

Analysis of software fault prediction using Machine Learning Algorithm

by

Dipanker Shaha

16201104

Md Mamun Uddin

16201088

Akash Chandra Paul

16301171

Bishal Roy

16201054

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
January 2021

© 2021. Brac University
All rights reserved.

Declaration

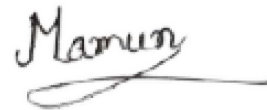
It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:



Dipanker Shaha
16201104



Md Mamun Uddin
16201088



Akash Chandra Paul
16301171



Bishal Roy
16201054

Approval

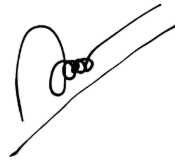
The thesis titled “Analysis of software fault prediction using Machine Learning Algorithm” submitted by

1. Dipanker Shaha (16201104)
2. Md Mamun Uddin (16201088)
3. Akash Chandra Paul (16301171)
4. Bishal Roy (16201054)

Of Fall, 2020 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 8, 2021.

Examining Committee:

Supervisor:
(Member)



DR. Muhammad Iqbal Hossain
Assistant Professor
Department of Computer Science and Engineering
BRAC University

Program Coordinator:
(Member)



DR. Md. Golam Rabiul Alam
Associate Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

DR. Mahbubul Alam Majumdar
Professor
Department of Computer Science and Engineering
BRAC University

Abstract

Today software performs a requisite role in our daily lives. Software's complexity keeps growing. The increasing complexity of any software system making it very difficult to improve its quality. The performance of the software depends on its bug-free operation. The main goal of developing any software is to identify and resolve bugs that may be required in various situations before the schedule is established. Software fault prediction is a way that seeks to classify fault-prone software modules by using specific underlying characteristics of software project before actual testing tends to start. Separate researchers have previously examined several classification ways for the prediction of software bugs. The output of various techniques varies from software to software, and no one technique is always successful throughout all fields. Nowadays, machine learning is widely using in software defect detection. We can save our valuable time and reduce costs by using machine learning algorithms in fault prediction. There are many machine learning algorithms used for the prediction of defects in software systems. Although most of the work is available for software systems classification, either fault-prone or non-fault prone, little attempt has been done to predict the fault ensemble techniques. We have set up a strategy in this paper to use some machine learning algorithms and Boosting Algorithms to analyze their performance on the promise dataset and unified Dataset. We have selected six machine learning algorithms, and they are KNN, Random Forest, Decision Tree, MLP, SVM, Naive Bayes, Logistic Regression and two Boosting Algorithms such as XGBoost and AdaBoost Algorithm. We applied those algorithms to our two types of datasets, such as the Unified Dataset and Promise Dataset (JM1, PC1, CM1). We have decided to analyze the best machine learning algorithm based on their maximum accuracy. We will ensure the best machine learning algorithm analysis for the unified and promise dataset.

Keywords: Software fault prediction; Machine learning; Data protection; XG-Boost; Support Vector Machine; Logistic regression; pre-process; AdaBoost.

Dedication

We would like to dedicate our thesis report to our parents for their constant support. Special gratitude towards our close friends and our Supervisor DR. Muhammad Iqbal Hossain.

Acknowledgement

Firstly, all praise to the almighty for whom our thesis has been completed without any significant interruption.

Secondly, to our supervisor Dr Muhammad Iqbal Hossain, Assistant Professor sir for his kind support and advice in our work. He helped us whenever we needed help.

Finally to our parents without their support, it may not be possible. We are now on the verge of our graduation with their kind encouragement and prayer.

Table of Contents

Declaration	i
Approval	ii
Abstract	iii
Dedication	iv
Acknowledgment	v
Table of Contents	vi
List of Figures	viii
List of Tables	1
1 Introduction	2
1.1 Motivation	3
1.2 Problem Statement	3
1.3 Objective and Contribution	4
1.4 Thesis Orientation	5
2 Related Work	6
2.1 Background	6
2.2 Literature Review	6
2.3 Algorithms	8
2.3.1 K-Nearest Neighbor Algorithm	9
2.3.2 Random Forest	9
2.3.3 SVM (Support Vector Machine)	10
2.3.4 Naive Bayes Algorithm	11
2.3.5 Decision Tree	12
2.3.6 MLP Algorithm	12
2.3.7 XG Boost	13
2.3.8 AdaBoost	13
3 Proposed Model	15
3.1 Dataset description	15
3.1.1 Data preprocessing	22
3.1.2 Feature Selection	23
3.2 Model Description	23

4	Experimentation	25
5	Result Analysis (ROC)	30
5.1	Roc Curve	30
5.1.1	CM1 ROC curve	30
5.1.2	PC1 ROC curve	33
5.1.3	JM1 ROC curve	36
5.1.4	Unified Dataset ROC curve	39
6	Conclusion and Future Plan	45
	Bibliography	47

List of Figures

2.1	K-NN Algorithms	9
2.2	Random Forest Algorithms	10
2.3	SVM Algorithms	11
2.4	Decision Tree Algorithms	12
2.5	MLP Algorithms	13
2.6	Adaboost Algorithms	14
3.1	Histogram of Defect's Frequency for CM1 Dataset	17
3.2	Heatmap of input data for CM1 dataset	17
3.3	Histogram of Defect's Frequency for PC1 Dataset	18
3.4	Heatmap of input data for PC1 dataset	18
3.5	Histogram of Defect's Frequency for JM1 Dataset	19
3.6	Heatmap of input data for JM1 dataset	19
3.7	Histogram of Defect's Frequency for Unified Dataset	22
3.8	Heatmap of the input data for unified dataset	22
3.9	Workflow of the System	23
5.1	Random Forest and Decision Tree ROC Curve	31
5.2	KNN and MLP ROC Curve	31
5.3	Naïve Bayes and SVM ROC curve	32
5.4	XGBoost and AdaBoost ROC Curve	32
5.5	Combined ROC curve of all algorithms	33
5.6	Random Forest and Decision Tree ROC Curve	34
5.7	KNN and MLP ROC Curve	34
5.8	Naïve Bayes and SVM ROC curve	35
5.9	XGBoost and AdaBoost ROC Curve	35
5.10	Combined ROC curve of all algorithms	36
5.11	Random Forest and Decision Tree ROC Curve	37
5.12	KNN and MLP ROC curve	37
5.13	Naïve Bayes and SVM ROC Curve	38
5.14	XGBoost and AdaBoost ROC Curve	38
5.15	Combined ROC curve of all algorithms	39
5.16	Random Forest and Decision Tree ROC Curve	40
5.17	KNN and MLP ROC Curve	40
5.18	Naïve Bayes and SVM ROC Curve	41
5.19	XGBoost and AdaBoost ROC Curve	41
5.20	Combined ROC curve of all algorithms	42

List of Tables

3.1	Promise Dataset Attribute	15
3.2	Number of attributes	16
3.3	Number of Instance	16
3.4	Class Distribution	16
3.5	Unified dataset attributes	20
3.6	Unified dataset attributes	21
4.1	CM1 Experiemnts Result	26
4.2	PC1 Experiemnts Result	27
4.3	JM1 Experiemnts Result	28
4.4	Unified Dataset Experiemnts Result	29
5.1	accuracy table of all four datasets for all Eight algorithms	42
5.2	comparison of related paper accuracy with our accuracy	43
5.3	comparison of related paper AUC with ours AUC	44

Chapter 1

Introduction

Predicting software bug is a way that is used to improve software quality. Fault handling is a significant issue in the process of software advancement. The presence of severe flaws involves high potential risks that are caused by the project disruption. Besides, it similarly reduces the quality of the project also increases the expense of maintenance. Early detection of errors minimizes the time and cost of implementation. To achieve bug-free software is very tough because there may remain some masked flaws, even if everything is implemented carefully. The real challenge for software engineering is developing a software bug prediction model that could predict defective modules in the early stage. Fault prediction is an eventual development activity. Because early fault predicting improves user satisfaction and software performance, reducing the time and maintenance cost. Fault prediction offers the developers to concentrate on faulty modules and fix the problem. Many machine learning algorithms are used for predicting software bug to predict defects in the early stages. Many researchers have explored the vital connection between software metrics and fault proneness. Several studies have shown that machine learning algorithms are the most efficient way for predicting software bugs. Fault prediction is considered by using a machine-learning algorithm to classify flaws in the initial improvement period. K-nearest neighbour Algorithm, Logistic Regression, Multilayer Perceptron, Random Forest, Support Vector Machine, Decision Tree, Naive Bayes, and Boosting Algorithm are the most efficient machine-learning algorithm to predict early bug from software or dataset. However, some machine learning algorithms are not effective in detecting faults. Because some machine learning algorithms cannot predict the maximum accuracy of the dataset. It is essential to identify the best machine learning algorithm to get the maximum accuracy of a dataset and predict the fault as soon as possible. The software fault prediction analysis depends on machine learning algorithm accuracy, recall, precision, F-measures, and the classifiers' ROC curves.

1.1 Motivation

Software bugs are a critical issue in the software industry. Sometimes one small mistake leads to an enormous loss of money and time. The amount of time and money it takes to improve software after the implementation is much higher than during the SDLC process. If we study historical bugs of software and its cost, it will always motivate us to decrease this loss. In 1998, NASA's Climate Orbiter spacecraft lost in space because a bug failed to convert the English unit to metric [1]. It causes \$125 million. Then, Europe's satellite Ariane 5 used its predecessor working software, but its fast engines faced bugs, which is why the software tried to squeeze a 64-bit number into 16-bit space [2]. As a result, the satellite started self-destruction after thirty-six seconds. The losses of this project were \$8 billion. In 2004, the EDS buildup and It system for U.K.'s Child Support Agency [3]. This system overpays 1.9 million people, which causes \$7 billion. Soviet Gas Pipeline Explosion in 1982, the CIA made a conspiracy and stole software components of the Soviet Union's gas pipeline controlling software, which lead to a massive explosion [4]. In 2013, Mt. GOX said that a hacker hacked 850,000-bitcoins [5]. Furthermore, the Heathrow terminal 5 baggage handling system collapsed when handling the passenger's baggage [6].

In 10 days, overall, 42 thousand bags failed to travel with their owner. The mariner, one spacecraft in 1962, crashed because of a software error [7]. The error caused by the omission of a hyphen in the computing instructions only for guidance problems allowed them to send incorrect signals to the spacecraft. The project's losses amounted to \$18 million. In 1988, the Morris Worm program crashed thousands of computers because it attempted to make an error [8]. One of the most extensive software bug failures is the Patriot Missile error [9]. The attack on army barracks failed to be detected by the U.S. Patriot Missile system. The British Airways system failure occurred because of a system error, which implies that 100 flights cancelled and 200 delayed [10]. Moreover, the most recent virus ransomware attacks are also the most significant failure of software. We can understand that software defects are a very complex problem; that is why we should test our software as much as possible before launching the software [1].

1.2 Problem Statement

Several machine learning algorithms often use to identify faults in various sectors like software testing, detect diseases, and analyze big data Etc. A broad range of machine learning algorithms can automatically predict the fault proneness module—the Machine learning algorithm classified as classification. Many Machine Learning Algorithms will explain in this section. Image recognition is currently mainly focused on the Convolutional Neural Network (CNN) algorithm. CNN is also used to place and detect hidden faults in industrial objects and devices.

As an example of a machine learning technique, Vector Machine is a classifier for data unseen. It works by constructing an N-dimensional hyperplane and supports a Vector Machine for finding an optimal hyperplane that can separate the vector's clusters. For maximizing the margin, SVM modeling finds the oriented hyperplane. SVM may use a kernel function for mapping the data into various spaces to manage the nonlinear separator between points. The hyperplane is used for space separation. There are many classification trees in Random Forest Algorithm. The classification trees are acknowledged as decision trees. Each tree in the forest is put down to the input vector, where each tree provides a classification result. Tree voting for the class. Then most classification votes are determined by the forest.

Bootstrap aggregating is familiar as a bagging algorithm. It is a method that is frequently sampled from the dataset based on a uniform distribution of probability. Each bootstrap sample size and the original data are identical. For instance, it has the possibility that the sampling is done with replacement to happen many times during the corresponding training dataset. Artificial Neural Network is well popular lately, like Multilayer Perceptron (MLP). Any issues, like pattern recognition, could be solved with MLP. Maximum two secret layers included in the Multilayer Perceptron. This technique retrieves the closest stored example by using an entropic measure.

1.3 Objective and Contribution

We are working with machine learning algorithms, and we apply different algorithms in our historical bug dataset. We also worked with a very new dataset that is unified. It is a combination of well-known publicly available five promise datasets. Then we perform different preprocessing approaches to make our dataset usable for the algorithms. We know that the raw dataset is noisy and not in a state that can be used for any algorithms. We choose eight machine learning algorithms for each dataset, and after running those algorithms, we get accuracy for each algorithm. We compare these eight algorithms with accuracy and analysis, which algorithms give the highest accuracy. Lastly, we also make comparisons between our four datasets and their accuracy for each algorithm.

Software bugs are a common issue in today's world. No one could say that their software is 100% bug-free. Software quality depends on that issue. That is why, in this sector, extensive study is needed to improve software quality. Here we are giving a general guideline on software bug prediction. Which type of algorithms is used for which type of software bug prediction. Suppose there is any probability of software bugs when we are working on software. Our work will give an idea of a software buggy or not. Our research also assists other researchers in this area. Researchers can get an idea of which should work; we are analyzing the algorithms. If any researcher works with bug prediction without any prior knowledge, our paper will give them a general guideline. It is a crucial part of the research that most

researchers are misguided or work on the wrong path, but for them, this paper is a path for them to go ahead in this sector and find more critical issues of software bugs and resolve these issues.

1.4 Thesis Orientation

Chapter 1 - Introduction where motivation, problem statement, objectives and contribution are discussed.

Chapter 2 - Background where Literature Review and Related Machine Learning Algorithm has been discussed.

Chapter 3 - The proposed Model is organized with our promised dataset and unified dataset description, Pre-processing of those dataset, Feature selection, and Model Description.

Chapter 4 - Experimentation reflects all our dataset accuracy with the table using machine learning and boosting algorithms.

Chapter 5 - Result Analysis and Data Visualization section discussed the ROC curve of all machine learning algorithms and the combination of all algorithms with histogram and heatmap.

Chapter 6 - Conclusion and Future Works, where we have discussed our thesis work and our future work.

Bibliography - We have included several references in this thesis paper, mostly related to our work.

Chapter 2

Related Work

2.1 Background

Software fault is also called a defect issue where the expected and actual results do not match each other. It may also be a computer program error, flaw, failure, or error. Most bugs come off spontaneously from the developers' errors and failures.

The software fault prediction method intends to recognize fault-prone software modules using some software project's underlying properties before the actual software testing begins. It contributes to optimized costs and effort to achieve the desired software quality. When the software's size and complexity increase, it becomes more challenging to predict software flaws. A model may incorporate software to faulty and non-faulty prone modules, to maintain a high-quality software level. In the procedure of faulty and non-faulty prone, the prediction of defective-prone modules causes more costly and more time-consuming.

Bug prediction develops the software advancement method in terms of production expense, reduces the preservation period. The machine learning algorithm is used for predicting software bug and maintainability. In work done by S. Delphine et al. [11], they tried to use a model for this process and used several machine learning algorithms. Random forest algorithm is preferable and got the highest accuracy for their model. In work done by P. S. Bishnu et al. [12] they have judged the efficiency of the Quadtree-based K-Means clustering algorithm for predicting faulty software modules.

2.2 Literature Review

From the research paper [11], Software testing is an important software development process and software testing work as input for software fault prediction. Sometimes, the Naïve Bayes algorithm is used for software fault prediction and software per-

formance measurement. The alternatives bayesian works to create a network using fewer nodes. Here they compared 15 bayesian networks using ROC curves, and they also used H-measure.

In this paper [12],Quad tree-based K-Means algorithms use for predicting bugs in program modules. They compare the original k-mean outcome with the Quad tree-based K-Means algorithms. Finally, they compare these algorithms' error rates with other existing algorithms and take the best one.

This paper [13],This paper is based on predicting bugs In the software source code using supervised machine learning algorithms. This bug prediction model can be applied in any SDLC model. Here they used decision trees, Logistic regression and naïve Bayes to build their model to use historical datasets. Finally, they used random forest and got the highest accuracy. K-fold validation is used in a dataset to reduce biases

This paper [14], works with a new methodology, which is fault injection. For both automatic test case generation and fault detection, this method is used. Here LBT testing is automated black box requirement testing, and this machine learning approach integrates with L* algorithm. This approach's goal can divide into three phases: automated test case generation, based on formal requirement model construction of automatic verdict, and automated fault injection.

This paper [15], is based on the study of the failure of traditional defect prediction design. CNN used for the defect prediction. First, they find token vectors. Then they convert these token vectors into numerical vectors. Next, they use this numerical vector on CNN, which helps CNN learn both the program's semantic and structural features. Finally, they combined this feature with hand-crafted features to get better software prediction.

In this paper [16], for the improvement of open source software activities, they proposed a deep learning-based approach. The users and the members of the project detect bugs in software using bug tracking systems. However, there is one problem that no software tool still cannot analyze this data of software bug tracking systems. They work on this issue to work on software bug tracking systems using deep learning to learn more about this fault data.

According to paper [17],], SFP is a process of machine learning used for software testing, managing an efficient solution, and trained with faulty and non-faulty categories. It uses different types of non-faulty samples based on single class SVM for 100 times and ensures the model's real-time data. This model builds an efficient predator in software testing's early life that will be an SFP solution for software.

This paper [18] is written about the fault-prone prediction, which is efficient and accurate. To predict the statistical fault method, machine learning method, neural networking techniques, clustering technique. Here they use clustering techniques to improve the NASA metrics data program's scheduling and planning for cost avoidance after adequate verification. This clustering model gives better performance by finding faulty and non-faulty software product modules. They finally find the best model, which gives the best performance among them.

In this paper [19], They work with NASA and MDP datasets in this paper[19] and apply multi-classifiers to get the best results. To get the highest accuracy, they use Help Vector Machine, Naïve Bayes and Random Forest classifiers.

The comparative analysis between bagging, boosting, stacking, and foundation learning strategies is done in this paper [20]. For prediction, they use a promise dataset. In this paper, they find that random forest classifiers should be stacked with other classifiers to obtain a better prediction of faults. The performance of a

wide range of classification models within software defect prediction is benchmarked in the paper[21]. The NASA MDP data set was used in the same paper to find AUC for ten classification models.

The purpose of paper [21] intends to evaluate the software fault prediction capability in terms of precision, f-measure, accuracy, recall, and ROC curve.

The most frequently studied machine learning algorithms are analyzed in the paper [22]. For validation, they use the k-fold cross-validation technique. They use NASA's promised dataset repository in their paper.

2.3 Algorithms

At first, for predicting software fault using Machine Learning, we need a dataset. We managed a promised dataset, and we analyse each attribute of what they reflect. We started preprocessed our dataset. We know that our real-world raw data is not consistent. Sometimes there will be errors or null values. There also be sometimes unnecessary data which we do not want to use. So, we remove those unnecessary and null data from the dataset.

Similarly, we divided the data set into training and test dataset. Over here we can split our dataset for training, the one part of the dataset and rest of them are used of testing. Most of the time, 80% of data are used for the training dataset and 20%

dataset used for testing the trained algorithm. For testing and to train the dataset, we choose three algorithms to train using our dataset. They are-

1. K-nearest neighbour Algorithm (KNN)
2. Random Forest Algorithm
3. Support Vector Machine (SVM) Algorithm
4. Naive Bayes Algorithm
5. Decision Tree Algorithm
6. Multilayer Perceptron (MLP) Algorithm
7. XG Boost Algorithm
8. AdaBoost Algorithm

2.3.1 K-Nearest Neighbor Algorithm

KNN is a predictive classification method that intensively examined for four decades of pattern recognition. This algorithm has utilized to software flaws and revealed promising outcomes. It is a simple algorithm that is used for classification and regression problems. It works properly with multi-label classes. This algorithm needs no training before giving forecasts, and new data can be attached, which will not affect the accuracy of the algorithm.

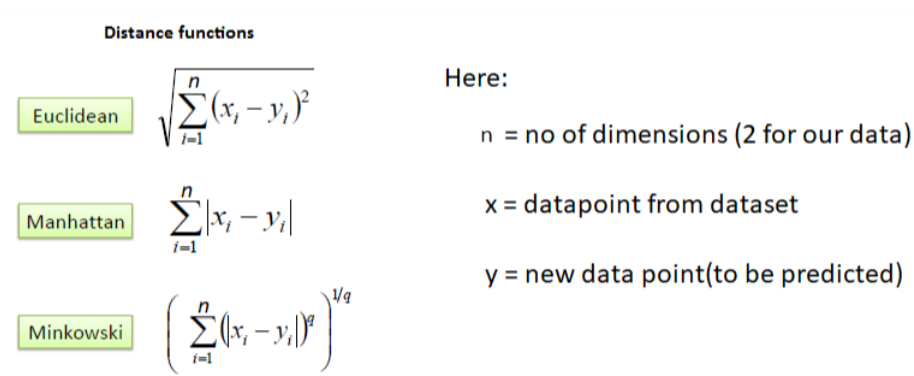


Figure 2.1: K-NN Algorithms

2.3.2 Random Forest

For classifying software defects, a comparative study of different classification methods was performed. It uses to provide a reliable forecast outcome. Its default hyperparameters deliver excellent outcomes and the method is excellent at avoiding overfitting. We are using on public NASA datasets of PROMISE repository.

For both classification and regression functions, the Random Forest Algorithm can be used. It allows maximum accuracy. The random forest classifier manages the missing values, and the precision of a significant proportion of the data is maintained. It can tackle with higher dimensionality a large data set.

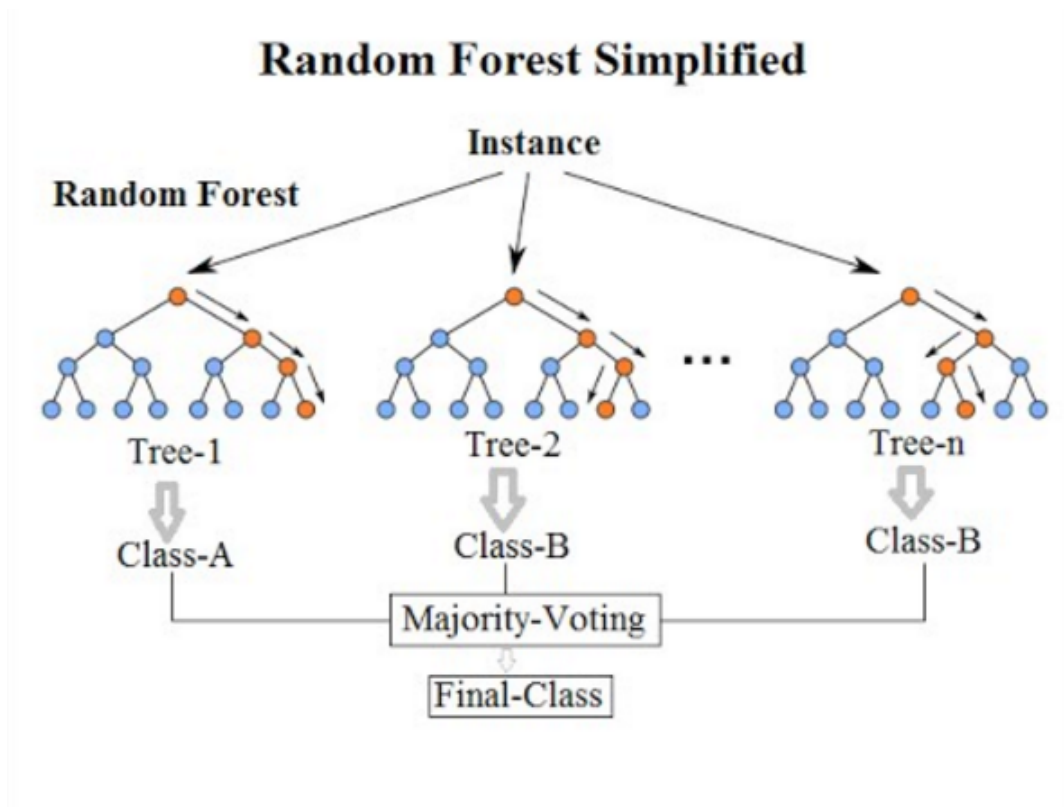


Figure 2.2: Random Forest Algorithms

In the random forest, we need to grow multiple trees in a model. Of all the other trees in the forest, the forest picks the classifications with the most votes and takes the average difference from different trees' output. In general, multiple trees were constructed by Random Forest and combined to gain more accurate results.

2.3.3 SVM (Support Vector Machine)

SVM is a machine learning algorithm used for issues with classification or regression and outlier detection. Its aim is to develop the best line or decision boundary. A hyperplane is the best line. The extreme points help to create that. The decision boundary maximises the range from the closest data points of all the classes. SVMs is unique from other classifier algorithms.

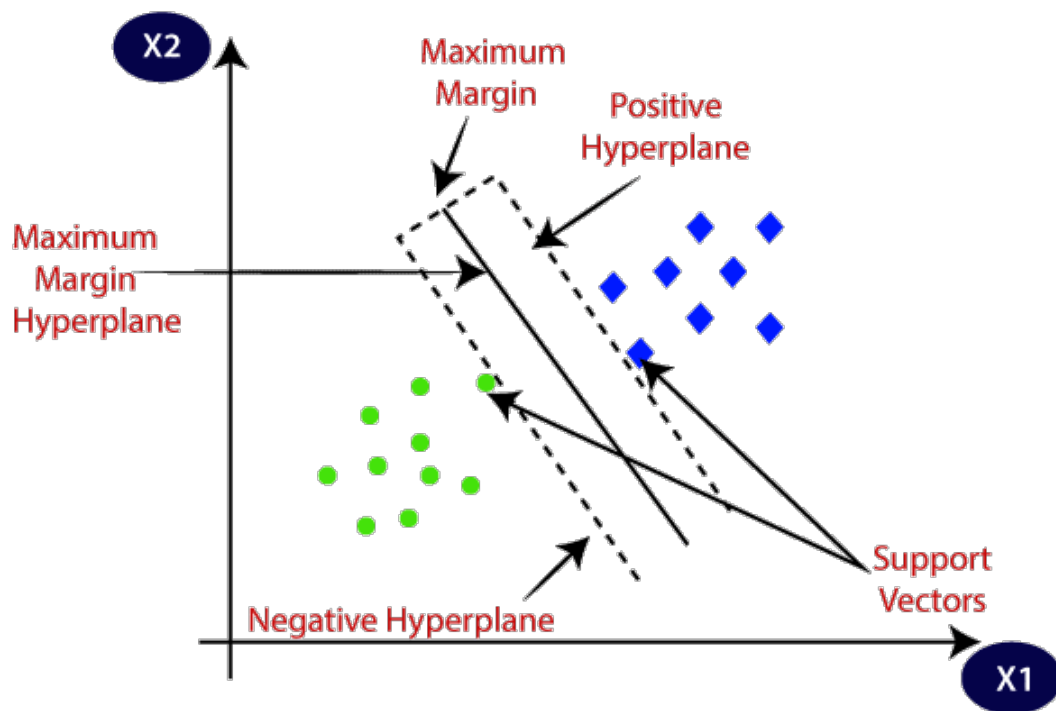


Figure 2.3: SVM Algorithms

2.3.4 Naive Bayes Algorithm

Naïve Bayes Classifier is an easy and most useful classifier algorithm that builds fast machine learning models. It can execute fast forecasts. It uses for text classification. Its uses for predicting the probability of various classes based on various attributes. The Naive Bayes model is easy to build and it runs best when it has a small training data set. For an extensive dataset, it may not measure better accuracy.

The Naive Bayes model is simple to create and especially helpful for extensive data sets. The Bayes theorem presents a way to measure posterior probability $P(c|x)$ from $P(c)$, $P(x)$, and $P(x|c)$. The equation has given below:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Likelihood
Class Prior Probability
↓
↓
Posterior Probability
Predictor Prior Probability

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

2.3.5 Decision Tree

Decision Tree is a supervised learning method but can typically use to overcome problems with classification and regression. In this algorithm, the data is continuously divided based on a certain parameter. There are two nodes in the decision tree, the Decision Node, and the Leaf Node. Decision Tree is utilized to create classification and regression models. It is applied to build data models. For the decision-making process, it will predict class labels. The decisions are focused on the characteristics of the initial data. It is a graphic representation to achieve all possible solutions based on conditions to a decision. It is shaped like a tree. The parent node is root node and growing on small branches.

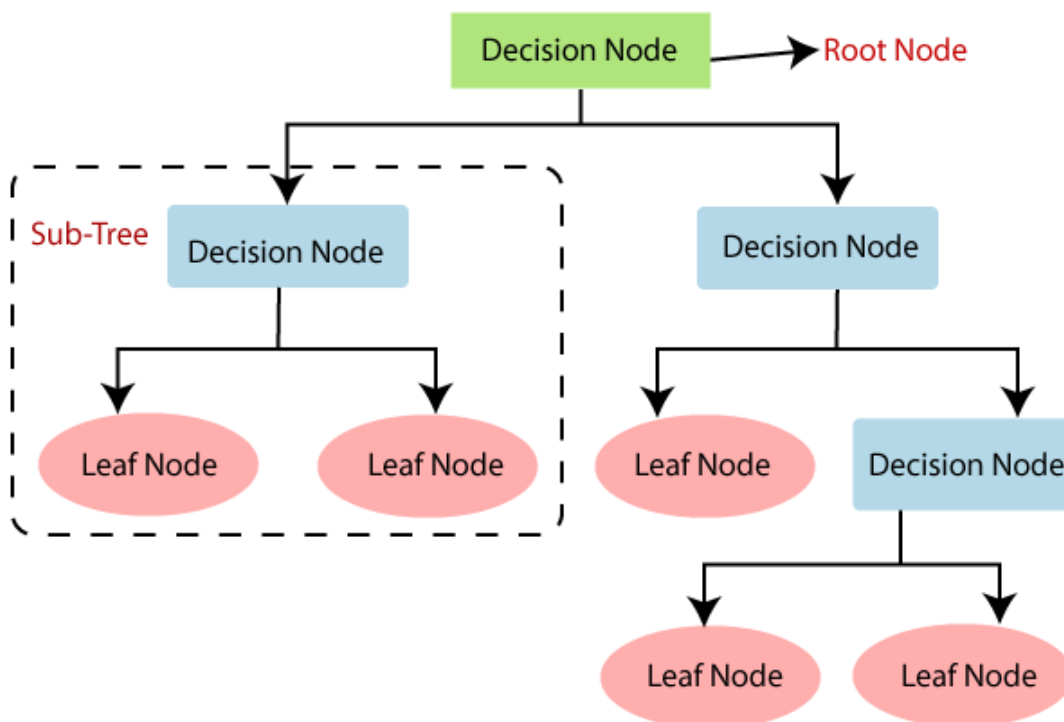


Figure 2.4: Decision Tree Algorithms

2.3.6 MLP Algorithm

MLP is a supplement of neural network. It consists of three layers input, hidden and output layer. First of all, input layer accepts the input signal for processing. The output layer performs the required task, such as prediction and classification. Output and hidden layer uses nonlinear activation function.

Like a feed-forward network in an MLP algorithm, data flows in the forward direction

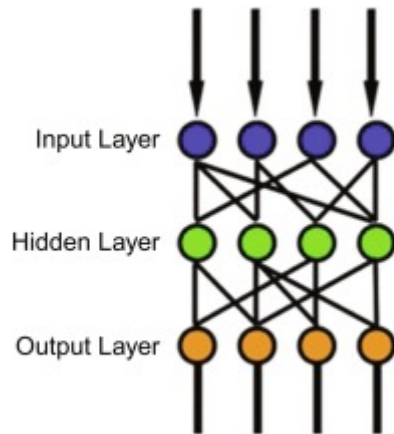


Figure 2.5: MLP Algorithms

from input to the output layer. The major use cases of MLP are prediction, pattern classification, approximation and recognition.

2.3.7 XG Boost

XG Boost is the most used machine learning algorithm, whether it is a classification or a regression problem. It is known for its good performance as compared to all other machine learning algorithms. XG Boost is an implementation of decision trees with gradient boosts optimised for speed and efficiency. It is a software library that can download and install on the machine, then access various interfaces. The library focuses on computational speed and model performance with a laser, as there are few frills. XG Boost mostly uses for supervised learning in machine learning. It carries out the decision tree algorithm of gradient boosting. It has many common names, such as gradient boosting, gradient boosting machine, Etc. Boosting is nothing but ensemble techniques where previous model errors resolve in the new models. These models are applied immediately until no further progress is seen.

2.3.8 AdaBoost

AdaBoost is short for Adaptive Boosting. To improve the accuracy of classifiers, it integrates several classifiers. AdaBoost is a method for an iterative ensemble. By merging several poorly performing classifiers, the AdaBoost classifier creates a robust classifier to get a strong classifier with high accuracy. The fundamental principle behind AdaBoost is to train the data in every iteration to ensure that unexpected observations are predicted accurately. It also fixed the weights of the classifiers. If weights are taken on the training set, any machine learning algorithm may be used as a base classifier. It is the perfect starting point for boosting understanding. Besides, modern boosting techniques are based on AdaBoost, especially stochastic gradient boosting machines.

AdaBoost Working Model:

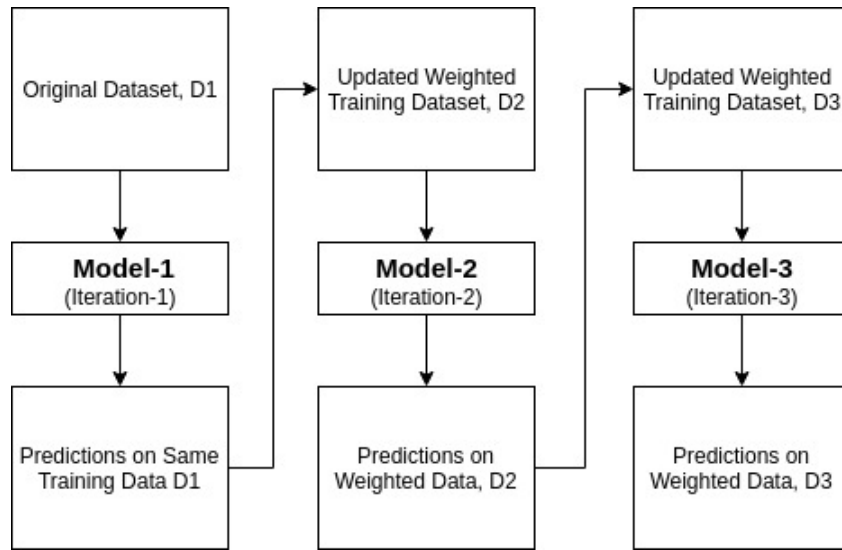


Figure 2.6: Adaboost Algorithms

It is best used to improve the efficiency of binary classification issues in decision trees. Besides, the first tree is generated, and the tree's performance will use for each training example. We also use it to weight how much attention the next tree gives to us. Hence more weight is assigned to training knowledge that is difficult to predict. However, less weight is given to instances that are easy to predict.

Chapter 3

Proposed Model

3.1 Dataset description

Promise Dataset (Jm1, Cm1, Pc1): Jm1, Cm1 and Pc1 all three of these dataset publicly available NASA promise dataset [23].

Table 3.1: Promise Dataset Attribute

Attribute	Attribute Detail	Type
1. loc	Line of Code	numeric
2. V(g)	Cyclomatic complexity	numeric
3. ev(g)	Essential Complexity	numeric
4. iv(g)	Decision Complexity	numeric
5. n	Total number of operands and operators	numeric
6. v	Volume of program in bits	numeric
7. l	Length of program	numeric
8. d	Handling Difficulties	numeric
9. i	Intelligence content count	numeric
10. e	Amount of mental activity need to translate the existing algorithm into implementation	numeric
11. b	Number of delivered bugs	numeric
12. t	Time estimation	numeric
13. lOCcode	Line count of program	numeric
14. lOCcomment	Line of comment	numeric
15. lOBblank	Count of blank lines	numeric
16. lOCcodeAndComment		numeric
17. uniq_Op	Total Operators	numeric
18. uniq_Opnd	Count of unique Operands	numeric
19. total_Op	Total Operators	numeric
20. total_Opnd	Total Operands	numeric
21. branchCount	The flow graphs	numeric
22. defects		False, True

Table 3.2: Number of attributes

Attribute Types	JM1	CM1	PC1
Different lines of code measure	5	5	5
McCabe metrics	3	3	3
Base Halstead measures	4	4	4
Derived Halstead measures	8	8	8
Branch-count	1	1	1
Goal Field	1	1	1

Table 3.3: Number of Instance

	JM1	CM1	PC1
Number of Instance	10885	498	1109

Table 3.4: Class Distribution

Class Distribution	JM1		CM1		PC1	
	True	False	True	False	True	False
Discrete	2106 = 19.35%	8779 = 80.65%	449 = 90.16%	4 = 9.83%	77 = 6.94%	1032 = 93.05%

Histogram and HeatMap of Promise Dataset

Histogram and HeatMap of CM1

Our dataset has 327 elements, and we preprocessed our dataset. From these 327 values, we have 42 False values and 285 True values. We can see the frequency of defects of our dataset in the histogram.

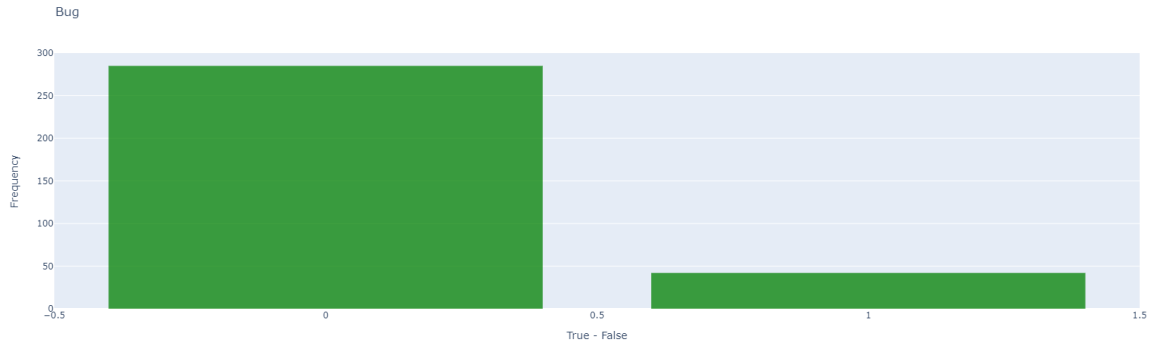


Figure 3.1: Histogram of Defect's Frequency for CM1 Dataset

We use a heat map tool to visualise our software fault dataset, a two-dimensional graphical representation of the matrix, and show the correlation among the input data. Seaborn visualisation library is used to indicate the heat map of our input data, as shown below.

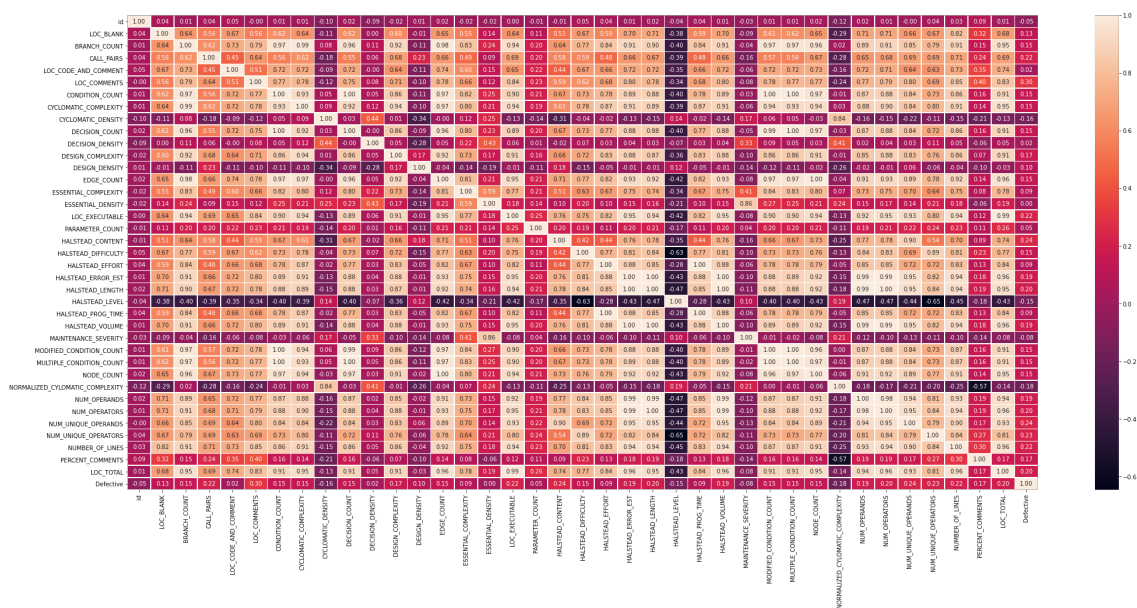


Figure 3.2: Heatmap of input data for CM1 dataset

Histogram and HeatMap of PC1

Our dataset has 705 elements, and we preprocessed our dataset. From these 705 values, we have 61 False values and 644 True values. We can see the frequency of defects of our dataset in the histogram.

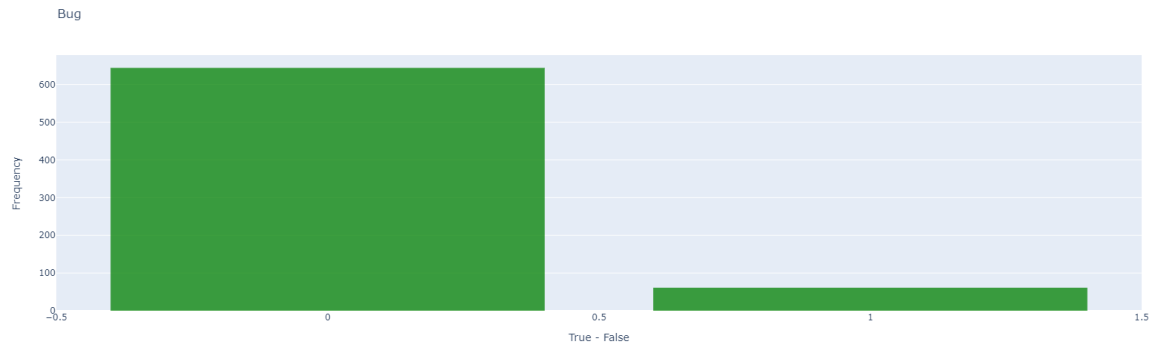


Figure 3.3: Histogram of Defect's Frequency for PC1 Dataset

We use a heat map tool to visualise our software fault dataset, a two-dimensional graphical representation of the matrix, and show the correlation among the input data. Seaborn visualisation library is used to indicate the heat map of our input data, as shown below.

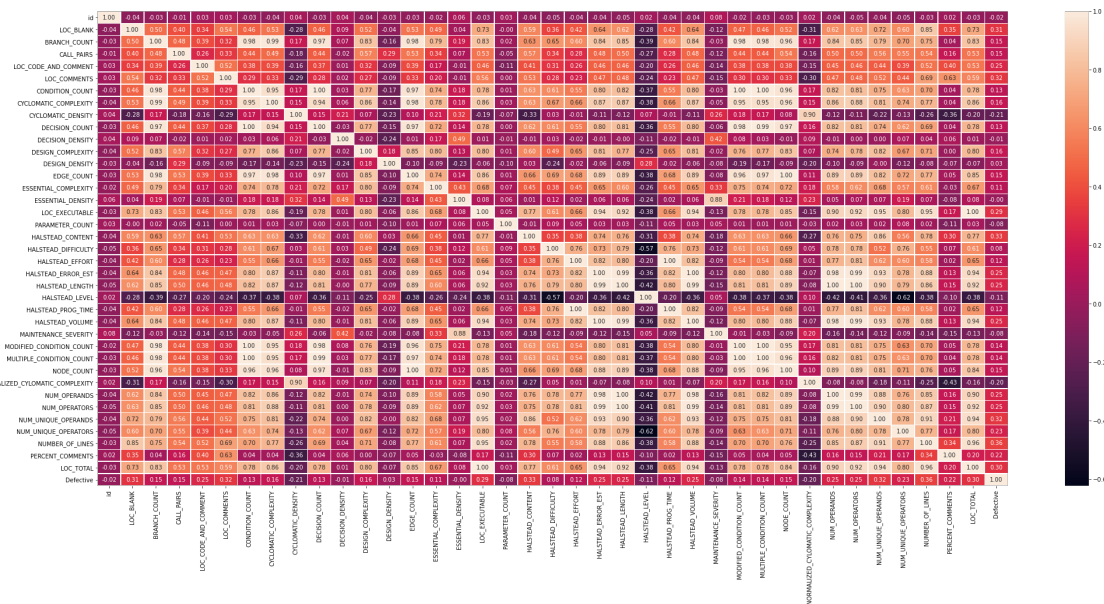


Figure 3.4: Heatmap of input data for PC1 dataset

Histogram and HeatMap of JM1

Our dataset has 10885 values. From these 10885 values, we have 8779 false value and 2106 true values. We can see the frequency of defects of our dataset in the histogram.

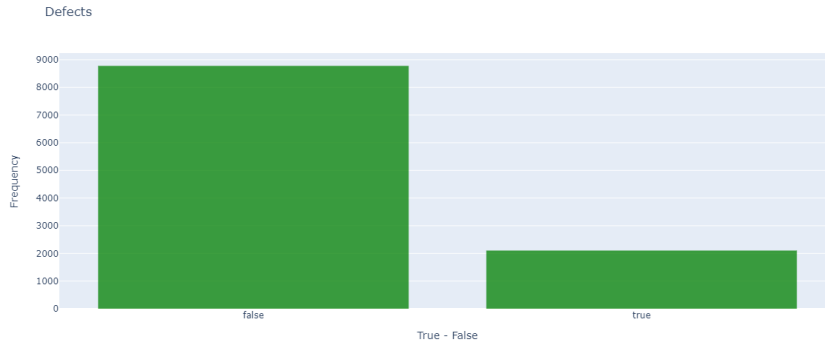


Figure 3.5: Histogram of Defect's Frequency for JM1 Dataset

We use a heat map tool to visualise our software fault dataset, a two-dimensional graphical representation of the matrix, and show the correlation among the input data. Seaborn visualisation library is used for indicating the heat map of our input data.

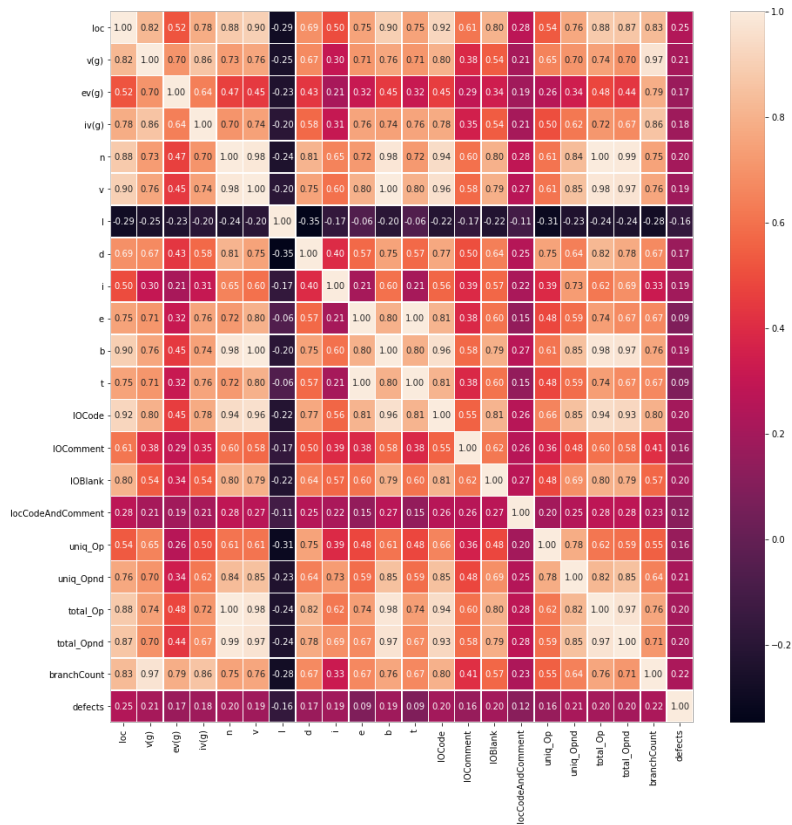


Figure 3.6: Heatmap of input data for JM1 dataset

Unified Dataset

This Unified dataset is the combination of 5 public datasets and this dataset is generated by using source code analysis [24] and [25]. those datasets are ,PROMISE – Jureczko and Madeyski (2010), Eclipse Bug Dataset (Zimmermann et al. 2007), Bug Prediction Dataset (D’Ambros et al. 2010), Bugcatchers Bug Dataset (Hall et al. 2014), GitHub Bug Dataset (Tóth et al. 2016).

Table 3.5: Unified dataset attributes

Category	Attributes Name	Attributes Details
Clone metrics	CC, CCL, CCO, CI, CLC, CLLC, LDC, LLDC	Clone Coverage, Clone Classes, Clone Complexity, Clone Instances, Clone Line Coverage, Clone Logical Line Coverage, Lines of Duplicated Code
Cohesion metrics	LCOM5	Lack of Cohesion in Methods 5
Complexity metrics	NL, NLE, WMC	Nesting Level, Nesting Level Else-If, Weighted Methods per Class
Coupling metrics	CBO, CBOI, NII, NOI, RFC	Weighted Methods per Class, Coupling Between Object Classes Inverse, Number of Incoming Invocations, Number of Outgoing Invocations, Response set For Class
Documentation metrics	AD, CD, CLOC, DLOC, PDA, PUA, TCD, TCLOC	API Documentation, Comment Density, Comment Lines of Code, Documentation Lines of Code, Public Documented API, Public Undocumented API, Total Comment Density, Total Comment Lines of Code

Table 3.6: Unified dataset attributes

Inheritance metrics	DIT, NOA, NOC, NOD, NOP	Depth of Inheritance Tree, Number of Ancestors, Number of Children, Number of Descendants, Number of Parents
Size metrics	LLOC, LOC, NA, NG, NLA, NLG, NLM, NLPA, NLPM, NLS, NM, NOS,	Logical Lines of Code, Lines of Code, Number of Attributes, Number of Getters, Number of Local Attributes, Number of Local Getters, Number of Local Methods, Number of Local Public Attributes, Number of Local Public Methods, Number of Local Setters, Number of Methods, Number of Statements, Number of Public Attributes,
Size metrics	NPA, NPM, NS, TLLOC, TLOC, TNA, TNG, TNLA, TNLG, TNLM, TNLPA, TNLPM, TNLS, TNM, TNOS, TNPA, TNPM, TNS	Number of Public Methods, Number of Setters, Total Logical Lines of Code, Total Lines of Code, Total Number of Attributes, Total Number of Getters, Total Number of Local Attributes, Total Number of Local Getters, Total Number of Local Methods, Total Number of Local Public Attributes, Total Number of Local Public Methods, Total Number of Local Setters, Total Number of Methods, Total Number of Statements, Total Number of Public Attributes, Total Number of Public Methods, Total Number of Setters
	BUG	False and True

Histogram and HeatMap of UNIFIED Dataset

Our dataset has 47,618 elements, and we preprocessed our dataset. From these 47618 values, we have 38838 False values and 8780 True values. We can see the frequency of defects of our dataset in the histogram.

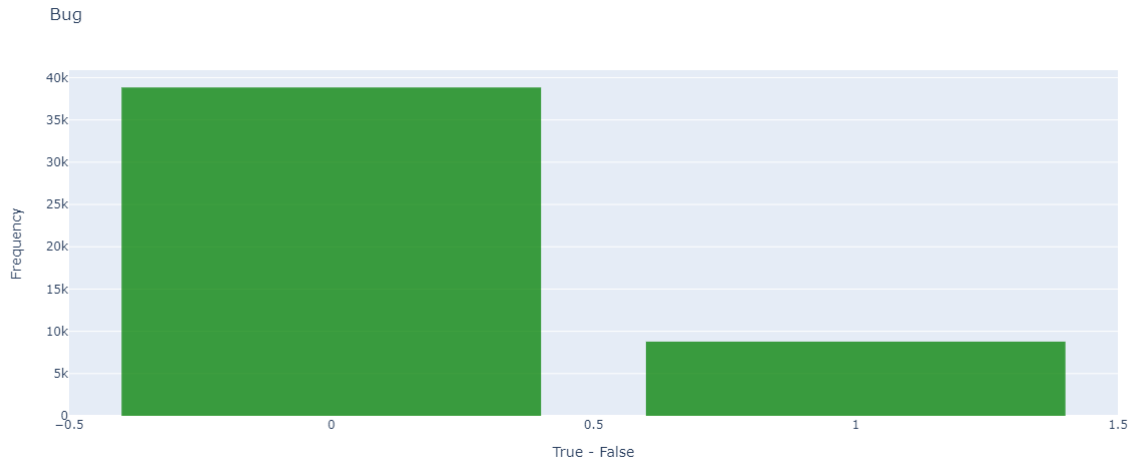


Figure 3.7: Histogram of Defect's Frequency for Unified Dataset

We use a heat map tool to visualize our software fault dataset, a two-dimensional graphical representation of the matrix, and show the correlation among the input data. Seaborn visualization library is used to show the heat map of our input data, as shown below.

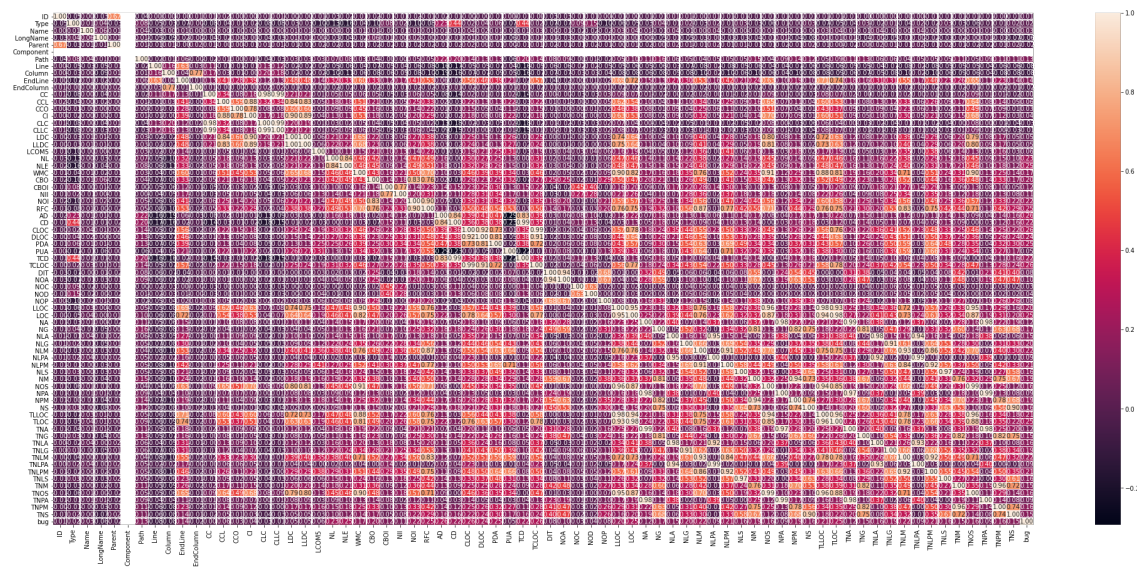


Figure 3.8: Heatmap of the input data for unified dataset

3.1.1 Data preprocessing

For each dataset, we choose different preprocessing methods to reduce noise from the dataset. We use the correlation matrix to see the correlation of each attribute to others. Then we use the histogram to see our level of the dataset and the frequency of the output feature of our dataset. Then we normalize our dataset to work within

it smoothly. We scaled our dataset attribute within zero to one. After that, as our dataset is not balanced, we balanced our dataset using the Smote library; a balanced dataset will give a more accurate result. Our all of this dataset, the frequency for the output feature is nearly 80%, 20% we balanced our entire dataset.

3.1.2 Feature Selection

In the dataset, the feature is each column. We have input and our features to train our model. For jm1, cm1 and pc1 dataset, we choose ‘defects’ for the output feature and the rest of the feature is used for input for each algorithm. We choose each feature in the input because we got very near values in the correlation. For, Unified bug dataset, we select the “bug” feature for output and the rest of the features we select as input.

3.2 Model Description

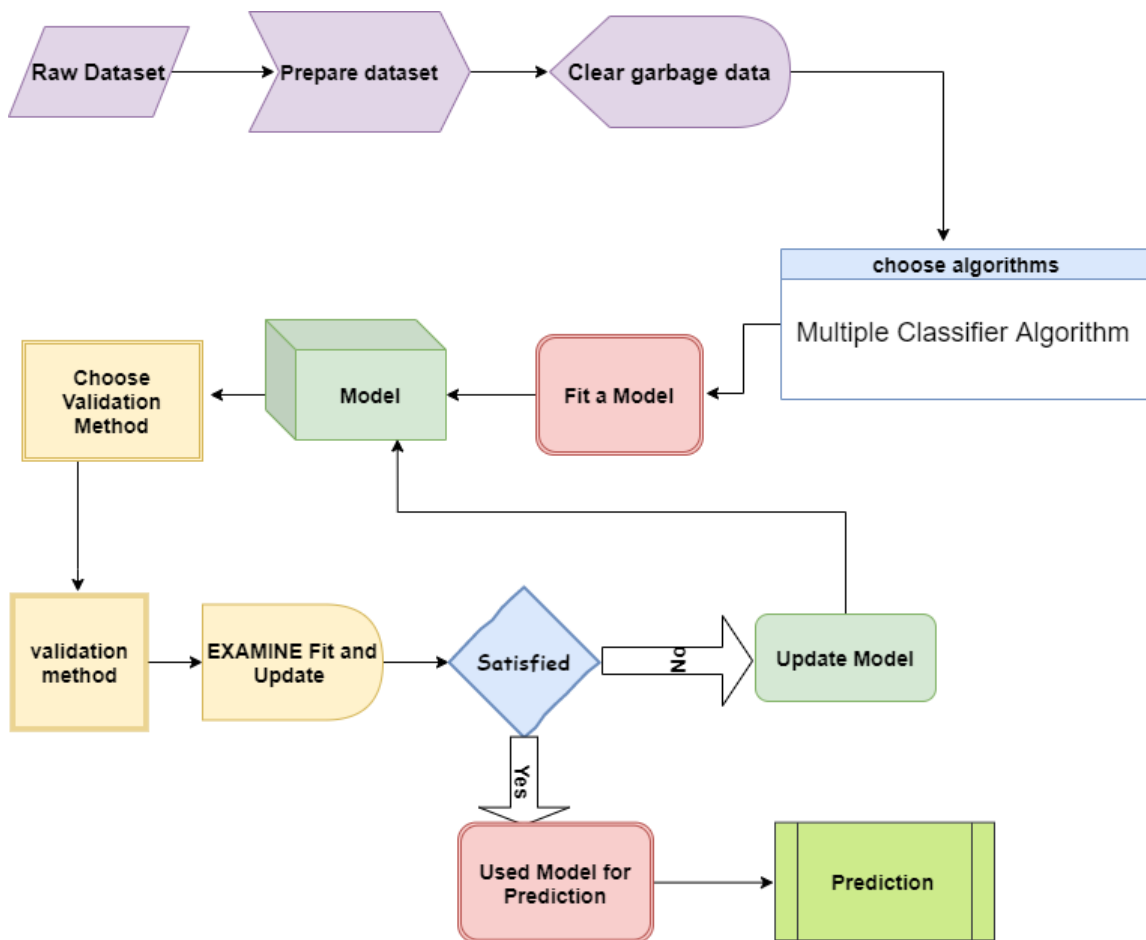


Figure 3.9: Workflow of the System

We used these six-classifier machine learning algorithms and two boosting algorithms. We worked in Google Co-lab online IDE. We first chose four datasets for our thesis work. We use the different preprocessing library and our programming knowledge to preprocess our dataset. After that, we select our features for our model. Then we split the dataset to train and test. In each dataset, we apply eight different algorithms on the train part. Then we apply the test part on this model to validate our model. We use precision, recall, F1 measures to validate our model. Next, we get our accuracy by using classification metrics. After that, we get our accuracy. Then we go to the evaluation part, where we use the confusion matrix and ROC curve to evaluate our model. By using the ROC curve, we can visualize our model performance.

Chapter 4

Experimentation

i. **Confusion Matrix:** This matrix actually helps to find the accuracy of algorithms. It is a 2 by 2 matrix and helps to analyze the properties of classification algorithms. This matrix gives all the values needed for accurate measurement. From the below table, we can see that it gives True positive, false positive, false negatives and true negatives. we can measure precision, recall and f1 from these attributes for each algorithm. This matrix also helps to plot ROC curves.

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

Table 4.1: CM1 Experiments Result

DataSet	Algorithms		Precision	Recall	F1-Measure	Accuracy
CM1 dataset	Random Forest	0	0.95	0.89	0.92	91%
		1	0.87	0.94	0.91	
	AdaBoost	0	0.92	0.89	0.90	89%
		1	0.87	0.90	0.88	
	KNN	0	0.97	0.62	0.76	78%
		1	0.68	0.98	0.80	
	Decision Tree	0	0.95	0.89	0.92	91%
		1	0.87	0.94	0.91	
	XGBoost	0	0.93	0.87	0.90	89%
		1	0.85	0.92	0.89	
	MLP	0	0.90	0.75	0.82	82%
		1	0.74	0.90	0.81	
	SVM	0	0.82	0.67	0.74	74%
		1	0.67	0.82	0.74	
	Naïve Bayes	0	0.93	0.85	0.89	80%
		1	0.10	0.20	0.13	

Here Random Forest its prediction accuracy is 91%, whereas, for AdaBoost with random forest, its accuracy is 89%. For KNN its accuracy in software fault prediction is 78%. For Decision Tree its accuracy in software fault prediction is 91%. For XGBoost its accuracy in software fault prediction is 89%. For MLP its accuracy in software fault prediction is 82%. For SVM its accuracy in software fault prediction is 74%. For Naïve Bayes its accuracy in software fault prediction is 80%. Here, we can see that we have similar accuracy for Random forest and Decision Tree, AdaBoost and XGBoost. The Random Forest and Decision tree have slightly more accuracy than these algorithms.

Table 4.2: PC1 Experiments Result

DataSet	Algorithms		Precision	Recall	F1- Measure	Accuracy
PC1 dataset	Random Forest	0	1.00	0.94	0.97	97%
		1	0.95	1.00	0.97	
	AdaBoost	0	1.00	0.94	0.97	97%
		1	0.95	1.00	0.97	
	KNN	0	1.00	0.75	0.86	88%
		1	0.81	1.00	0.90	
	Decision Tree	0	0.92	0.90	0.91	91%
		1	0.90	0.92	0.91	
	XGBoost	0	1.00	0.92	0.96	96%
		1	0.93	1.00	0.96	
	MLP	0	0.96	0.84	0.90	91%
		1	0.87	0.97	0.91	
	SVM	0	0.89	1.00	0.94	89%
		1	0.00	0.00	0.00	
	Naïve Bayes	0	0.93	0.94	0.93	88%
		1	0.43	0.40	0.41	

Here Random Forest accuracy is 97%, whereas, for AdaBoost accuracy is 97%. For KNN accuracy in software fault prediction is 88%. For the Decision Tree its accuracy in software fault prediction is 91%. For XGBoost its accuracy in software fault prediction is 96%. For MLP its accuracy in software fault prediction is 91%. For SVM its accuracy in software fault prediction is 86%. For Naïve Bayes's accuracy in software fault prediction is 88%. Here, we can see that we have similar accuracy for Random Forest and AdaBoost, Decision Tree and MLP, but the Random Forest and AdaBoost have slightly more accuracy than these algorithms.

Table 4.3: JM1 Experiemnts Result

Dataset	Algorithms		Precision	Recall	F1- Measure	Accuracy
JM1	Random Forest	0	0.89	0.90	0.89	90%
		1	0.90	0.89	0.90	
	AdaBoost	0	0.91	0.88	0.90	90%
		1	0.89	0.92	0.90	
	KNN	0	0.88	0.68	0.77	80%
		1	0.74	0.91	0.82	
	Decision Tree	0	0.83	0.82	0.83	83%
		1	0.83	0.84	0.83	
	XGBoost	0	0.79	0.85	0.82	82%
		1	0.85	0.78	0.81	
	MLP	0	0.80	0.98	0.88	79%
		1	0.53	0.08	0.14	
	SVM	0	0.79	1.00	0.88	79%
		1	1.00	0.00	0.00	
	Naïve Bayes	0	0.82	0.95	0.88	79%
		1	0.51	0.19	0.28	

Here Random Forest its prediction accuracy is 88%, whereas, for AdaBoost, its accuracy is 89%. For the KNN its accuracy in software fault prediction is 78%. For the Decision tree its accuracy in software fault prediction is 82%. For XGBoost its accuracy in software fault prediction is 81%. For MLP its accuracy in software fault prediction is 68%. For SVM accuracy in software fault prediction is 79%. For Naïve Bayes its accuracy in software fault prediction is 58%. Here, we can see that we have similar accuracy for Random Forest and AdaBoost. So Random Forest and AdaBoost have slightly more accuracy than these algorithms.

Table 4.4: Unified Dataset Experiments Result

Dataset	Algorithms		Precision	Recall	F1- Measure	Accuracy
Unified Bug Dataset	Random Forest	0	0.92	0.92	0.92	92%
		1	0.92	0.92	0.92	
	AdaBoost	0	0.93	0.91	0.92	92%
		1	0.91	0.93	0.92	
	KNN	0	0.95	0.78	0.86	87%
		1	0.82	0.96	0.88	
	Decision Tree	0	0.87	0.87	0.87	87%
		1	0.87	0.88	0.87	
	XGBoost	0	0.88	0.87	0.88	88%
		1	0.87	0.89	0.88	
	MLP	0	0.88	0.96	0.92	86%
		1	0.67	0.40	0.50	
	SVM	0	0.82	0.99	0.90	81%
		1	0.13	0.01	0.02	
	Naïve Bayes	0	0.86	0.92	0.89	81%
		1	0.45	0.29	0.35	

Here Random Forest its prediction accuracy is 92%, whereas, for AdaBoost, its accuracy is 92%. For KNN its accuracy in software fault prediction is 87%. For the Decision Tree its accuracy in software fault prediction is 87%. For XGBoost its accuracy in software fault prediction is 87%. For MLP its accuracy in software fault prediction is 83%. For SVM its accuracy in software fault prediction is 70%. For Naïve Bayes its accuracy in software fault prediction is 70%. Here, we can see that we have similar accuracy for KNN and Decision Tree and XGBoost, but the Random Forest and AdaBoost have slightly more accuracy than these algorithms.

Chapter 5

Result Analysis (ROC)

5.1 Roc Curve

A ROC curve is actually a way of visualizing the performance of one algorithm. The ROC curve works with TPR and FPR. ROC curve plot is based on TPR as the x-axis and FPR as the y-axis at several threshold settings. TPR and FPR on the x-axis and y-axis give a different probability, and on this probability, the curve is plotted. If the curve is above the middle line, the algorithms work well; the curve under the threshold means algorithms give poor results here. For better accuracy, the area under the curve should be high.

5.1.1 CM1 ROC curve

Here, there are some receiver operating characteristic (ROC) curves for the existing algorithms. We used some machine learning algorithms and demonstrated the ROC curve for each algorithm.

The Random Forest and Decision Tree ROC curve is given below:

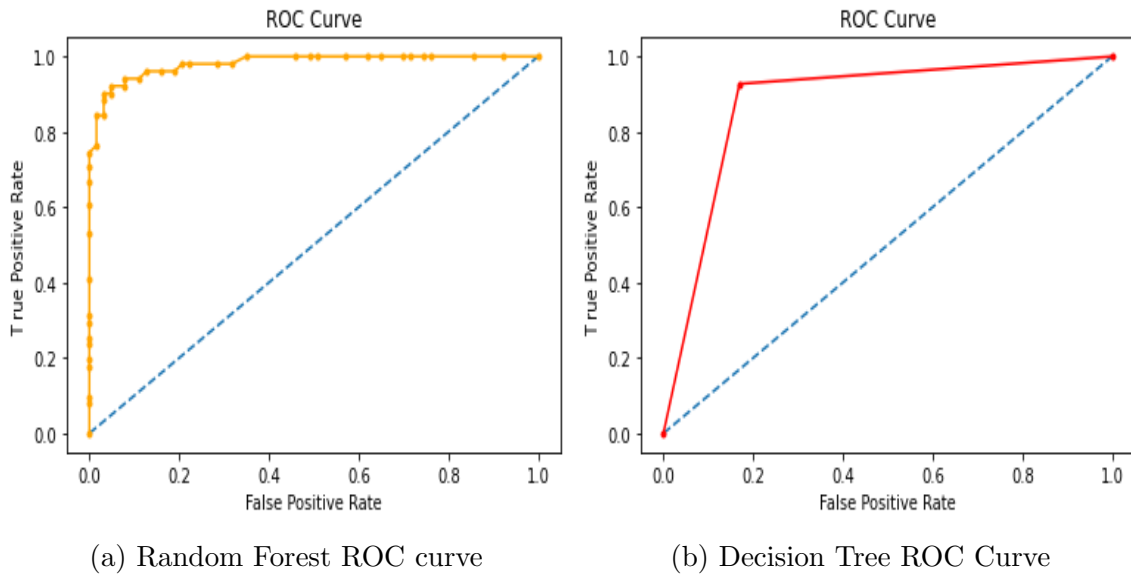


Figure 5.1: Random Forest and Decision Tree ROC Curve

We have applied KNN algorithm and MLP algorithm to our dataset and demonstrate ROC curve.

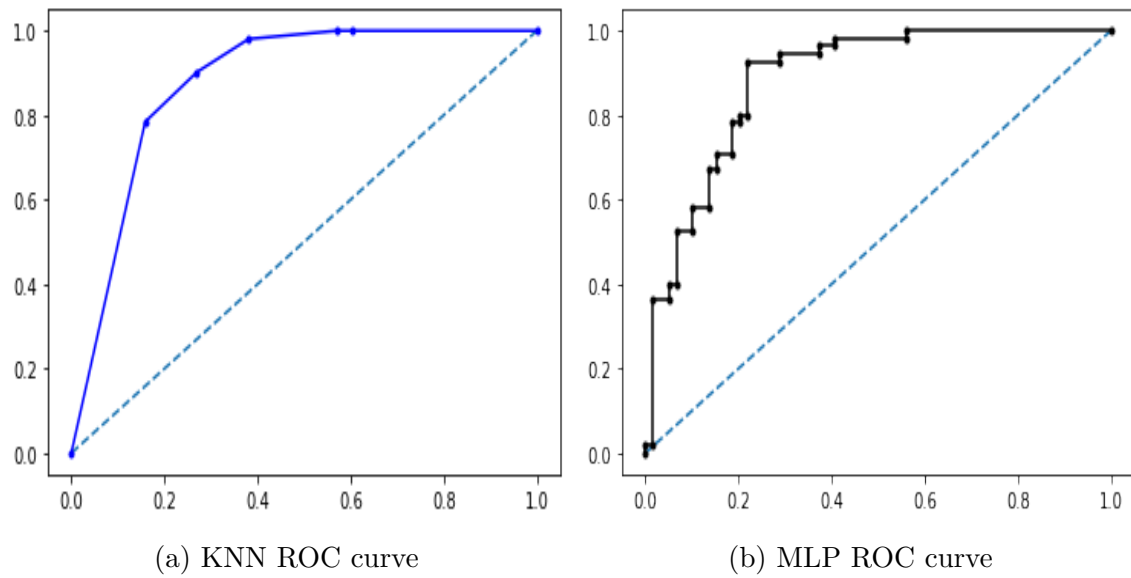


Figure 5.2: KNN and MLP ROC Curve

We have applied Naïve Bayes algorithm and SVM algorithm to our dataset and demonstrate ROC curve.

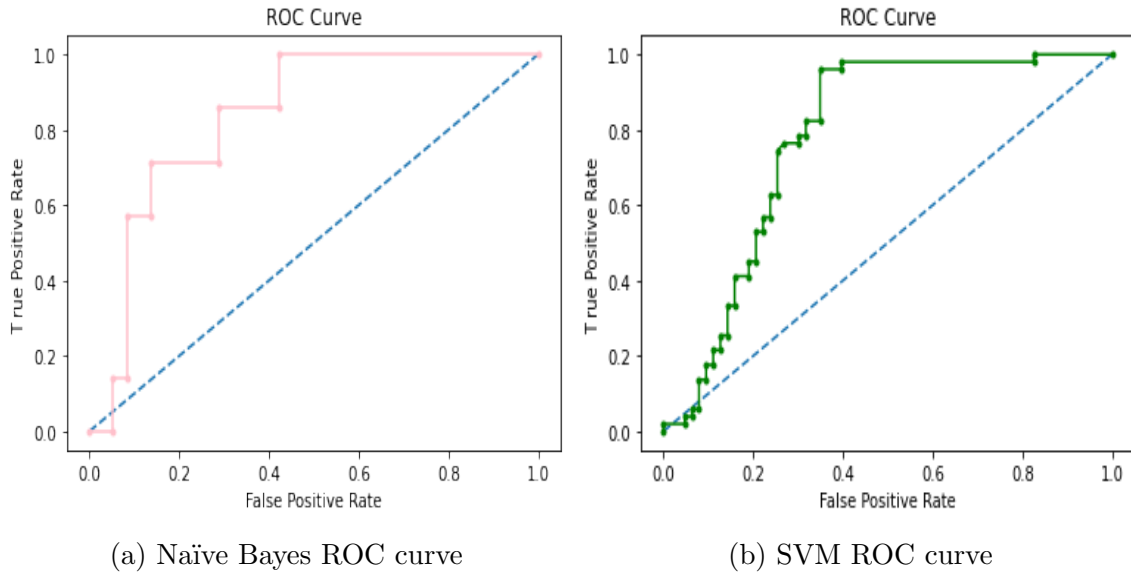


Figure 5.3: Naïve Bayes and SVM ROC curve

We have applied XGBoost algorithm and AdaBoost algorithm to our dataset and demonstrate ROC curve.

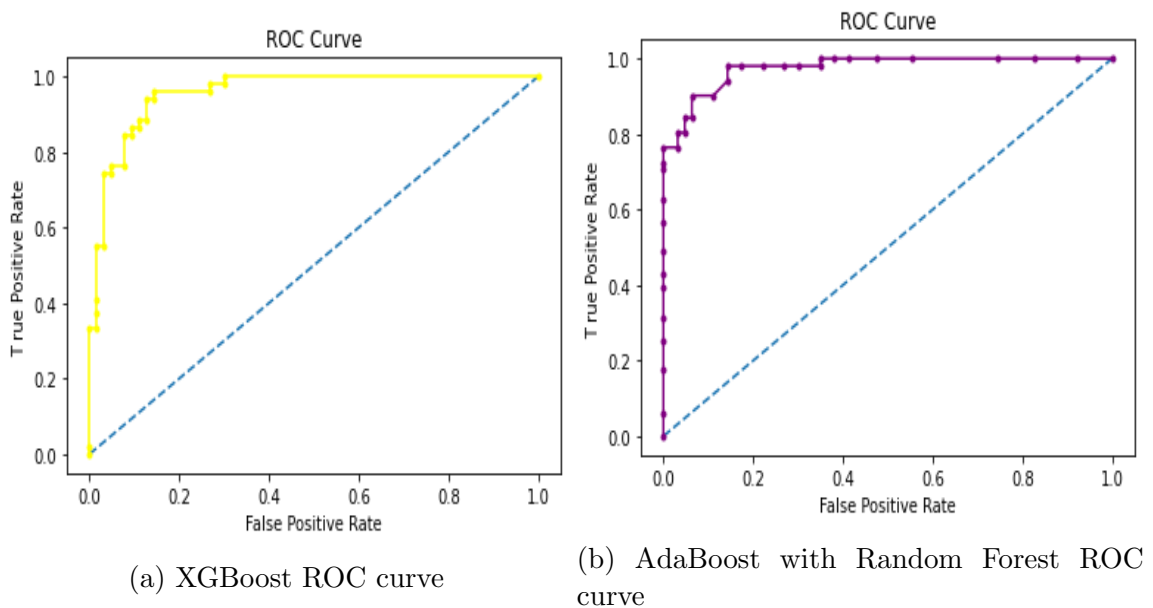


Figure 5.4: XGBoost and AdaBoost ROC Curve

We have combined all the algorithms and demonstrate a combined ROC curve, where the y-axis plots the True positive rate and the false positive rate at the x-axis.

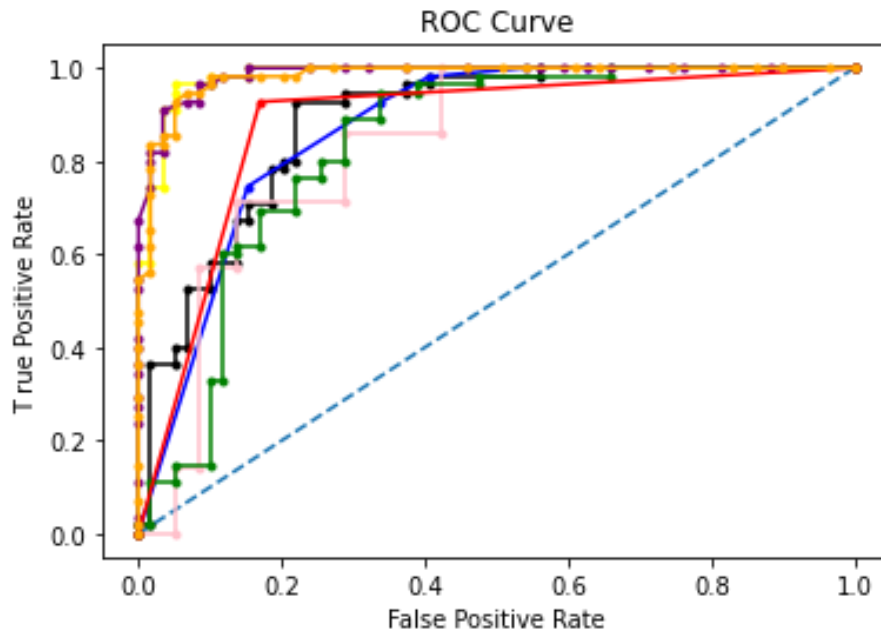


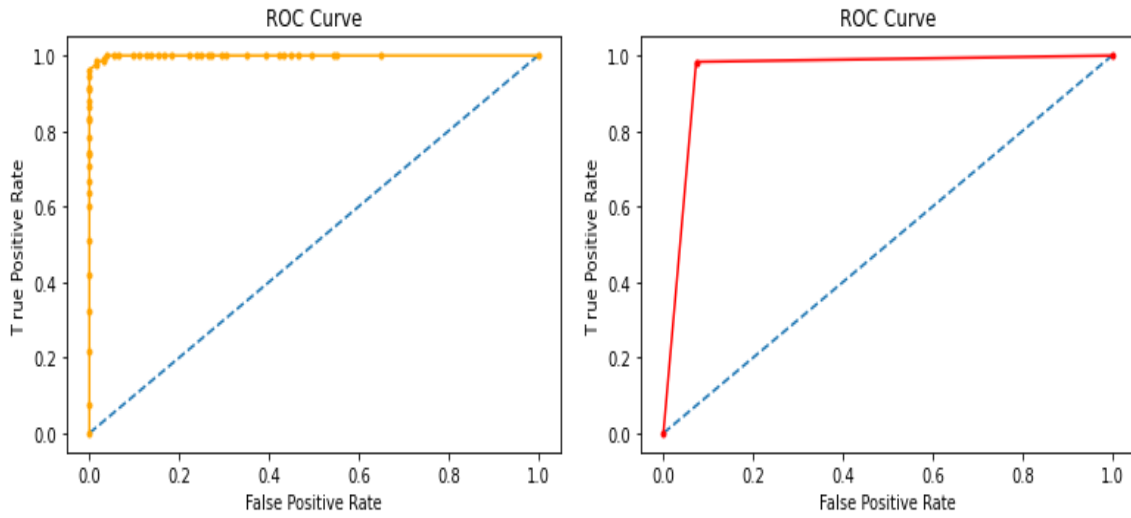
Figure 5.5: Combined ROC curve of all algorithms

Here in the combined ROC curve, we use Orange colour for Random Forest, Purple for AdaBoost, Red for Decision Tree, Green for SVM, Yellow for XGBoost, Pink for Naive Bayes, Blue for KNN and Black for MLP.

5.1.2 PC1 ROC curve

Here, there are some receiver operating characteristic (ROC) curves for the existing algorithms. We used some machine learning algorithms and demonstrated the ROC curve for each algorithm.

We have applied the Random Forest and Decision Tree algorithm to our dataset and demonstrate the ROC curve

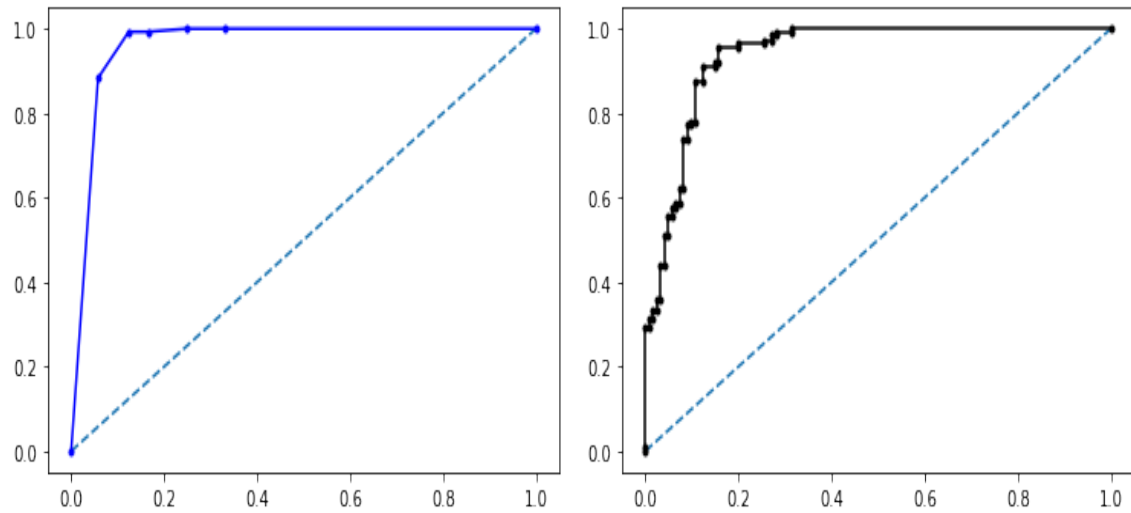


(a) Random Forest ROC curve

(b) Decision Tree ROC Curve

Figure 5.6: Random Forest and Decision Tree ROC Curve

We have applied KNN algorithm and MLP algorithm to our dataset and demonstrate ROC curve.



(a) KNN ROC curve

(b) MLP ROC curve

Figure 5.7: KNN and MLP ROC Curve

We have applied Naïve Bayes algorithm and SVM algorithm to our dataset and demonstrate ROC curve.

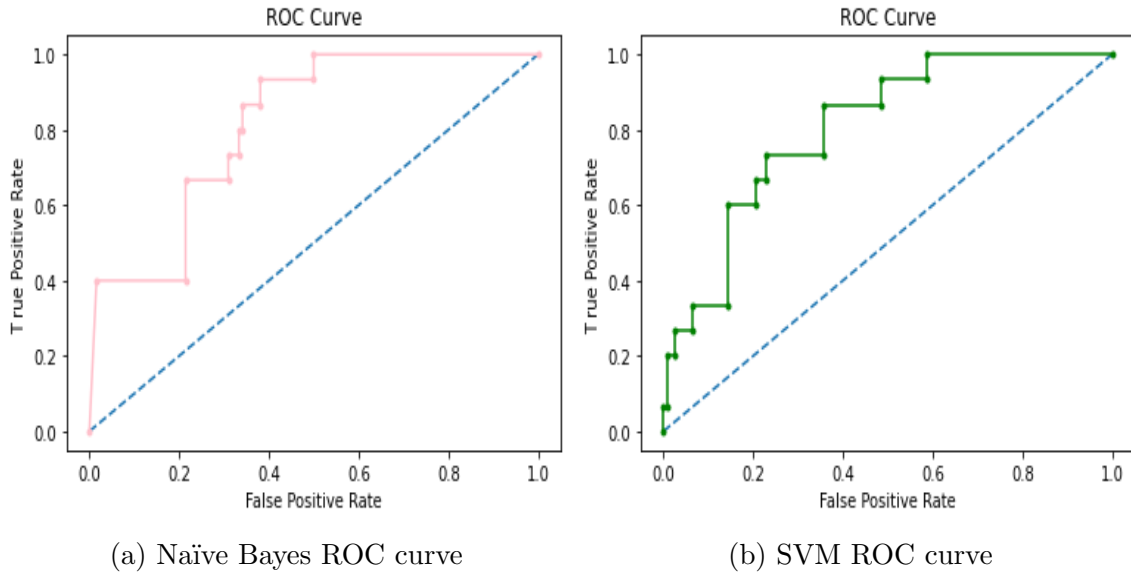


Figure 5.8: Naïve Bayes and SVM ROC curve

We have applied XGBoost algorithm and AdaBoost with Random Forest algorithm to our dataset and demonstrate ROC curve.

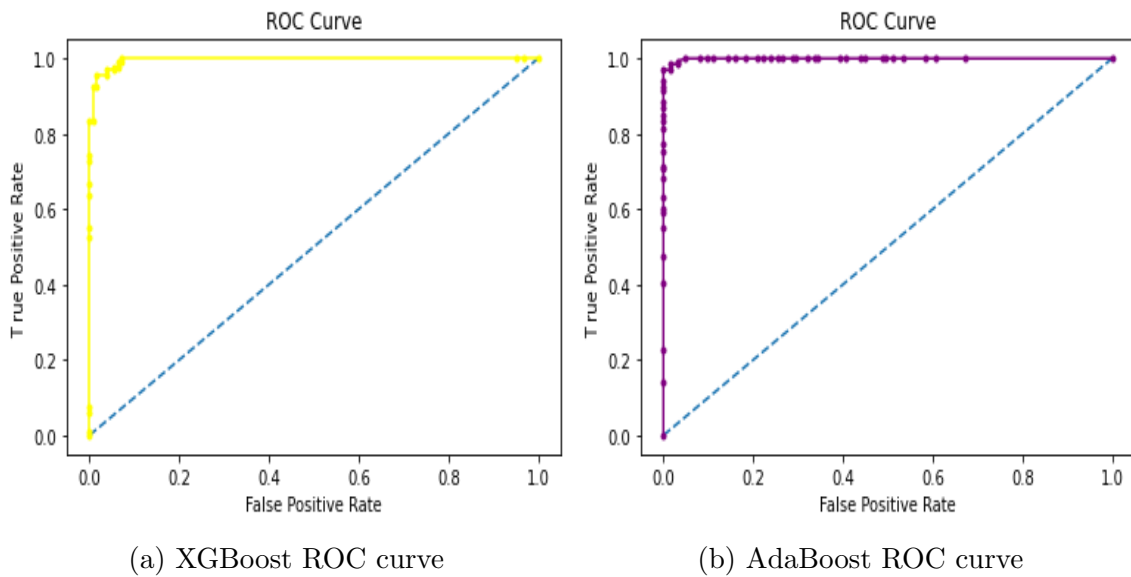


Figure 5.9: XGBoost and AdaBoost ROC Curve

We have combined all the algorithms and demonstrate a combined ROC curve. where the y-axis plots the True positive rate and the false positive rate at the x-axis.

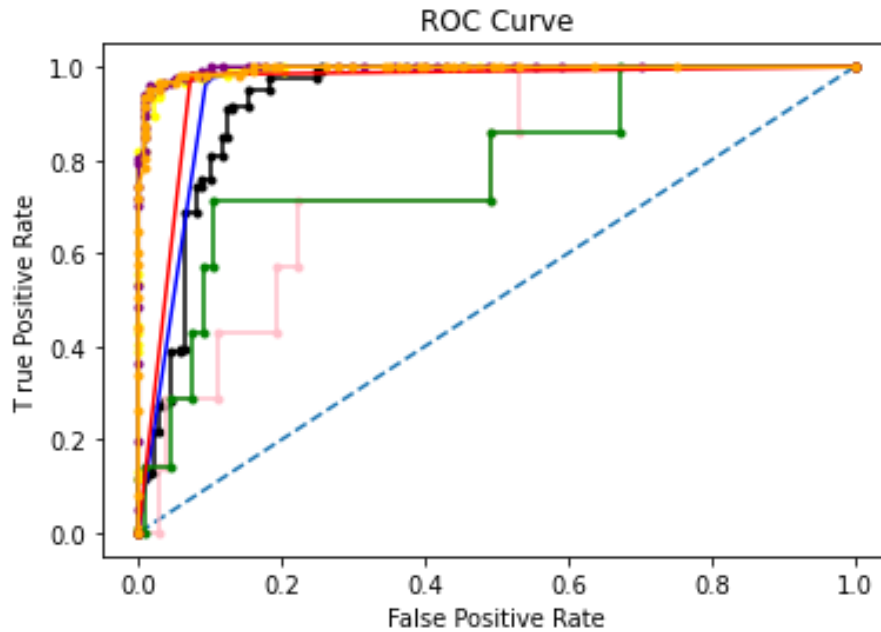


Figure 5.10: Combined ROC curve of all algorithms

Here in the combined ROC curve, we use Orange colour for Random Forest, Purple for AdaBoost, Red for Decision Tree, Green for SVM, Yellow for XGBoost, Pink for Naive Bayes, Blue for KNN and Black for MLP.

5.1.3 JM1 ROC curve

Here, there are some receiver operating characteristic (ROC) curves for the existing algorithms. We used some machine learning algorithms and demonstrated the ROC curve for each algorithm.

We have applied the Random Forest and Decision Tree algorithm to our dataset and demonstrate the ROC curve.

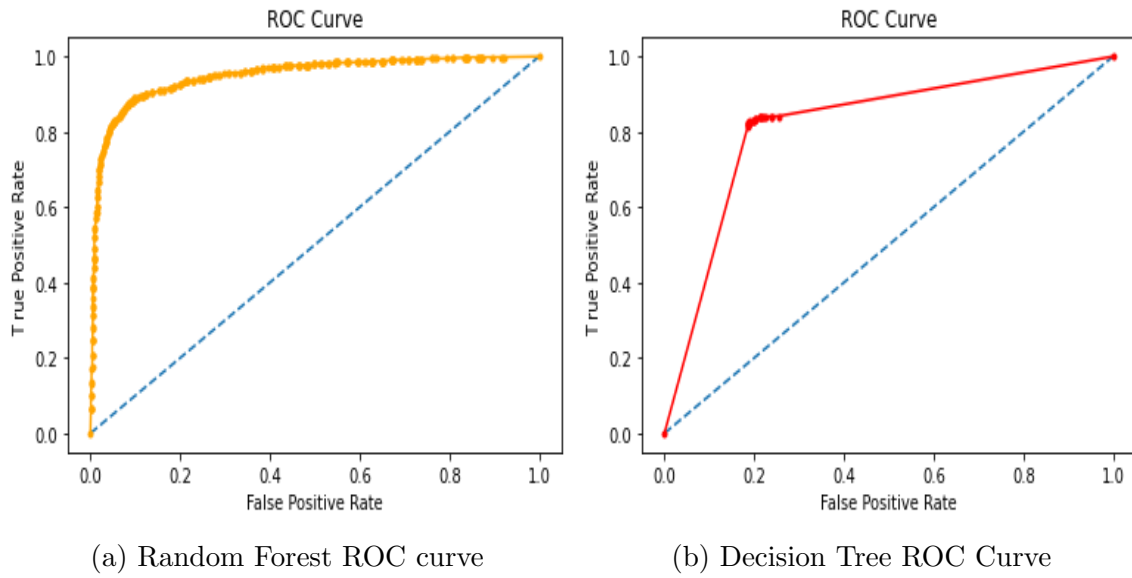


Figure 5.11: Random Forest and Decision Tree ROC Curve

We have applied KNN algorithm and MLP algorithm to our dataset and demonstrate ROC curve.

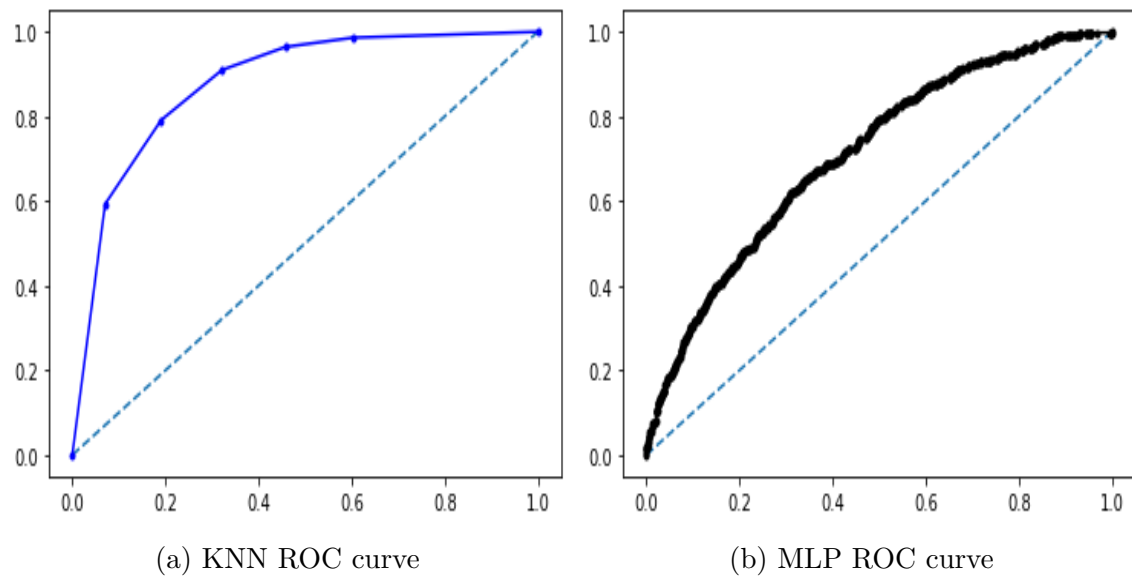


Figure 5.12: KNN and MLP ROC curve

We have applied Naïve Bayes algorithm and SVM algorithm to our dataset and demonstrate ROC curve.

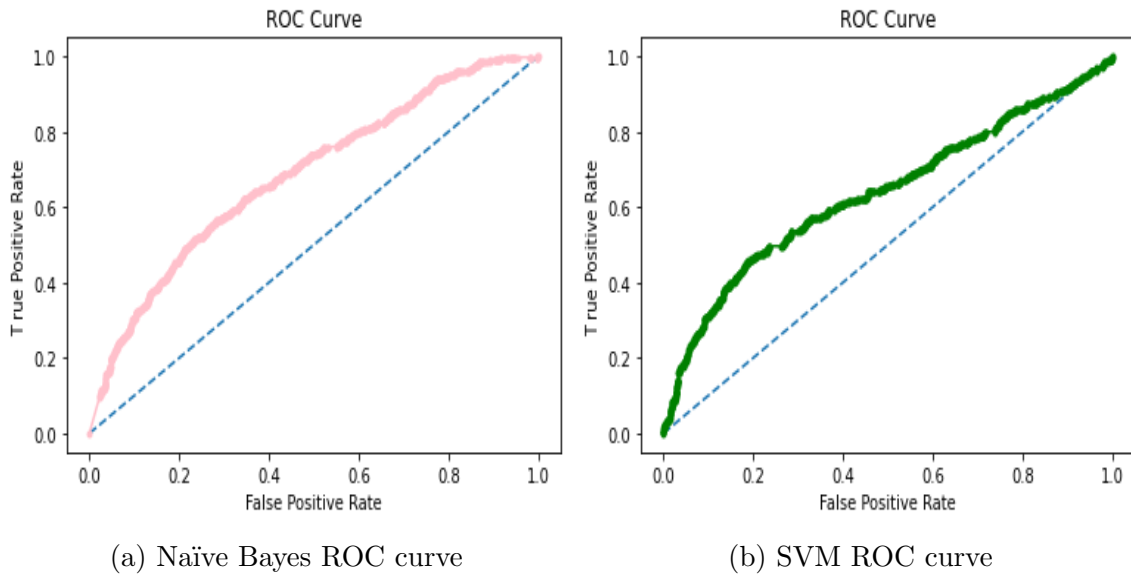


Figure 5.13: Naïve Bayes and SVM ROC Curve

We have applied XGBoost algorithm and AdaBoost with Random Forest algorithm to our dataset and demonstrate ROC curve.

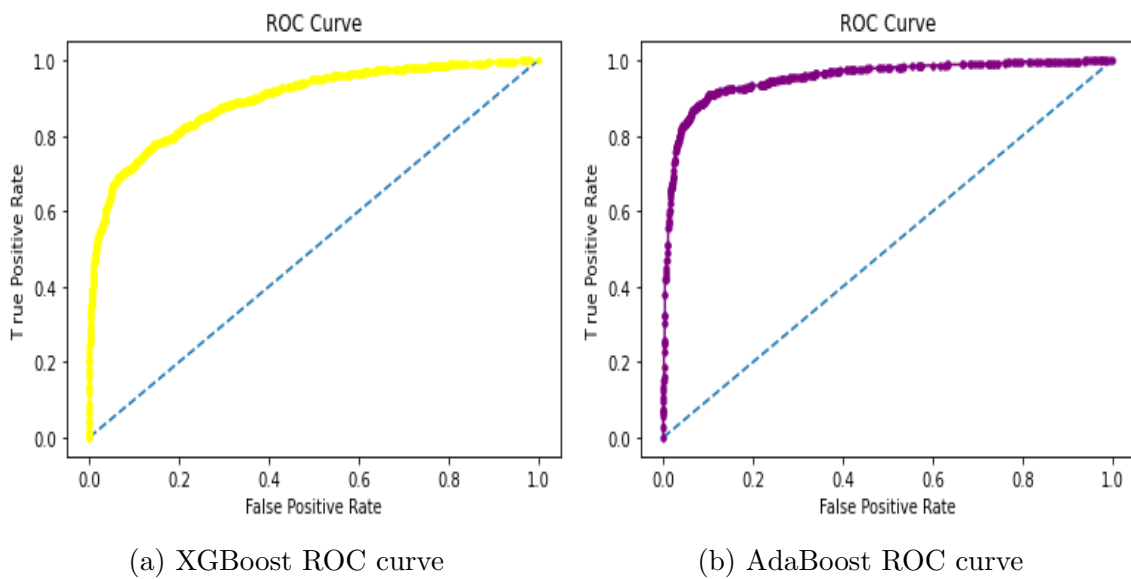


Figure 5.14: XGBoost and AdaBoost ROC Curve

We have combined all the algorithms and demonstrate a combined ROC curve. where the y-axis plots the True positive rate and the false positive rate at the x-axis.

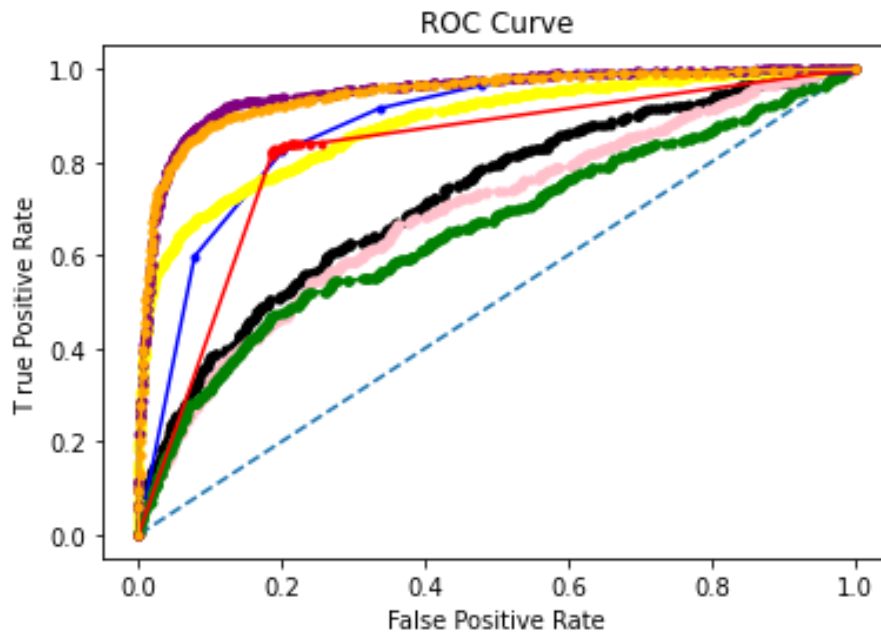


Figure 5.15: Combined ROC curve of all algorithms

5.1.4 Unified Dataset ROC curve

Here, there are some receiver operating characteristic (ROC) curves for the existing algorithms. We used some machine learning algorithms and demonstrated the ROC curve for each algorithm.

We have applied the Random Forest and Decision Tree algorithm to our dataset and demonstrate the ROC curve.

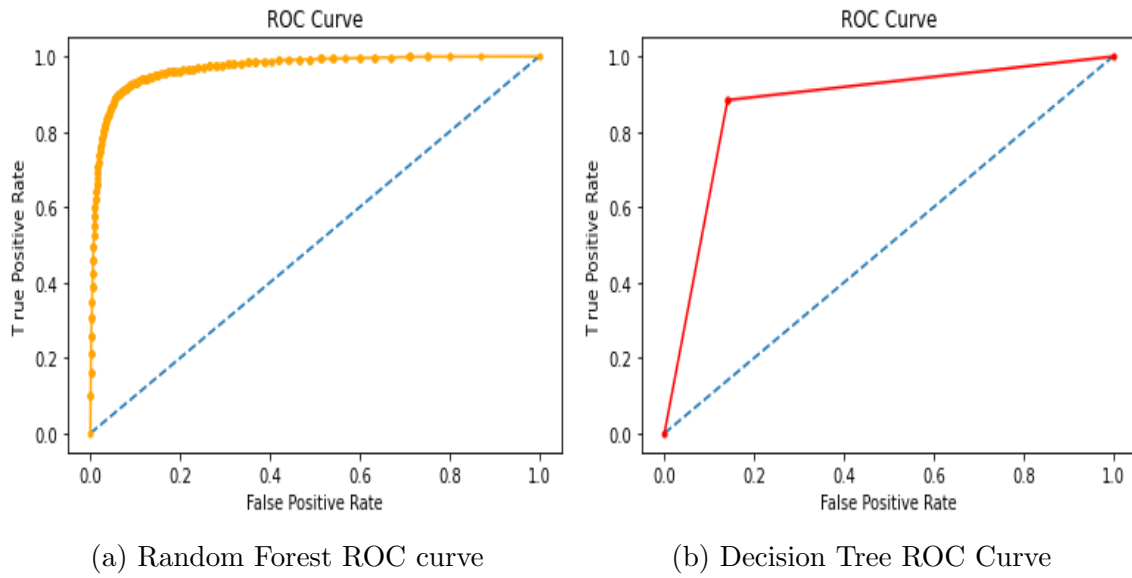


Figure 5.16: Random Forest and Decision Tree ROC Curve

We have applied KNN algorithm and MLP algorithm to our dataset and demonstrate ROC curve.

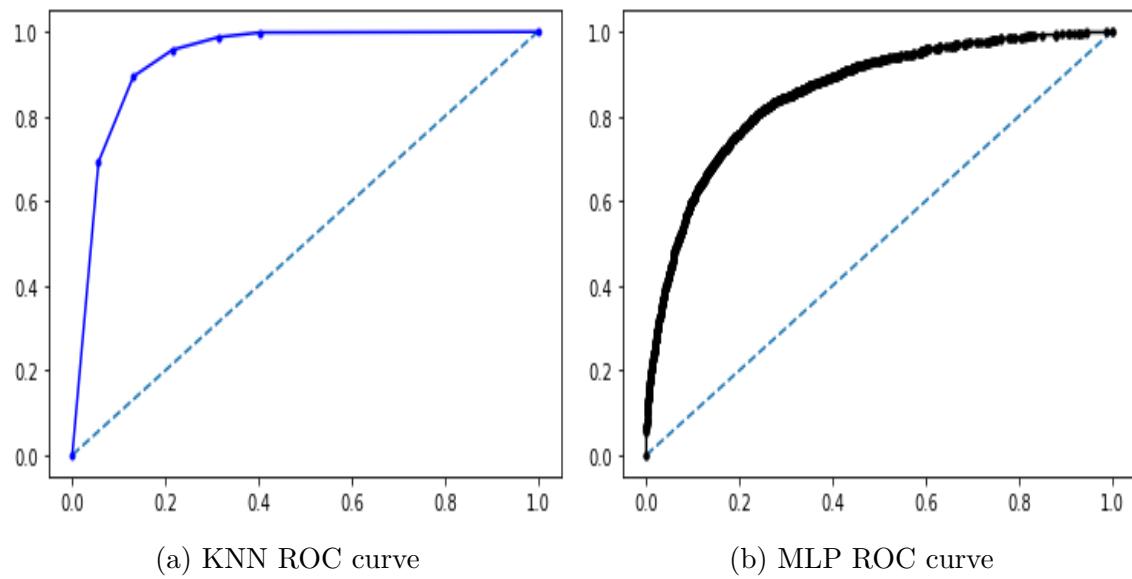


Figure 5.17: KNN and MLP ROC Curve

We have applied Naïve Bayes algorithm and SVM algorithm to our dataset and demonstrate ROC curve.

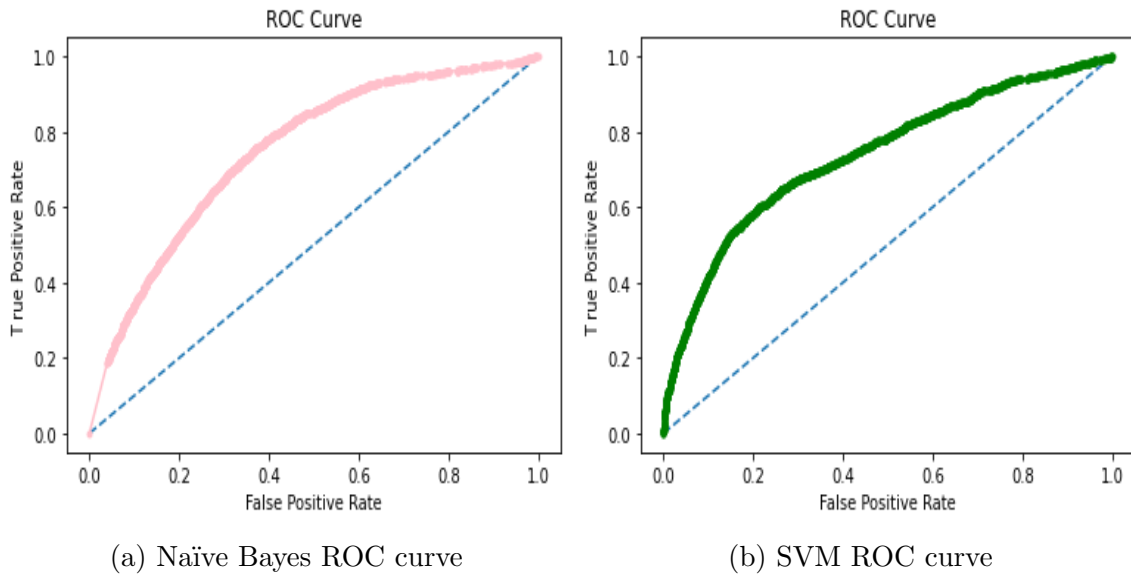


Figure 5.18: Naïve Bayes and SVM ROC Curve

We have applied XGBoost algorithm and AdaBoost with Random Forest algorithm to our dataset and demonstrate ROC curve.

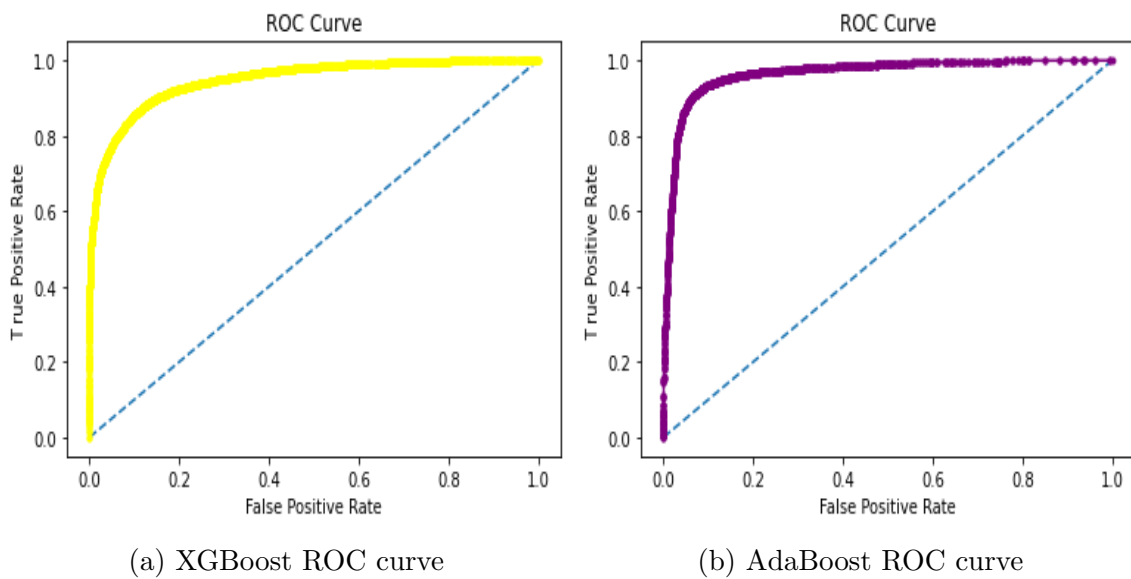


Figure 5.19: XGBoost and AdaBoost ROC Curve

We have combined all the algorithms and demonstrate a combined ROC curve. where the y-axis plots the True positive rate and the false positive rate at the x-axis.

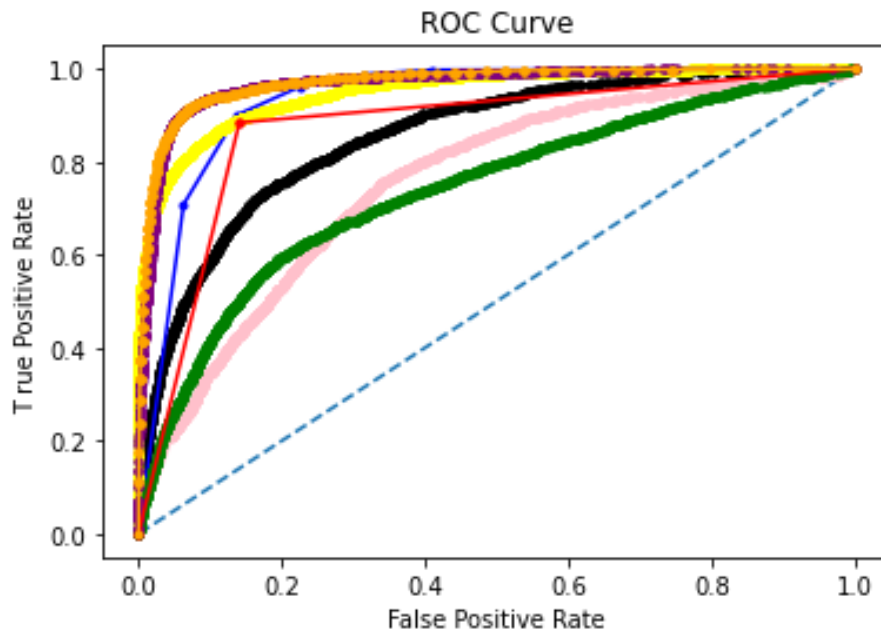


Figure 5.20: Combined ROC curve of all algorithms

Here in the combined ROC curve, we use Orange colour for Random Forest, Purple for AdaBoost, Red for Decision Tree, Green for SVM, Yellow for XGBoost, Pink for Naive Bayes, Blue for KNN and Black for MLP.

Table 5.1: accuracy table of all four datasets for all Eight algorithms

Algorithm	Unified Dataset	JM1 Dataset	CM1 Dataset	PC1 Dataset
Random Forest	92%	90%	91%	97%
Adaboost	92%	90%	89%	97%
Decision Tree	87%	83%	91%	91%
SVM	81%	79%	74%	89%
XGBoost	88%	82%	89%	96%
Naive Bayesian	81%	79%	80%	88%
KNN	87%	80%	78%	88%
MLP	86%	79%	82%	91%

Table 5.2: comparison of related paper accuracy with our accuracy

Algorithms	Dataset	Our Work	Paper[19]	Paper[22]	Paper[23]
Random Forest	Jm1	90%	81%	81.7%	
	Pc1	97%	93%	93.84%	
	Cm1	91%	87.93		
Adaboost	Jm1	90%		81%	
	pc1	97%		93%	
	Cm1	89%			
KNN	Jm1	80%			
	Pc1	88%			
	Cm1	78%			
Decision Tree	Jm1	83%		81%	80%
	Pc1	91%		93%	93%
	Cm1	91%			90%
XGBoost	Jm1	82%			
	Pc1	96%			
	Cm1	89%			
MLP	Jm1	79%			
	Pc1	91%			
	Cm1	82%			
SVM	Jm1	79%	81%	81%	
	Pc1	86%	93%	93%	
	Cm1	74%	90%		
Naïve Bayes	Jm1	79%	80%	80%	77%
	Pc1	88%	89%	89%	90%
	Cm1	80%	85%		82%

From the above table, we can see for JM1; we get the highest 90% in both Random forest and Adaboost. Compared with paper [19], they got 88% for their model and got 87.93% in Random Forest, which is the highest accuracy in jm1 for this paper. On the other hand, in this paper [21], they got 81.7% in the Random forest for Jm1. Another paper [22] had their highest accuracy, also 80% in Decision Tree, but we got 81%, which was a little ahead of them. Now, let's talk about Cm1; we got 91% highest accuracy in both Decision Tree and Random Forest. But paper[22] got 88% in Random forest for Cm1 and paper [22] got 90% in Decision for Cm1 dataset. Next, we get the highest accuracy in Pc1 is 97% in both Random forest and Adaboost. in paper [19], they got the highest accuracy for pc1 is 93% for Random Forest and SVM. In the meantime, the next team [21] also got good accuracy, which is 94% in Random Forest for this dataset and in paper [20], they got the highest accuracy, 93% for this dataset in Decision Tree. From all of these analyses, we can conclude that the Random forest always gives the best accuracy for those types. We also use another dataset called a unified dataset, and no one still did not do any work on this dataset, and we also got 92% accuracy for the Random forest.

Table 5.3: comparison of related paper AUC with ours AUC

Algorithms	Dataset	Our Work	Paper[20]	Paper[21]
Random Forest	Jm1	0.947	0.754	
	Pc1	0.993	0.86	0.618
	Cm1	0.977	0.764	0.573
Adaboost	Jm1	0.951		
	pc1	0.994		
	Cm1	0.970		
KNN	Jm1	0.879		
	Pc1	0.944		0.5
	Cm1	0.886		0.5
Decision Tree	Jm1	0.819	0.658	
	Pc1	0.907	0.82	
	Cm1	0.817	0.746	
XGBoost	Jm1	0.886		
	Pc1	0.984		
	Cm1	0.975		
MLP	Jm1	0.724		
	Pc1	0.931		
	Cm1	0.908		
SVM	Jm1	0.657		
	Pc1	0.765		0.79
	Cm1	0.851		0.753
Naïve Bayes	Jm1	0.689	0.679	
	Pc1	0.679	0.668	0.781
	Cm1	0.775	0.668	0.734

In paper [20] and [26], we compare their AUC with ours. By using the random forest in the jm1 dataset, we got 0.947, in pc1 dataset we got 0.9933 in cm1 dataset we got 0.977 and paper [20], in jm1 dataset they got 0.754, in the pc1 dataset they got 0.86, in the cm1 dataset, they got 0.764, and in paper [26], in the pc1 dataset, they got 0.618. In the cm1 dataset, they got 0.573. We got the best AUC value from these two papers. Using the decision tree algorithm in the jm1 dataset, we got 0.819 in the pc1 dataset; we got 0.907; in the cm1 dataset, we got a total of 0.817. in paper [20], in jm1 dataset, they got 0.658, in pc1 dataset, they got 0.82, in the cm1 dataset they got 0.746. This time also we got the best AUC value. By using Naive Bayes in the jm1 database, we got 0.689; in the pc1 database, we got 0.679; in the cm1 database, we got 0.775. On the other hand, in paper [20] using the jm1 dataset, they got 0.679; in the pc1 dataset, they got 0.668; in the cm1 dataset, they got 0.668. In paper [26], in the pc1 dataset, they got 0.781, and in the cm1 dataset, they got 0.734.

Chapter 6

Conclusion and Future Plan

This paper we have used six classifier machine learning algorithms and two boosting algorithms (K nearest neighbours, Random Forest, SVM, Naïve Bayes, MLP, Decision Tree, AdaBoost, XGBoost) to identify the best accuracy based on Unified Dataset and promise dataset. We have found 92% accuracy on the Random Forest and AdaBoost Algorithm for Unified Dataset, which is maximum from other existing papers. We have found 90% accuracy on the Random Forest and AdaBoost Algorithm for JM1, which is maximum from other existing papers. We have found 91% accuracy on the Random Forest and Decision Tree Algorithm for CM1, which is maximum from other existing papers. We have found 97% accuracy on the Random Forest and AdaBoost Algorithm for PC1, which is maximum from other existing papers. We have used different preprocessing libraries and our programming knowledge to preprocess our dataset. We have shown our working accuracy through the ROC curve and analyzed different algorithm accuracy using the comparison table. We have tried our level best to measure the maximum accuracy at this stage. We will identify the bug of software in the shortest possible time in our future work and implement a hybrid algorithm to measure the maximum accuracy from any promise dataset and unified dataset. Besides this, it will predict the bug of software, and based on these characteristics, it will show the dataset's output, whether it is efficient. Our goal is to implement such a hybrid algorithm model, which will work for any promise dataset to show the maximum accuracy. For this, we will be working on several datasets in the future.

Bibliography

- [1] R. L. Hotz, “Mars probe lost due to simple math error,” *Los Angeles Times*, vol. 1, 1999.
- [2] D. N. Arnold, “The explosion of the ariane 5,” 2000.
- [3] “3 of the worst it disasters in history,” 2016.
- [4] S. Kettmann, “Soviets burned by cia hackers,” *Wired News (www.wired.com)*, 2004.
- [5] C. Decker and R. Wattenhofer, “Bitcoin transaction malleability and mtgox,” in *European Symposium on Research in Computer Security*, Springer, 2014, pp. 313–326.
- [6] H. of Commons Transport Committee *et al.*, “The opening of heathrow terminal 5,” *London, House of Commons*, 2008.
- [7] J. MACNEIL, “Mariner 1 destroyed due to code error, july 22, 1962,” 2019.
- [8] L. Milner, “The morris worm, 30 years since first major attack on the internet,” 2018.
- [9] Y.-G. Guéhéneuc, “Département d’informatique et de recherche opérationnelle université de montréal, québec, canada guehene@ iro. umontreal. ca hiver 2008,” 2008.
- [10] L. Milner, “British airways passengers stranded after it failures,” 2019.
- [11] K. Dejaeger, T. Verbraeken, and B. Baesens, “Toward comprehensible software fault prediction models using bayesian network classifiers,” *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 237–257, 2012.
- [12] P. S. Bishnu and V. Bhattacharjee, “Software fault prediction using quad tree-based k-means clustering algorithm,” *IEEE Transactions on knowledge and data engineering*, vol. 24, no. 6, pp. 1146–1150, 2011.
- [13] S. D. Immaculate, M. F. Begam, and M. Floramary, “Software bug prediction using supervised machine learning algorithms,” in *2019 International Conference on Data Science and Communication (IconDSC)*, IEEE, 2019, pp. 1–7.
- [14] H. Khosrowjerdi, K. Meinke, and A. Rasmusson, “Virtualized-fault injection testing: A machine learning approach,” in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, IEEE, 2018, pp. 297–308.
- [15] M. R. Lyu, “Department of computer science and engineering the chinese university of hong kong,” *Department of Computer Science and Engineering, CUHK*, vol. 20, 2010.

- [16] Y. Tamura, S. Ashida, and S. Yamada, “Fault identification tool based on deep learning for fault big data,” in *2016 International Conference on Information Science and Security (ICISS)*, IEEE, 2016, pp. 1–4.
- [17] A. Kaur, P. S. Sandhu, and A. S. Bra, “Early software fault prediction using real time defect data,” in *2009 Second International Conference on Machine Vision*, IEEE, 2009, pp. 242–245.
- [18] L. Chen, B. Fang, and Z. Shang, “Software fault prediction based on one-class svm,” in *2016 International Conference on Machine Learning and Cybernetics (ICMLC)*, IEEE, vol. 2, 2016, pp. 1003–1008.
- [19] P. Singh and S. Verma, “Multi-classifier model for software fault prediction.,” *Int. Arab J. Inf. Technol.*, vol. 15, no. 5, pp. 912–919, 2018.
- [20] M. Akour, I. Alsmadi, and I. Alazzam, “Software fault proneness prediction: A comparative study between bagging, boosting, and stacking ensemble and base learner methods,” *International Journal of Data Analysis Techniques and Strategies*, vol. 9, no. 1, pp. 1–16, 2017.
- [21] G. P. Bhandari and R. Gupta, “Machine learning based software fault prediction utilizing source code metrics,” in *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, IEEE, 2018, pp. 40–45.
- [22] P. D. Singh and A. Chug, “Software defect prediction analysis using machine learning algorithms,” in *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*, IEEE, 2017, pp. 775–781.
- [23] J. Sayyad Shirabad and T. Menzies, *The PROMISE Repository of Software Engineering Databases*. School of Information Technology and Engineering, University of Ottawa, Canada, 2005. [Online]. Available: <http://promise.site.uottawa.ca/SERepository>.
- [24] F. Rudolf, T. Zoltán, L. Gergely, S. István, and G. Tibor, “A public unified bug dataset for java and its assessment regarding metrics and bug prediction,” *Software Quality Journal*, vol. 28, no. 4, pp. 1447–1506, 2020.
- [25] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, “A public unified bug dataset for java,” in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2018, pp. 12–21.
- [26] R. S. Wahono, N. S. Herman, and S. Ahmad, “A comparison framework of classification models for software defect prediction,” *Advanced Science Letters*, vol. 20, no. 10-11, pp. 1945–1950, 2014.