8   i                     10885 non-null  float64
 9   e                     10885 non-null  float64
 10  b                     10885 non-null  float64
 11  t                     10885 non-null  float64
 12  lOCode                10885 non-null  int64
 13  lOComment             10885 non-null  int64
 14  lOBlank               10885 non-null  int64
 15  locCodeAndComment     10885 non-null  int64
 16  uniq_Op               10885 non-null  object
 17  uniq_Opnd             10885 non-null  object
 18  total_Op              10885 non-null  object
 19  total_Opnd            10885 non-null  object
 20  branchCount           10885 non-null  object
 21  defects               10885 non-null  bool
 22  complexityEvaluation  10885 non-null  object
dtypes: bool(1), float64(12), int64(4), object(6)
memory usage: 1.6+ MB
```

[50]:
```
data.groupby("complexityEvaluation").size() #complexityEvalution rates (Succesfull/redisgn)
```

[50]:
```
complexityEvaluation
Redesign     1725
Succesful    9160
dtype: int64
```

[ ]:
```
# Histogram
trace = go.Histogram(
    x = data.complexityEvaluation,
    opacity = 0.75,
    name = 'Complexity Evaluation',
    marker = dict(color = 'darkorange')
)
hist_data = [trace]
hist_layout = go.Layout(barmode='overlay',
                   title = 'Complexity Evaluation',
                   xaxis = dict(title = 'Succesful - Redesign'),
                   yaxis = dict(title = 'Frequency')
)
fig = go.Figure(data = hist_data, layout = hist_layout)
iplot(fig)
```

-> Data Normalization (Min-Max Normalization)

[51]:
```
from sklearn import preprocessing

scale_v = data[['v']]
scale_b = data[['b']]

minmax_scaler = preprocessing.MinMaxScaler()

v_scaled = minmax_scaler.fit_transform(scale_v)
b_scaled = minmax_scaler.fit_transform(scale_b)

data['v_ScaledUp'] = pd.DataFrame(v_scaled)
data['b_ScaledUp'] = pd.DataFrame(b_scaled)

data
```

[51]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	locCodeAndComment	uniq
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	1	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	2	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
10880	18.0	4.0	1.0	4.0	52.0	241.48	0.14	7.33	32.93	1770.86	...	0	
10881	9.0	2.0	1.0	2.0	30.0	129.66	0.12	8.25	15.72	1069.68	...	0	
10882	42.0	4.0	1.0	2.0	103.0	519.57	0.04	26.40	19.68	13716.72	...	0	
10883	10.0	1.0	1.0	1.0	36.0	147.15	0.12	8.44	17.44	1241.57	...	0	
10884	19.0	3.0	1.0	1.0	58.0	272.63	0.09	11.57	23.56	3154.67	...	1	

10885 rows × 25 columns

[52]:
```
scaled_data = pd.concat([data.v , data.b , data.v_ScaledUp , data.b_ScaledUp], axis=1)
scaled_data
```

[52]:

	v	b	v_ScaledUp	b_ScaledUp
0	1.30	1.30	0.000016	0.048237
1	1.00	1.00	0.000012	0.037106
2	1134.13	0.38	0.014029	0.014100
3	4348.76	1.45	0.053793	0.053803
4	599.12	0.20	0.007411	0.007421
...	...	...	...	...
10880	241.48	0.08	0.002987	0.002968
10881	129.66	0.04	0.001604	0.001484
10882	519.57	0.17	0.006427	0.006308
10883	147.15	0.05	0.001820	0.001855
10884	272.63	0.09	0.003372	0.003340

10885 rows × 4 columns

-> Model Selection

1. Naive Bayes

[53]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 25 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   loc                   10885 non-null  float64
 1   v(g)                  10885 non-null  float64
 2   ev(g)                 10885 non-null  float64
 3   iv(g)                 10885 non-null  float64
 4   n                     10885 non-null  float64
 5   v                     10885 non-null  float64
 6   l                     10885 non-null  float64
 7   d                     10885 non-null  float64
 8   i                     10885 non-null  float64
 9   e                     10885 non-null  float64
 10  b                     10885 non-null  float64
 11  t                     10885 non-null  float64
 12  lOCode                10885 non-null  int64
 13  lOComment             10885 non-null  int64
 14  lOBlank               10885 non-null  int64
 15  locCodeAndComment     10885 non-null  int64
 16  uniq_Op               10885 non-null  object
 17  uniq_Opnd             10885 non-null  object
 18  total_Op              10885 non-null  object
 19  total_Opnd            10885 non-null  object
 20  branchCount           10885 non-null  object
 21  defects               10885 non-null  bool
 22  complexityEvaluation  10885 non-null  object
 23  v_ScaledUp            10885 non-null  float64
 24  b_ScaledUp            10885 non-null  float64
dtypes: bool(1), float64(14), int64(4), object(6)
memory usage: 1.8+ MB
```

[54]:
```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn import model_selection

X = data.iloc[:, :-10].values  #Select related attribute values for selection
Y = data.complexityEvaluation.values   #Select classification attribute values
```

[55]:
```
Y
```

[55]:
```
array(['Succesful', 'Succesful', 'Redesign', ..., 'Succesful',
       'Succesful', 'Succesful'], dtype=object)
```

[56]:
```
#Parsing selection and verification datasets
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size = v
```

[57]:
```
#Creation of Naive Bayes model
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

[58]:
```
Calculation of ACC value by K-fold cross validation of NB model
coring = 'accuracy'
fold = model_selection.KFold(n_splits = 10, random_state = seed, shuffle = True)
/_results = model_selection.cross_val_score(model, X_train, Y_train, cv = kfold, scoring = scoring)
```

[59]:
```
cv_results
```

[59]:
```
array([0.97818599, 0.98507463, 0.98277842, 0.97703789, 0.98277842,
       0.97474168, 0.98507463, 0.97474168, 0.96896552, 0.98390805])
```

[60]:
```
msg = "Mean : %f - Std : (%f)" % (cv_results.mean(), cv_results.std())
msg
```

[60]:
```
'Mean : 0.979329 - Std : (0.005166)'
```

[61]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

#Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))
```

```
              precision    recall  f1-score   support

    Redesign       0.93      0.94      0.94       319
   Succesful       0.99      0.99      0.99      1858

    accuracy                           0.98      2177
   macro avg       0.96      0.97      0.96      2177
weighted avg       0.98      0.98      0.98      2177

[[ 301   18]
 [  22 1836]]
ACC:  0.9816260909508497
```

2. Decision Tree

[62]:
```
from sklearn import tree
```

[63]:
```
model = tree.DecisionTreeClassifier()
```

[64]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

#Summary of the predictions made by the classifier
print("Decision Tree Algorithm")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))
```

```
Decision Tree Algorithm
              precision    recall  f1-score   support

    Redesign       1.00      1.00      1.00       319
   Succesful       1.00      1.00      1.00      1858

    accuracy                           1.00      2177
   macro avg       1.00      1.00      1.00      2177
weighted avg       1.00      1.00      1.00      2177

[[ 319    0]
 [   1 1857]]
ACC:  0.9995406522737712
```

3. SVM

[80]:
```
from sklearn import svm
```

[81]:
```
model = svm.SVC(kernel='linear', C=0.01)
```

[82]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

#Summary of the predictions made by the classifier
print("SVM Algorithm")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))
```

```
SVM Algorithm
              precision    recall  f1-score   support

    Redesign       1.00      0.97      0.99       319
   Succesful       1.00      1.00      1.00      1858

    accuracy                           1.00      2177
   macro avg       1.00      0.99      0.99      2177
weighted avg       1.00      1.00      1.00      2177

[[ 311    8]
 [   0 1858]]
ACC:  0.99632521819017
```

4. Random Forest

[68]:
```
from sklearn.ensemble import RandomForestClassifier
```

[69]:
```
model=RandomForestClassifier(n_estimators=100)
```

```
Random Forest Algorithm
              precision    recall  f1-score   support

    Redesign       1.00      0.99      1.00       319
   Succesful       1.00      1.00      1.00      1858

    accuracy                           1.00      2177
   macro avg       1.00      1.00      1.00      2177
weighted avg       1.00      1.00      1.00      2177

[[ 316    3]
 [   0 1858]]
ACC:  0.9986219568213137
```

5. KNN

[73]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[74]:
```
model = KNeighborsClassifier(n_neighbors=5)
```

[75]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

#Summary of the predictions made by the classifier
print("K-Nearest Neighbors Algorithm")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))
```

```
K-Nearest Neighbors Algorithm
              precision    recall  f1-score   support

    Redesign       0.98      0.96      0.97       319
   Succesful       0.99      1.00      0.99      1858

    accuracy                           0.99      2177
   macro avg       0.99      0.98      0.98      2177
weighted avg       0.99      0.99      0.99      2177

[[ 306   13]
 [   7 1851]]
ACC:  0.9908130454754249
```

6. Logistic Regression

[76]:
```
from sklearn.linear_model import LogisticRegression
```

[77]:
```
model = LogisticRegression()
```

[84]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

#Summary of the predictions made by the classifier
print("Logistic Regression Algorithm")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))
```

```
Logistic Regression Algorithm
              precision    recall  f1-score   support

    Redesign       1.00      0.97      0.99       319
   Succesful       1.00      1.00      1.00      1858

    accuracy                           1.00      2177
   macro avg       1.00      0.99      0.99      2177
weighted avg       1.00      1.00      1.00      2177

[[ 311    8]
 [   0 1858]]
ACC:  0.99632521819017
```

REFERENCES

[1] Md. Razu Ahmed, Md. Asraf Ali, Md Fahad Bin Zamal, Nasim Ahmed, “The Impact of Software Fault Prediction in Real-World Application: An Automated Approach for Software Engineering”.

[2] Bartłomiej Wójcicki, Robert Dąbrowski, “Applying Machine Learning to Software Fault Prediction, Institute of Informatics, University of Warsaw”.

[3] Tim Menzies, Justin DiStefano, Andres Orrego, Robert (Mike) Chapman, “Assessing Predictors of Software Defects”.

[4] Manzura Jorayeva, Akhan Akbulut, Cagatay Catal and Alok Mishra, “Machine Learning-Based Software Defect Prediction for Mobile Applications: A Systematic Literature Review”.

[5] Fatih Yucalara , Akin Ozcifta , Emin Borandaga, Deniz Kilinc, “Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability”.

[6] Mitt Shah, Nandit Pujara, “Software Defects Prediction Using Machine Learning”.

[7] Revoori Veeharika Reddy, Nagella Kedharnath, Mandi Akif Hussain, S. Vidya, “Software Defect Estimation using Machine Learning Algorithms”.

[8] Jaswitha Abbineni, Ooha Thalluri, “Software Defect Detection Using Machine Learning Techniques”.

[9] Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan, Fatima Alsarayrah, “Software Bug Prediction using Machine Learning Approach”.

[10] Marwa Assim, Qasem Obeidat, Mustafa Hammad, “Software Defects Prediction using Machine Learning Algorithms”.

[11] Burcu Yalçıner, Merve Özdeş, “Software Defect Estimation Using Machine Learning Algorithms”.

[12] Awni Hammouri, Mustafa Hammad, Mohammad M Alnabhan, “Software Bug Prediction using Machine Learning Approach”.

[13] M. Jureczko and L. Madeyski, “Towards identifying software project clusters with regard to defect prediction,” in Proceedings of the 6th International Conference on Predictive Models in Software Engineering - PROMISE ’10, 2010, p. 1.

[14] H. Tanwar and M. Kakkar, “A Review of Software Defect Prediction Models,” Springer, Singapore, 2019, pp. 89–97.

[15] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” Appl. Soft Comput., vol. 27, pp. 504–518, Feb. 2015.

[16] Catal, C.; Diri, B. A systematic review of software fault prediction studies. Expert Syst. Appl. 2009, 36, 7346–7354.

[17] Malhotra, R.; Jain, A. Software fault prediction for object-oriented systems: A systematic literature review. ACMSIGSOFT Softw. Eng. 2011, 36, 1–6.

[18] Malhotra, R. A systematic review of machine learning techniques for software fault prediction. Appl. Soft Comput. 2015, 27,504–518.

[19] Radjenovic, D.; Heriko, M. Software fault prediction metrics: A systematic literature review. Inf. Softw. Technol. 2013, 55,1397–1418.

[20] Misirli, A.T.; Bener, A.B. A mapping study on Bayesian networks for software quality prediction. In Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, Hyderabad, India, 3 June 2014; pp. 7–11.

[21] Murillo-Morera, J.; Quesada-López, C.; Jenkins, M. Software Fault Prediction: A Systematic Mapping Study; CIbSE: London, UK, 2015; p. 446.

[22] Özakıncı, R.; Tarhan, A. Early software defect prediction: A systematic map and review. J. Syst. Softw. 2018, 144, 216–239.

[23] Son, L.H.; Pritam, N.; Khari, M.; Kumar, R.; Phuong, P.T.M.; Thong, P.H. Empirical Study of Software Defect Prediction: A Systematic Mapping. Symmetry 2019, 11, 212.

[24] Najm, A.; Zakrani, A.; Marzak, A. Decision Trees Based Software Development Effort Estimation: A Systematic Mapping Study. In Proceedings of the 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), Agadir, Morocco, 22–24 July 2019.

[25] Alsolai, H.; Roper, M. A systematic literature review of machine learning techniques for software maintainability prediction. Inf. Softw. Technol. 2020, 119, 106214.