REPORT ON
COMPARATIVE ANALYSIS OF SOFTWARE DEFECT PREDICTION USING ENSEMBLE TECHNIQUES

by

Ujjwal Gupta (1809113115)

Himanshi Kumari Gupta (1900910139004)

Madhurika Sharma (1900910139005)

Supriya Kumari Gupta (1900910139008)

Department of Information Technology

JSS Academy of Technical Education

C-20/1, Sector-62, Noida - 201301

MAY 2022

COMPARATIVE ANALYSIS OF ENSEMBLE TECHNIQUES FOR SOFTWARE DEFECT PREDICTION

Submitted by

Ujjwal Gupta (1809113115)

Himanshi Kumari Gupta (1900910139004)

Madhurika Sharma (1900910139005)

Supriya Kumari Gupta (1900910139008)

Under the Supervision of

MR. BIRENDRA KUMAR VERMA

Submitted to the Department of Information Technology
in partial fulfillment of the requirements for the degree of
Bachelor of Technology
in
Information Technology

JSS Academy of Technical Education

C-20/1, Sector-62, Noida - 201301

Dr. A.P.J. Abdul Kalam Technical University, Uttar Pradesh,Lucknow

MAY, 2022

DECLARATION

We hereby declare that this submission is our work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material that to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature:

Name: Ujjwal Gupta

RollNo.: 1809113115

Date: 25/05/2022

Signature:

Name: Himanshi

Kumari Gupta

RollNo:1900910139004

Date: 25/05/2022

Signature:

Name: Madhurika Sharma

Roll No.:1900910139005

Date: 25/05/2022

Signature:

Name: Supriya Kumari Gupta

Roll No.:1900910139007

Date: 25/05/2022

CERTIFICATE

This is to certify that the Project Report entitled “**COMPARATIVE ANALYSIS OF SOFTWARE DEFECT PREDICTION USING ENSEMBLE LEARNING TECHNIQUES**” which is submitted by Ujjwal Gupta, Himanshi Kumari Gupta, Madhurika Sharma, and Supriya Kumari Gupta in partial fulfillment of the requirement for the award of degree B.Tech in the Department of Information Technology of Dr. A.P.J. Abdul Kalam Technical University, Uttar Pradesh, Lucknow is a record of the candidate’s work carried out by him/her under my/our supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

SUPERVISOR:
Mr BIRENDRA KUMAR VERMA
Associate Professor, IT Department, JSSATE

Date: 25/05/2022

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the report of the B. Tech Project undertaken during B. Tech Final Year. We owe a special debt of gratitude to Associate Professor Birendra Verma, Department of Information Technology, JSSATE, Noida for her constant support and guidance throughout our work. Her sincerity, thoroughness, and perseverance have been a constant source of inspiration for us. It is only through her cognizant efforts that our endeavors have enlightened the day.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution to the completion of the project.

Name: Ujjwal Gupta
Roll No.: 1809113115
Date: 27/05/2022

Name: Himanshi Kumari Gupta
Roll No.: 190910139004
Date: 27/05/2022

Name: Madhurika Sharma
Roll No.: 1900910139005
Date: 27/05/2022

Name: Supriya Kumari Gupta
Roll No.: 190910139007
Date: 27/05/2022

ABSTRACT

In today’s world, we can see our daily life full of software, so it's of maximum probability that the software might contain some bugs. Software Defect Prediction (SDP) is predicting all the technical as well as front end, and back end bugs.

All we have to predict is bugs using various Ensemble Learning techniques. Software Defect Prediction (SDP) is one of the most helpful tools in the Testing Phase of the Software Development Life Cycle (SDLC). It identifies the modules that are prone to defects and require testing.

Along these lines, the testing assets can be utilized proficiently without abusing the constraints. However, SDP is extremely useful in testing, foreseeing the faulty modules is difficult all the time. Late advances in the area of Software Defect Prediction (SDP) incorporate the coordination of numerous characterization strategies to make a group or half and half methodology. This procedure was acquainted with further developing the forecast exhibition by beating the constraints of any single arrangement strategy. Basically Ensemble methods are classified into two types based on the base learners, first, one is Homogeneous ensemble methods such as Bagging, and Boosting, and the second one is Heterogeneous ensemble methods such as Stacking, Super Learner. In homogenous ensemble methods, the same base learners are applied to a different set of instances in a dataset using various classifiers. Examples include bagging, boosting, rotation forest, etc. And using Stacking We achieved an accuracy of 90.3% on the Lucene dataset.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SDP	Software Defect Prediction
SQA	Software Quality Assurance
SDLC	Software Development Life Cycle
SVM	Support Vector Machine
ML	Machine Learning
SMOTE	Synthetic Minority Oversampling Technique
DT	Decision Tree
BN	Bayes Network
AUC	Area Under Curve
RF	Extra Trees

CHAPTER - 1

INTRODUCTION

1.1 INTRODUCTION

The most valuable and cost-effective activity in programming is defect prediction. It is regarded as a crucial stage in the development process that defines the quality of the final product by software professionals. It has been critical in lowering the costs connected with the product company, such as the failure to meet demands on time and budget. SQA is a procedure that comprises tracking and regulating the development life cycle to maintain software quality while minimizing costs. This type of work includes structured code audits, code walkthroughs, software testing, and software fault prediction, to name a few elements.

The capacity of the product to fulfill its necessary duty under specified conditions for a certain timeframe," according to the definition of software reliability. Software reliability is often difficult to accomplish because of the rapid expansion and increasing complexity of software. One of the most fundamental and crucial parts of software quality is reliability.

Software reliability, according to ANSI, is "the probability of failure-free operation of a computer program for a particular period of time in a specified environment." (Quyoum and colleagues, 2010). The model was used to predict and estimate the number of errors in the programme in this study.

The main purpose of software reliability modeling is to determine the likelihood of a system failure in a given time interval or the expected time interval between subsequent failures. Machine Learning (ML) techniques have shown to be more accurate and precise in forecasting results than statistical methods and can be used to better predict software failures. ML is a machine learning approach that allows computers to automatically learn and predict system behavior based on past and current failure data. As a result, it's only natural to want to know which strategy works best for a certain failure dataset and to what extent statistically. (Aggarwalet al. 2006; Goel and Singh et al. 2009).

To forecast and evaluate software models based on their performance, an ensemble learning approach is applied in this study. To estimate the period between software failures, various machine learning algorithms are applied. Different models are trained on input data and then used to forecast values based on test data in prediction work. The most difficult aspect of the prediction problem is determining the performance of the models using various error measuring approaches. Using a single model to forecast data may not always produce superior results. To determine the next failure time, various regression models are utilized, and the error measurements of the various models are compared. Furthermore, the goal is to offer a model by merging several models. The goal is also to provide a model that improves performance on many datasets by merging several models.

Software Defect Prediction (SDP) is a highly cost-effective and valuable procedure in the realm of software engineering. Before the release of a new product, such approaches would focus on directing software testing activities by estimating potential flaws in code.

Software defect prediction is a way of constructing machine learning classifiers to forecast incorrect code snippets by using historical data from computer repositories such as code complexity and change in records to model software metrics flaws. Program faults are common in sophisticated source code, and these flaws can lead to software failure.

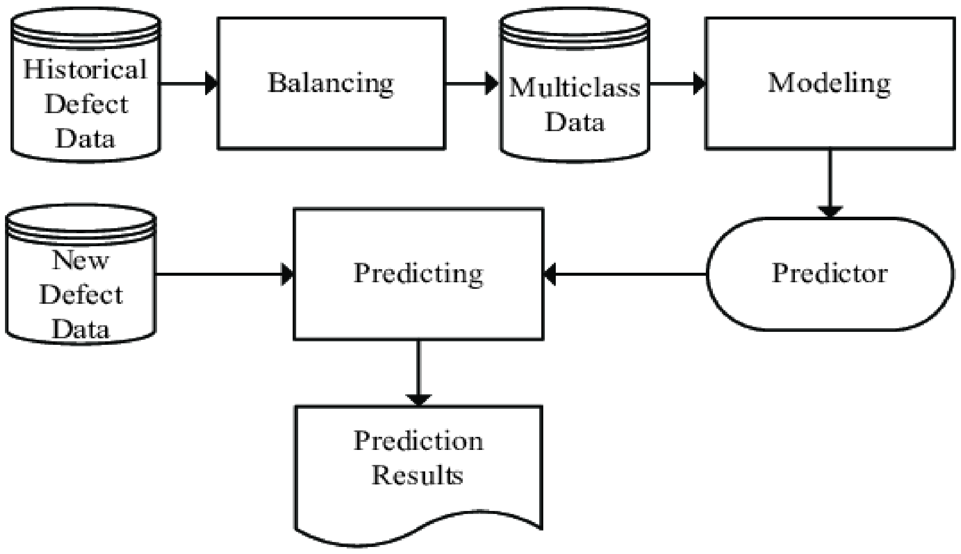


Fig 1.1: Software defect prediction method

The ensemble learning model is created by integrating many machine learning classifiers to increase prediction performance. Ensemble learning is described in the literature using a variety of names, including hybrid, combined, integrated, and aggregated classification. To develop the prediction model on a pre-labeled dataset, the classic technique of defect prediction uses an individual classifier, such as the naive Bayes classifier, decision trees, or a multilayer perceptron, as shown in Fig 1.1.

Under certain circumstances, individual classifiers may have flaws in predicting a given fault. As a result, ensemble learning was employed to combine the strengths of many classifiers in order to improve defect detection in the dataset. In the recent decade, a slew of studies has shown that ensemble approaches outperform individual classifiers in terms of classification accuracy.

Ensemble methods are broadly classified into two types based on the types of base learners:

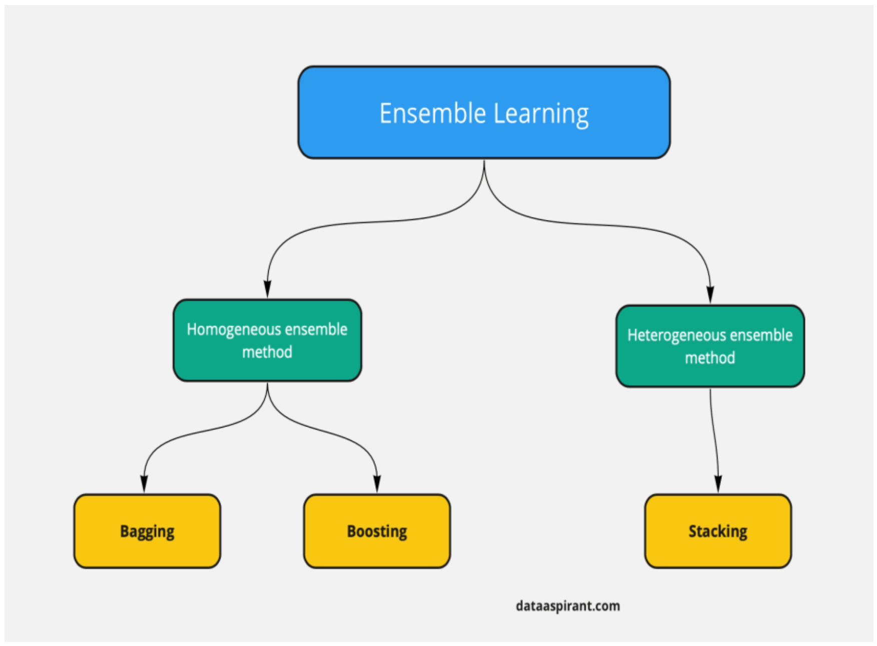
Hybrid ensemble methods

Hexagonal ensemble methods

1. The same base learners are applied to a different set of instances in a dataset in homogenous ensemble methods. Bagging, boosting, and forest rotation are a few examples.

2. Different base learners are generated using different machine learning techniques in the heterogeneous ensemble method. These base learners are combined, and the final prediction is performed by statistically or democratically integrating the results of the base learners.

3. Because of the different natures of base learners, heterogeneous methods are more diverse than homogeneous methods.



[Fig 1.2: ENSEMBLE TECHNIQUES]

BAGGING: Bagging, also known as bootstrap aggregation, is an ensemble learning method for reducing variance in a noisy dataset. It is the process of choosing a random sample of data from a training set with replacement—that is, the individual data points might be chosen many times.

BOOSTING: It is an ensemble modeling strategy that aims to create a strong classifier from a large number of weak ones. It is accomplished by constructing a model using a sequence of weak models. To begin, the training data is used to create a model. The second model is then created, which attempts to remedy the original model's flaws. This approach is repeated until either the entire training data set is properly predicted or the maximum number of models is added.

STACKING: It is a way of combining diverse ensemble approaches. Stacking is an ensemble strategy that builds a new model from predictions from numerous models (for example, decision tree, KNN, or SVM). When the outcomes of the individual algorithms are considerably varied, stacking comes in handy.

1.2 MOTIVATION

It can be seen that on different techniques and different datasets we observe different performance metrics and accuracy. So to get a perfect and accurate idea of accuracy and outcome we need to work on the common dataset and common performance metrics and propose an ensemble-learning algorithm for the software defect classification problem.
Various other motivations are listed below:

Statistical:

When the amount of training data is too tiny compared to the size of the search area, the learning algorithm produces the same accuracy on the training data despite the fact that the hypotheses are different. The program may average the accurate hypotheses, lower the danger of selecting the incorrect classifier, and obtain a good approximation to the genuine hypothesis by applying ensemble methods.

Computational:

The computational problem arises when training data is enough but the learning algorithm is still stuck. So it's still very difficult computationally for the learning algorithm to find the best hypothesis. An ensemble constructed by running the local search from many different starting points may provide a better approximation to the true unknown function than any of the individual classifiers.

Representational:

The function can be expressed in most machine learning systems by producing weighted sums of hypotheses. It may be possible to broaden the range of functions that can be represented in this way. As a result, for a given training dataset, we must consider the chosen search space to be the effective space of hypotheses examined by the learning algorithm.

1.3 PROJECT OBJECTIVE:

The objective of this project is to make a Comparative Analysis of several Ensemble Techniques for Software Defect Prediction to get the best accuracy.

1.4 SCOPE:

Instead of employing a single feature selection approach, ensembles of feature selection methods aim to mix numerous feature selection methods. Experiments show that ensembles with a small number of rankers are as effective as or better than ensembles with a large number of rankers. Although there are various Ensemble learning approaches, we are only allowed to work with three of them: Homogeneous Ensemble techniques like Bagging and Boosting, and Heterogeneous Ensemble techniques like Stacking.

1.5 RELATED WORK:

[1] Balogun et al. To increase the performance of defect prediction models, a method employing SMOTE (synthetic minority oversampling Technique) and homogeneous ensemble approaches (bagging and boosting) were presented. In their model, they used DT (Decision Tree) and BN (Bayes Network) as baseline classifiers. Their experiments revealed that the proposed technique outperformed base classifiers by a significant margin. The proposed defect prediction model's performance is measured using the AUC, Accuracy, and F-measure in this method. NASA software programs were used to create 5 datasets. The performance of DT and BN were compared to a SMOTE-based homogeneous ensemble approach on five projects with a homogeneous set of metrics. Bagging and boosting ensembles were used, with DT and BN as basic classifiers.

Alazba et al. [2] In defect prediction, proposed an approach that compared the prediction performance of seven Tree-based ensembles. Random forest and Extra Trees were used as bagging ensembles, and Ada boost, Gradient Boosting, Hist Gradient Boosting, XGBoost, and Cat Boost were used as boosting ensembles. The empirical data revealed that Tree-based bagging ensembles outperformed Tree-based boosting ensembles. However, none of the Tree-based ensembles performed significantly worse than individual decision trees in terms of prediction performance. Furthermore, of all Tree-based ensembles, the Ada boost ensemble performed the lowest. Extra Trees (ET), XG- Boost, Cat Boost, Gradient Boosting, Hist Gradient Boosting, Ada-Boost, and RF were used to compare the prediction performance of individual decision tree classifiers to tree Based ensembles using 11 datasets from NASA software projects. They discovered that, except for the Ada ensemble, the decision tree classifier performed poorly for all defect datasets, whereas Tree-based ensembles consistently outperformed the decision tree classifier. RF and ET bagging ensembles outperformed all other ensembles in terms of prediction.

Tong et al. [3] used a Heterogeneous layered method with '2-levels' to forecast software defects. The training and testing of base classifiers are at level one. 10-fold Cross-Validation is used to validate the level-1 learners. Then, in this model, the level-1 prediction outputs are provided as training data to meta learner or level-2 classifier. Nasa's CM1, KC1, KC2, PC1, and JM1 datasets in the SDP The suggested defect prediction model's performance is measured using the AUC, Accuracy, and F-measure models. The suggested classifier's performance is compared to that of traditional classifiers in terms of AUC and accuracy. Artificial Neural Network Decision Tree, Naive Bayes, K-nearest neighbor, and Support Vector Machine are the most prevalent classifiers found in the studies. All traditional classifiers and proposed stacking ensembles have their AUC values recorded. The results show that stacked Ensemble outperforms all other traditional classifiers across all datasets. The heterogeneous stacking technique outperforms the traditional technique by an average of 90%.

Kaur et al. [4] The bagging, boosting, and RF(Reinforcement) ensemble approaches were compared. In ensembles, fifteen basic classifiers were used. 9 open source datasets from the

Promise repositories were used in the experiments. On base learners, AUC performance gains for bagging, boosting, and RF was attained, according to the findings. With no decrease in AUC performance, RF outperformed bagging and boosting. For bagging, boosting, and RF, NB, logistic regression, and voted feature interval learners were not recommended as base learners. In this model, the AUC, TP-rate, FP-rate, and F-measure were used to compare their performance. IBM's integrated development environment (IDE) was used. 32 software metrics linked to various source code and structural measures were included in these datasets.

Friedman et al. [5] A technique was proposed. Boosting has been used to predict software defects. Their experiments revealed that the proposed technique outperformed base classifiers by a significant margin. The proposed defect prediction model's performance was measured using the F-measure accuracy and recall in this approach. KC1 dataset Nasa 716 samples for testing 1391 samples for training Boosting works by using classification algorithms successively on reweighted versions of training data. The completion class label is based on a weighted majority vote, the overall outcome, and software complexity measures such as LOC measure, Cyclomatic complexity Boostingm5, and Decision stump.

Rathore et al. [6] Heterogeneous and homogeneous ensemble approaches, as well as linear and nonlinear combination rules, were compared. On 15 datasets, they tested six defect prediction techniques: DTR, LR, MLP, genetic programming (GP), negative binomial regression (NBR), and zero-inflated Poisson regression (ZIP) and chose the top three techniques—DTR, MLP, and LR—as base learners for ensemble models. With M5P as a base learner, the researchers evaluated the proposed heterogeneous ensemble model against homogeneous bagging and boosting ensembles. Heterogeneous ensemble methods based on nonlinear combination rules outperformed homogeneous ensemble methods for the majority of datasets. However, as compared to homogeneous ensemble methods, heterogeneous ensemble approaches based on linear combination rules did not exhibit significant differences. In comparison to linear combination rule-based ensembles, they found that nonlinear-based ensemble approaches were more stable and performed better in forecasting the number of faults.

CHAPTER 2

LITERATURE SURVEY

2.1 LITERATURE SURVEY

Several machine learning algorithms for software dependability modeling have been proposed and applied in the literature. Some of the techniques employed include generic programming, gene expression programming, artificial neural networks, decision trees, support vector machines, feed-forward neural networks, fuzzy models, generalized neural networks, and others (Malhotra et al. 2011; Xingguo and Yanhua 2007; Hua Jung 2010; Karunanithi et al. 1992; Singh and Kumar 2010d; Eduardo et al. 2010). Ho et al. (2003) investigated connectionist models and their relevance to software reliability prediction in-depth, concluding that they outperform standard methods. Su and Huang (2006) employed a neural network to predict software dependability.

[7], H. Laradji. et.al., H. Laradji. et.al., H. Laradji Researchers The impact of feature selection (FS) paired with ensemble learning on defect prediction performance was investigated, and a strategy was presented. They presented an APE (average probability ensemble) technique for defect classification. Feature selection was done in the pre-processing step using greedy forward selection (GFS) and correlation-based FS. The APE model was built using RF, gradient boosting, stochastic gradient descent, W-SVMs, logistic regression, multinomial NB, and Bernoulli naive Bayes. According to the model's evaluations on six datasets, GFS outperformed correlation-based FS, and the APE model had the highest AUC when compared to W-SVMs and RF. They also supplied a more advanced version of APE with GFS for a higher AUC.

[8] Misha Kakkar. et.al An technique based on Feature Selection was proposed for Software Defect Prediction. To measure accuracy and ROC for software defect prediction before and after preprocessing, we used cleaned versions of five datasets from the NASA MDP repository, namely CM1, JM1, KC1, and PC1, and applied them to the pre-processed dataset's specified attributes. In almost every scenario, a close review of both tables demonstrates that accuracy

has improved. The PC1 and KC1 datasets yielded maximum accuracy of 91.96 and 91.17, respectively, with ROC values of 0.8760 and 0.8050, indicating that 10-fold cross-validation is the best testing method. After attribute selection, overall computing time was decreased by roughly 80%, with greater accuracy and ROC.

Faseeha. et. al[9] present a feature selection-based ensemble classification for accurate software defect prediction in this paper. The framework is broken down into four sections:

1) Dataset selection

2) Preprocessing

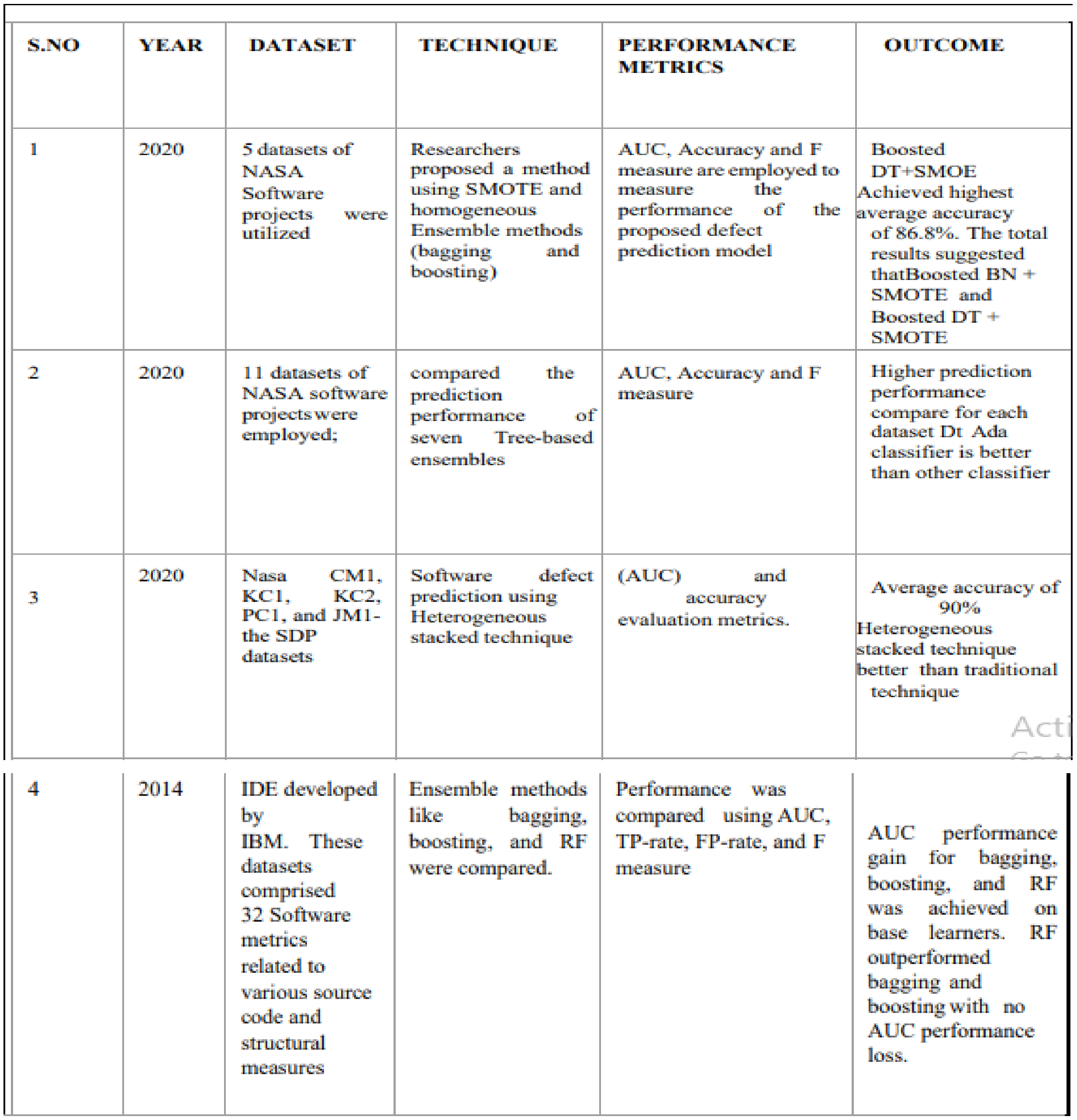
3) Classification and

4) Results Reflection

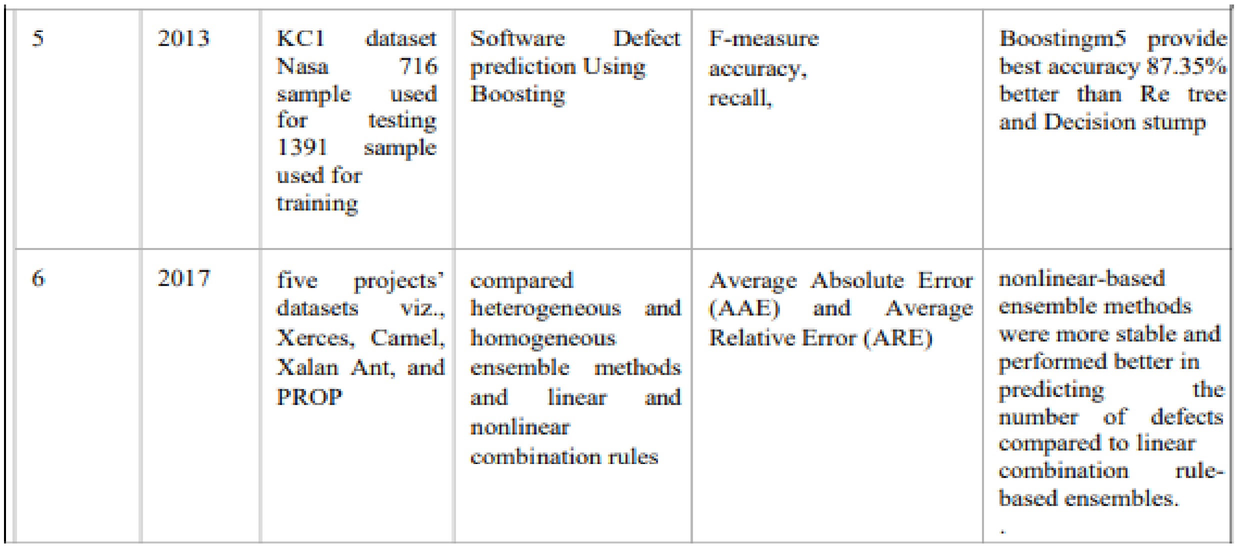
The preprocessing stage also includes three significant activities:

1) Normalization of Data
2) To remove irrelevant columns Feature Selection is to be done
3) Balancing Majority and Minority classes

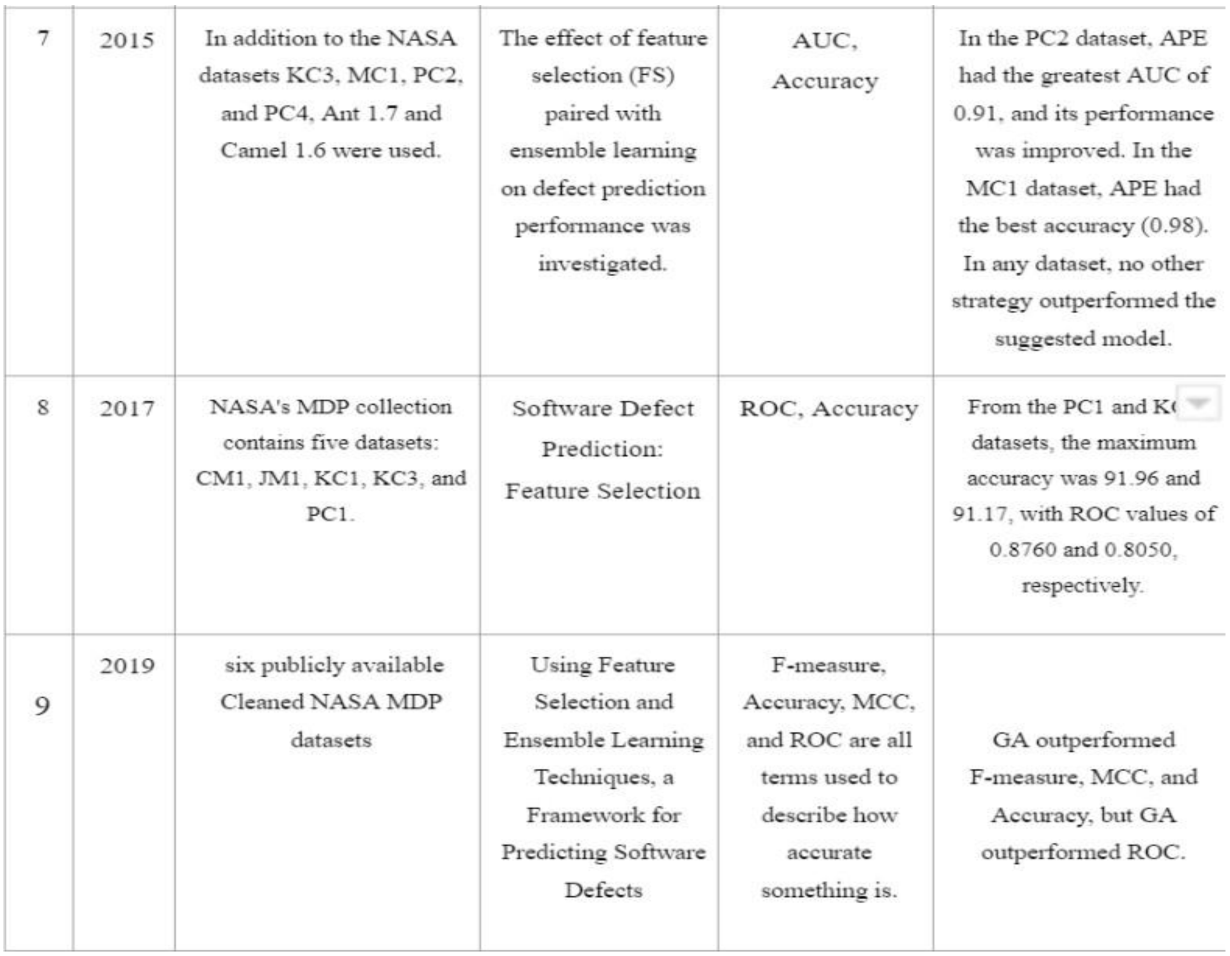
MLP is utilized as the Feature Subset Evaluator, and there are six search methods: BF, GS, GA, PSO, RS, and LFS. The suggested framework is assessed using F-measure, Accuracy, MCC, and ROC on six NASA MDP datasets that have been cleaned. The scores of all search methods within the framework are compared to each other, and the results of the proposed framework with all search methods are compared to the results of 10 well-known machine learning classifiers from a published paper in two dimensions: first, the scores of all search methods within the framework are compared to each other, and second, the results of the proposed framework with all search methods are compared to the results of 10 well-known machine learning classifiers from a paper. The suggested approach beat all other classifiers, according to the findings. Because none of the search strategies were able to outperform the standard.



S.NO	YEAR	DATASET	TECHNIQUE	PERFORMANCE METRICS	OUTCOME
1	2020	5 datasets of NASA Software projects were utilized	Researchers proposed a method using SMOTE and homogeneous Ensemble methods (bagging and boosting)	AUC, Accuracy and F measure are employed to measure the performance of the proposed defect prediction model	Boosted DT+SMOE Achieved highest average accuracy of 86.8%. The total results suggested thatBoosted BN + SMOTE and Boosted DT + SMOTE
2	2020	11 datasets of NASA software projects were employed;	compared the prediction performance of seven Tree-based ensembles	AUC, Accuracy and F measure	Higher prediction performance compare for each dataset Dt Ada classifier is better than other classifier
3	2020	Nasa CM1, KC1, KC2, PC1, and JM1- the SDP datasets	Software defect prediction using Heterogeneous stacked technique	(AUC) and accuracy evaluation metrics.	Average accuracy of 90% Heterogeneous stacked technique better than traditional technique
4	2014	IDE developed by IBM. These datasets comprised 32 Software metrics related to various source code and structural measures	Ensemble methods like bagging, boosting, and RF were compared.	Performance was compared using AUC, TP-rate, FP-rate, and F measure	AUC performance gain for bagging, boosting, and RF was achieved on base learners. RF outperformed bagging and boosting with no AUC performance loss.



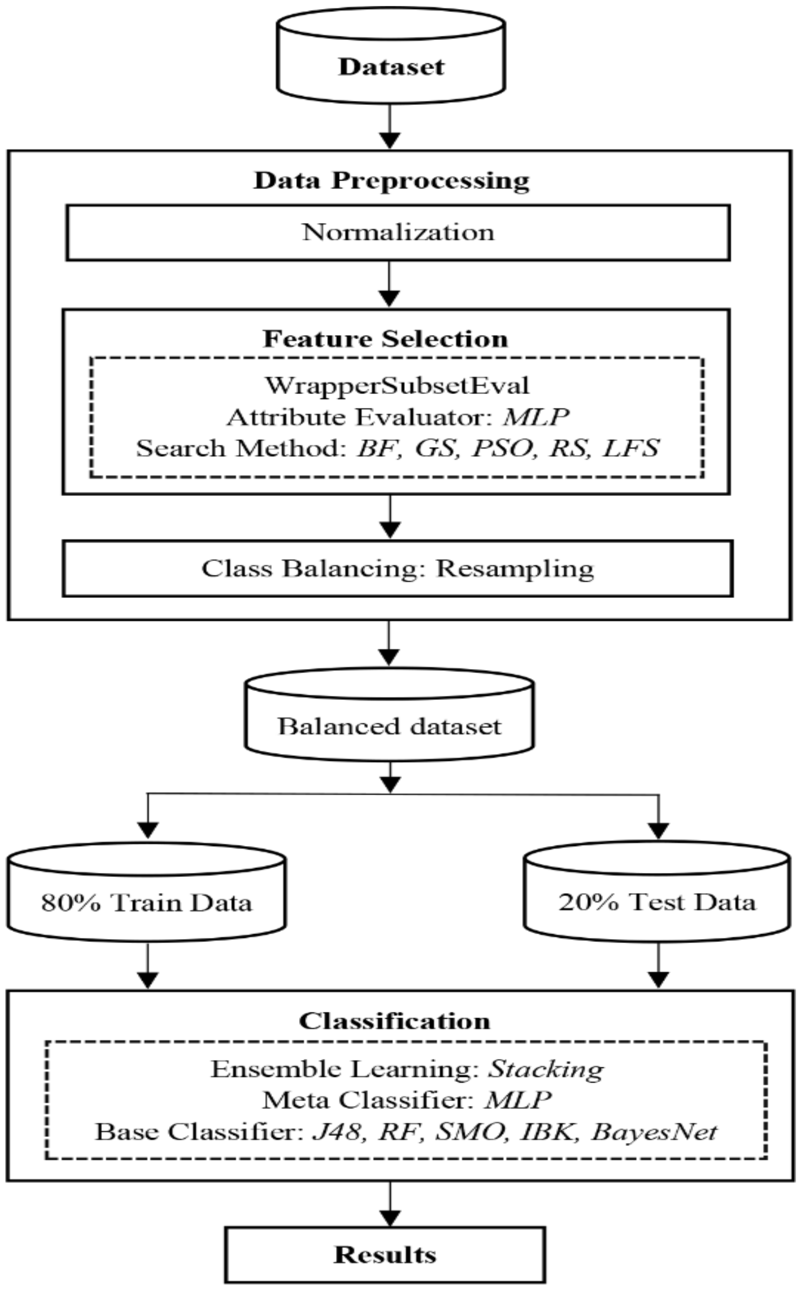
5	2013	KC1 dataset Nasa 716 sample used for testing 1391 sample used for training	Software Defect prediction Using Boosting	F-measure accuracy, recall,	Boostingm5 provide best accuracy 87.35% better than Re tree and Decision stump
6	2017	five projects' datasets viz., Xerces, Camel, Xalan Ant, and PROP	compared heterogeneous and homogeneous ensemble methods and linear and nonlinear combination rules	Average Absolute Error (AAE) and Average Relative Error (ARE)	nonlinear-based ensemble methods were more stable and performed better in predicting the number of defects compared to linear combination rule-based ensembles. .



7	2015	In addition to the NASA datasets KC3, MC1, PC2, and PC4, Ant 1.7 and Camel 1.6 were used.	The effect of feature selection (FS) paired with ensemble learning on defect prediction performance was investigated.	AUC, Accuracy	In the PC2 dataset, APE had the greatest AUC of 0.91, and its performance was improved. In the MC1 dataset, APE had the best accuracy (0.98). In any dataset, no other strategy outperformed the suggested model.
8	2017	NASA's MDP collection contains five datasets: CM1, JM1, KC1, KC3, and PC1.	Software Defect Prediction: Feature Selection	ROC, Accuracy	From the PC1 and K(datasets, the maximum accuracy was 91.96 and 91.17, with ROC values of 0.8760 and 0.8050, respectively.
9	2019	six publicly available Cleaned NASA MDP datasets	Using Feature Selection and Ensemble Learning Techniques, a Framework for Predicting Software Defects	F-measure, Accuracy, MCC, and ROC are all terms used to describe how accurate something is.	GA outperformed F-measure, MCC, and Accuracy, but GA outperformed ROC.

Table 2.1: Related Previous Work

2.2 FEATURE SELECTION



Dataset
Data Preprocessing
Normalization
Feature Selection
WrapperSubsetEval
Attribute Evaluator: *MLP*
Search Method: *BF, GS, PSO, RS, LFS*
Class Balancing: Resampling
Balanced dataset
80% Train Data
20% Test Data
Classification
Ensemble Learning: *Stacking*
Meta Classifier: *MLP*
Base Classifier: *J48, RF, SMO, IBK, BayesNet*
Results

Fig 2.1: Random Forest Feature Selection.

The purpose of feature selection is to improve data structure so that classification can produce better results. To execute ensemble learning, the classification step employs techniques such as stacking,' 'bagging,' and 'boosting.' The performance of the proposed framework is assessed using several measures, including the F-measure, accuracy, MCC, and ROC. The outcomes of the proposed framework are evaluated in two ways: first, the performance of search algorithms on each dataset is evaluated within the framework by comparing them across all accuracy calculations. Second, the outputs are compared to all of the suggested framework's findings (including all search methods).

Area Under Curve (AUC): Area Under Curve the ROC (Receiver Operating Characteristics) curve simply shows the tradeoff between True +ve and False +ve rates.

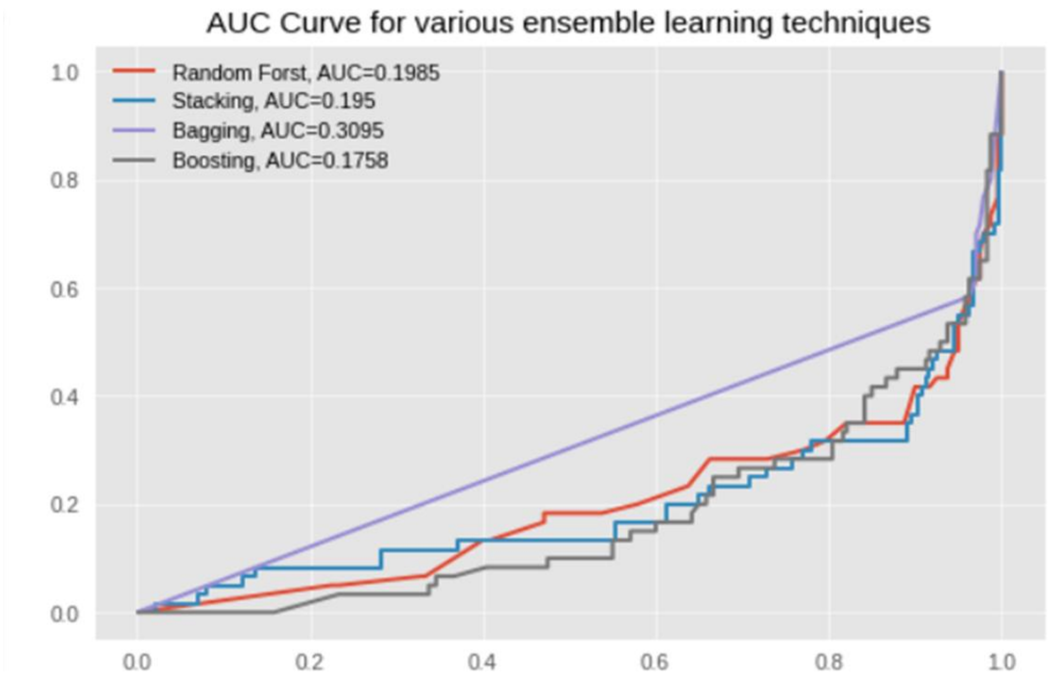


Fig 2.2: Area Under Curve for various Ensemble Learning techniques

CHAPTER – 3

SYSTEM DESIGN AND METHODOLOGY

3.1 SYSTEM ARCHITECTURE

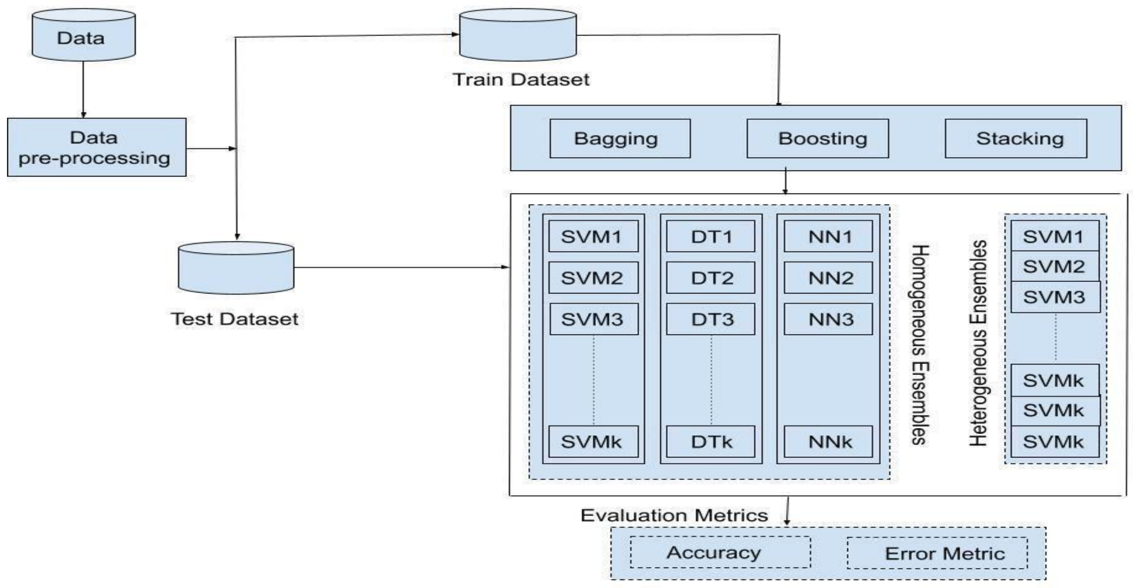


Fig 3.1: Structure of Ensemble Technique.

1. By focusing on evaluation criteria, simulation tools, and datasets, the goal of this work is to find, analyze, and summarise empirical evidence about the efficacy of ensemble learning approaches for software fault prediction.
2. This research will provide concise information on the most recent trends and developments in ensemble learning for software defect prediction, as well as a foundation for future innovations and reviews.
3. Random forest, boosting, and bagging are among the most commonly used ensemble methods, according to our research. Stacking, voting, and Extra Trees are some of the

less commonly used ways.

4. Here, we have used three different modules or techniques to find the best accuracy.

1. Bagging Ensemble Technique
2. Boosting Ensemble Technique
3. Stacking Ensemble Technique

3.2 METHODOLOGY

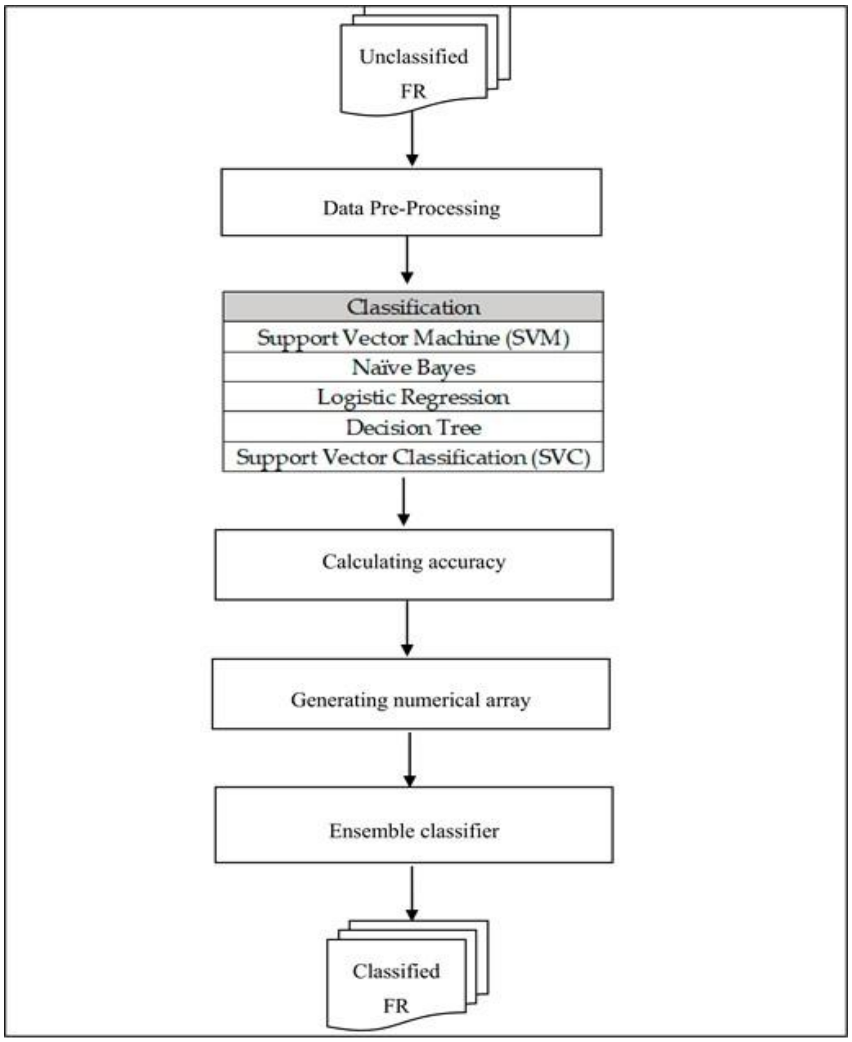


Fig 3.2: Architecture of the Ensemble Technique

The architecture illustrated in **Figure 3.2** consists of several elements:

- Cleaning and Preprocessing of data.
- Feature Selection for removing irrelevant features in the dataset.
- SVM, Naive Bayes, SVC, Decision Tree, and Logistic Regression are examples of ML base classifiers.
- For each basic classifier, create a confusion matrix.
- Calculating each base classifier's accuracy.

- The process of creating a numerical array.
- classifier in a group (the accuracy per class as per Ensemble Techniques).

3.3 ALGORITHMS

1. Bagging Ensemble Technique:

Step 1: From the original data set, multiple subsets with equivalent tuples are generated, with observations replaced.
Step 2: For each of these subgroups, a base model is generated.
Step 3: Each model is learned independently and in parallel from each training set.
Step 4: Based on the outputs from all models predictions will be done.

2. Boosting Ensemble Technique:

Step 1: Create a new dataset and give each data point an equal weight.
Step 2: Input this into the model and look for data points that have been incorrectly categorized.
Step 3: Increase the weight of the data points that were incorrectly categorized.
Step 4: If the requisite findings have been obtained, proceed to step 5; otherwise, proceed to step 6.
Go to the second step.
Step 5: Finish

3. Stacking Ensemble Technique:

Step 1: Similar to K-fold cross-validation, we partition the training data into K-folds.
Step 2: The K-1 parts are fitted using a base model, and predictions for the Kth part are made.
Step 3: We repeat this process for each segment of the training data.
Step 4: The base model is then fitted to the entire train data set to determine how well it performs on the test set.
Step 5: We repeat the previous three stages for the remaining base models.
Step 6: The train set predictions are used as features in the second level model.
Step 7: On the test set, the second-level model is utilized to produce a prediction.

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 SOFTWARE AND HARDWARE REQUIREMENTS:

Software:

1) Python 3.7 environment.
2) A user-friendly platform like GOOGLE COLAB enables easy execution of code.
3) Libraries like Scikit-learn, Matplotlib, Seaborn must be included in environment.
4) All types of classifiers support ensemble models to some extent.

Hardware:

1. A laptop with an i5 processor.
2. A min of 8 GB or more RAM is necessary.
3. A Win 64-bit or LINUX OS.

4.2 DATASET

1. AEEEM:

a. **EQ:** Shape of Dataset = (324, 62)

b. **JDT:** Shape of Dataset = (997, 62)

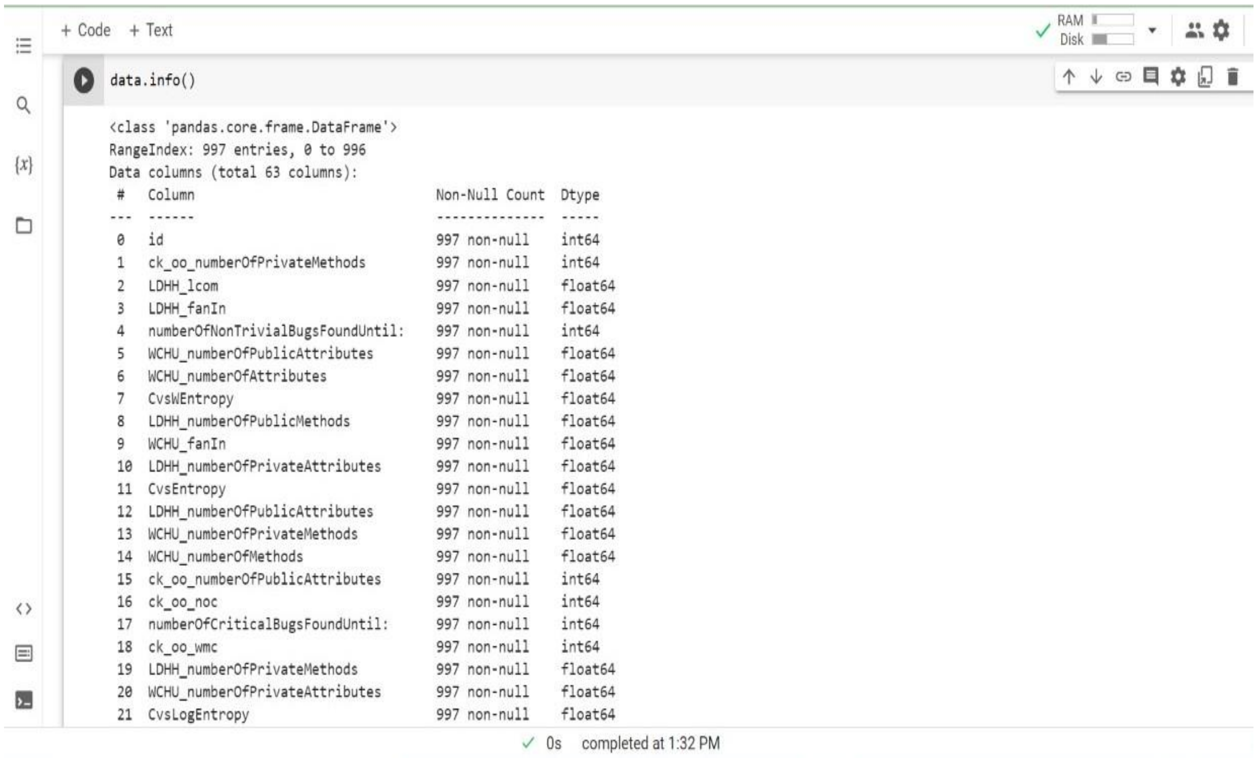
c. **LUCENE:** Shape of Dataset = (691, 62)

d. **MYLYN:** Shape of Dataset = (1862, 62)

e. **PDE:** Shape of Dataset = (1497, 62)

2. RELINK:

1. **APACHE:** Shape of Dataset = (194, 27)
2. **SAFE:** Shape of Dataset = (56, 27)
3. **ZXING:** Shape of Dataset = (399, 27)



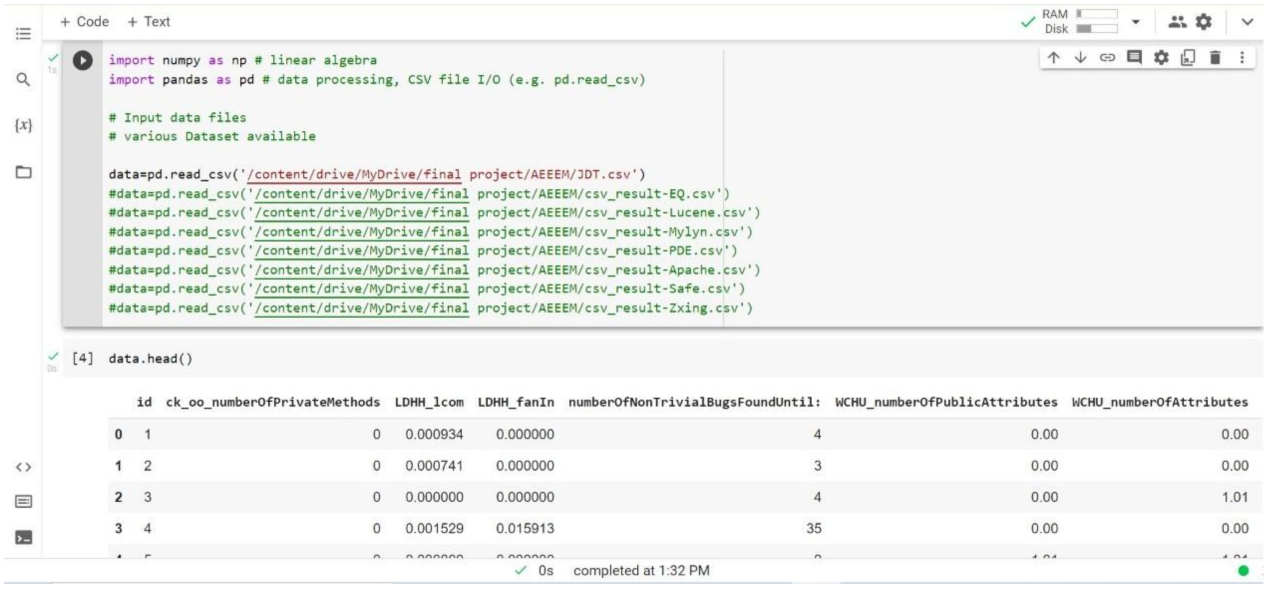
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 997 entries, 0 to 996
Data columns (total 63 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   id                                  997 non-null    int64
 1   ck_oo_numberOfPrivateMethods        997 non-null    int64
 2   LDHH_lcom                           997 non-null    float64
 3   LDHH_fanIn                          997 non-null    float64
 4   numberOfNonTrivialBugsFoundUntil:   997 non-null    int64
 5   WCHU_numberOfPublicAttributes       997 non-null    float64
 6   WCHU_numberOfAttributes             997 non-null    float64
 7   CvsWEntropy                         997 non-null    float64
 8   LDHH_numberOfPublicMethods          997 non-null    float64
 9   WCHU_fanIn                          997 non-null    float64
 10  LDHH_numberOfPrivateAttributes      997 non-null    float64
 11  CvsEntropy                          997 non-null    float64
 12  LDHH_numberOfPublicAttributes       997 non-null    float64
 13  WCHU_numberOfPrivateMethods         997 non-null    float64
 14  WCHU_numberOfMethods                997 non-null    float64
 15  ck_oo_numberOfPublicAttributes      997 non-null    int64
 16  ck_oo_noc                           997 non-null    int64
 17  numberOfCriticalBugsFoundUntil:     997 non-null    int64
 18  ck_oo_wmc                           997 non-null    int64
 19  LDHH_numberOfPrivateMethods         997 non-null    float64
 20  WCHU_numberOfPrivateAttributes      997 non-null    float64
 21  CvsLogEntropy                       997 non-null    float64
```

Fig 4.1: Description of Dataset used in the project

4.3 IMPLEMENTATION:

1. Snapshots of Interfaces:



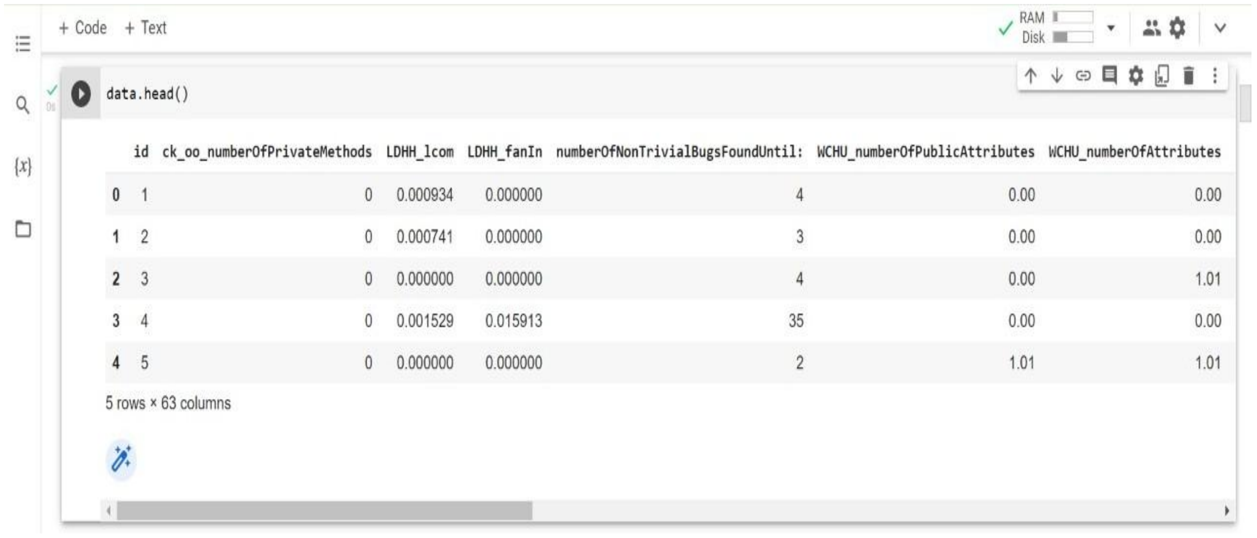
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files
# various Dataset available

data=pd.read_csv('/content/drive/MyDrive/final project/AEEEM/JDT.csv')
#data=pd.read_csv('/content/drive/MyDrive/final project/AEEEM/csv_result-EQ.csv')
#data=pd.read_csv('/content/drive/MyDrive/final project/AEEEM/csv_result-Lucene.csv')
#data=pd.read_csv('/content/drive/MyDrive/final project/AEEEM/csv_result-Mylyn.csv')
#data=pd.read_csv('/content/drive/MyDrive/final project/AEEEM/csv_result-PDE.csv')
#data=pd.read_csv('/content/drive/MyDrive/final project/AEEEM/csv_result-Apache.csv')
#data=pd.read_csv('/content/drive/MyDrive/final project/AEEEM/csv_result-Safe.csv')
#data=pd.read_csv('/content/drive/MyDrive/final project/AEEEM/csv_result-Zxing.csv')
```

```
[4] data.head()
```

	id	ck_oo_numberOfPrivateMethods	LDHH_lcom	LDHH_fanIn	numberOfNonTrivialBugsFoundUntil:	WCHU_numberOfPublicAttributes	WCHU_numberOfAttributes
0	1	0	0.000934	0.000000	4	0.00	0.00
1	2	0	0.000741	0.000000	3	0.00	0.00
2	3	0	0.000000	0.000000	4	0.00	1.01
3	4	0	0.001529	0.015913	35	0.00	0.00

✓ 0s completed at 1:32 PM

Fig 4.2: Snapshot of importing the dataset.

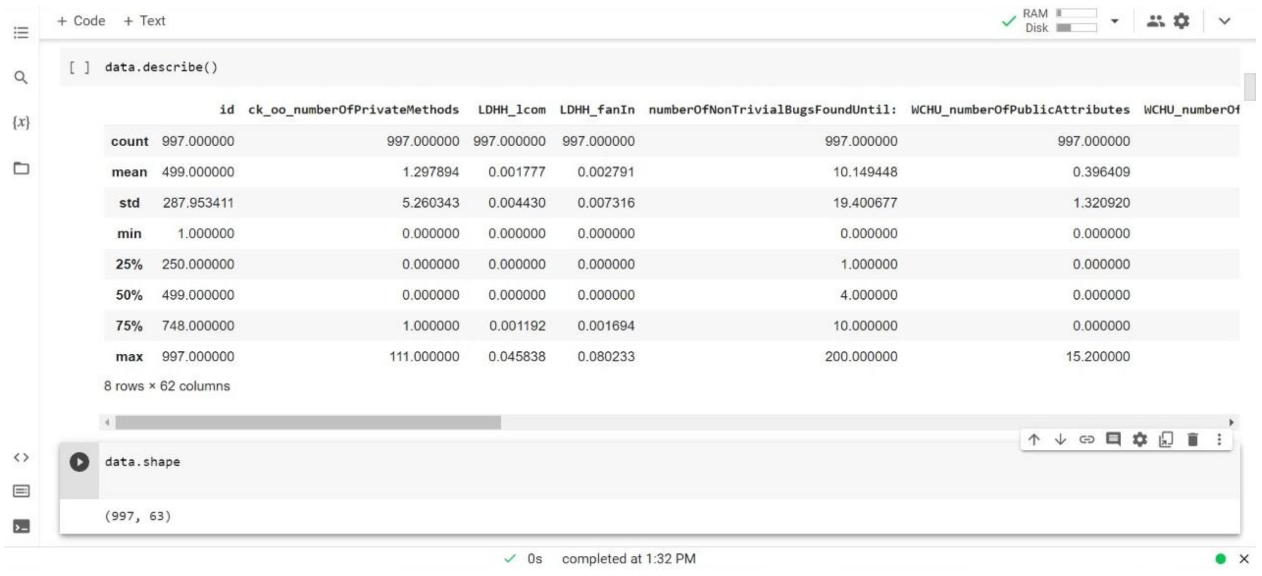


```
data.head()
```

	id	ck_oo_numberOfPrivateMethods	LDHH_lcom	LDHH_fanIn	numberOfNonTrivialBugsFoundUntil:	WCHU_numberOfPublicAttributes	WCHU_numberOfAttributes
0	1	0	0.000934	0.000000	4	0.00	0.00
1	2	0	0.000741	0.000000	3	0.00	0.00
2	3	0	0.000000	0.000000	4	0.00	1.01
3	4	0	0.001529	0.015913	35	0.00	0.00
4	5	0	0.000000	0.000000	2	1.01	1.01

5 rows × 63 columns

Fig 4.3: Preprocessing of Dataset



```
data.describe()
```

	id	ck_oo_numberOfPrivateMethods	LDHH_lcom	LDHH_fanIn	numberOfNonTrivialBugsFoundUntil:	WCHU_numberOfPublicAttributes	WCHU_numberOf
count	997.000000	997.000000	997.000000	997.000000	997.000000	997.000000	
mean	499.000000	1.297894	0.001777	0.002791	10.149448	0.396409	
std	287.953411	5.260343	0.004430	0.007316	19.400677	1.320920	
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	250.000000	0.000000	0.000000	0.000000	1.000000	0.000000	
50%	499.000000	0.000000	0.000000	0.000000	4.000000	0.000000	
75%	748.000000	1.000000	0.001192	0.001694	10.000000	0.000000	
max	997.000000	111.000000	0.045838	0.080233	200.000000	15.200000	

8 rows × 62 columns

```
data.shape
```

```
(997, 63)
```

Fig 4.4: Description of Dataset

4.4 RESULTS:

Results for the experiments mentioned in the previous chapters are from fig 4.5 to 4.11:

1. **AEEEM:**

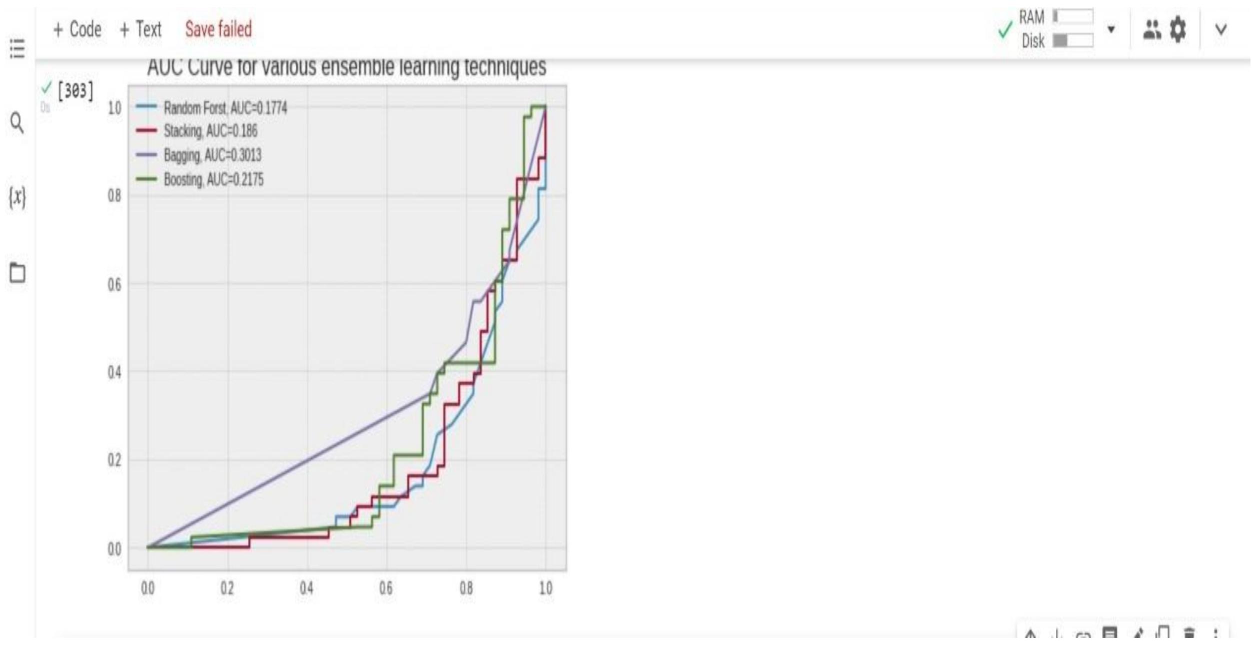


Fig 4.5: AOC of the accuracy of EQ dataset

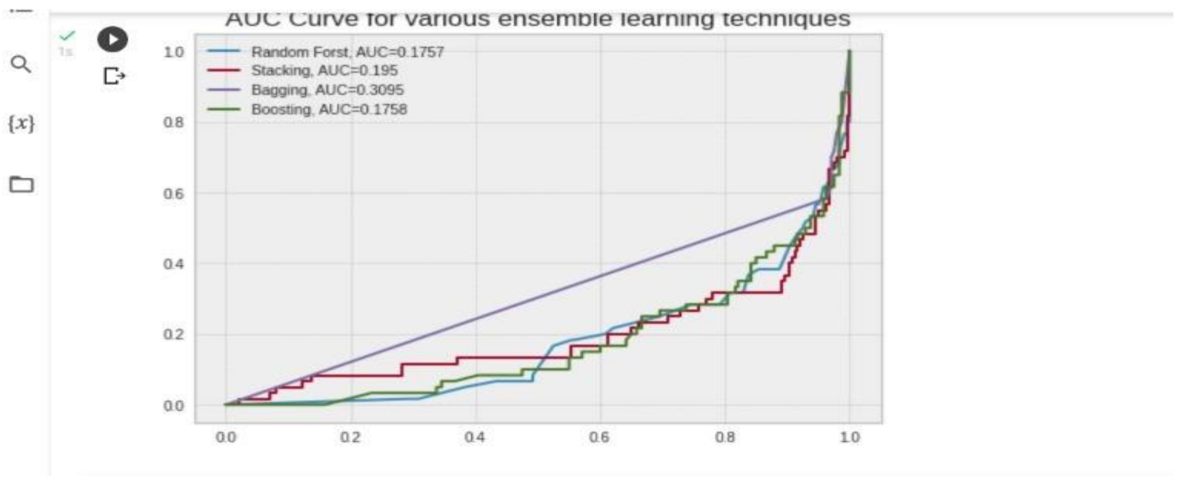
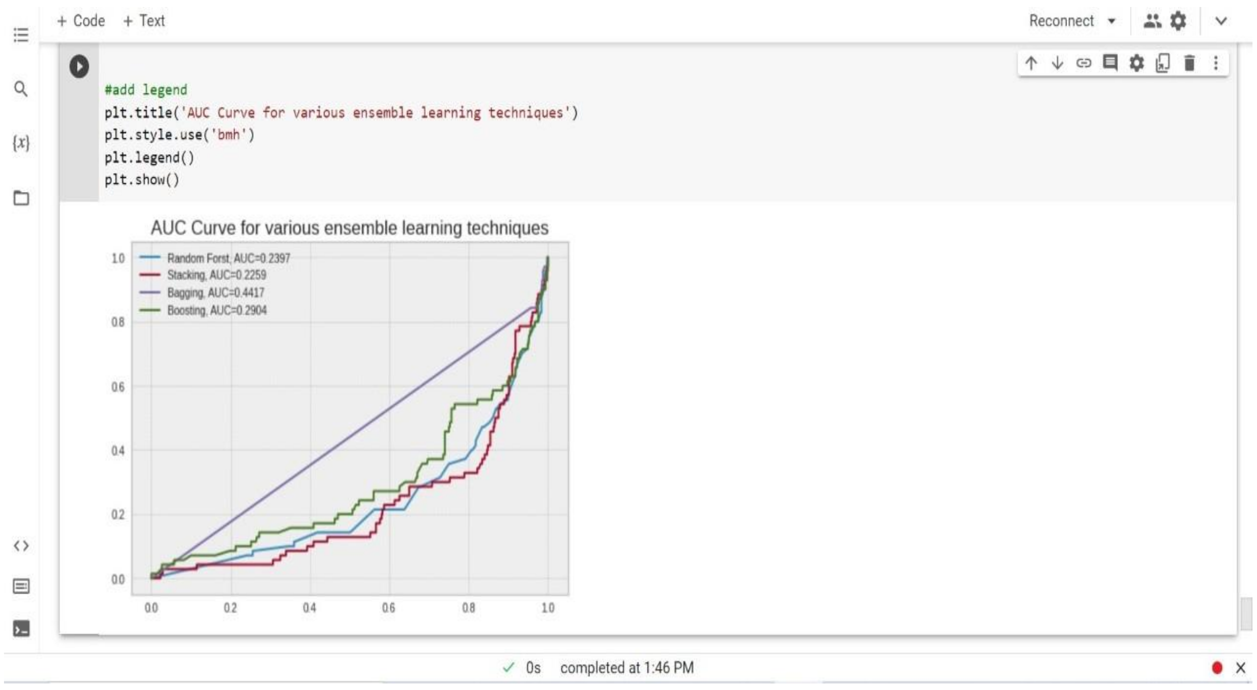
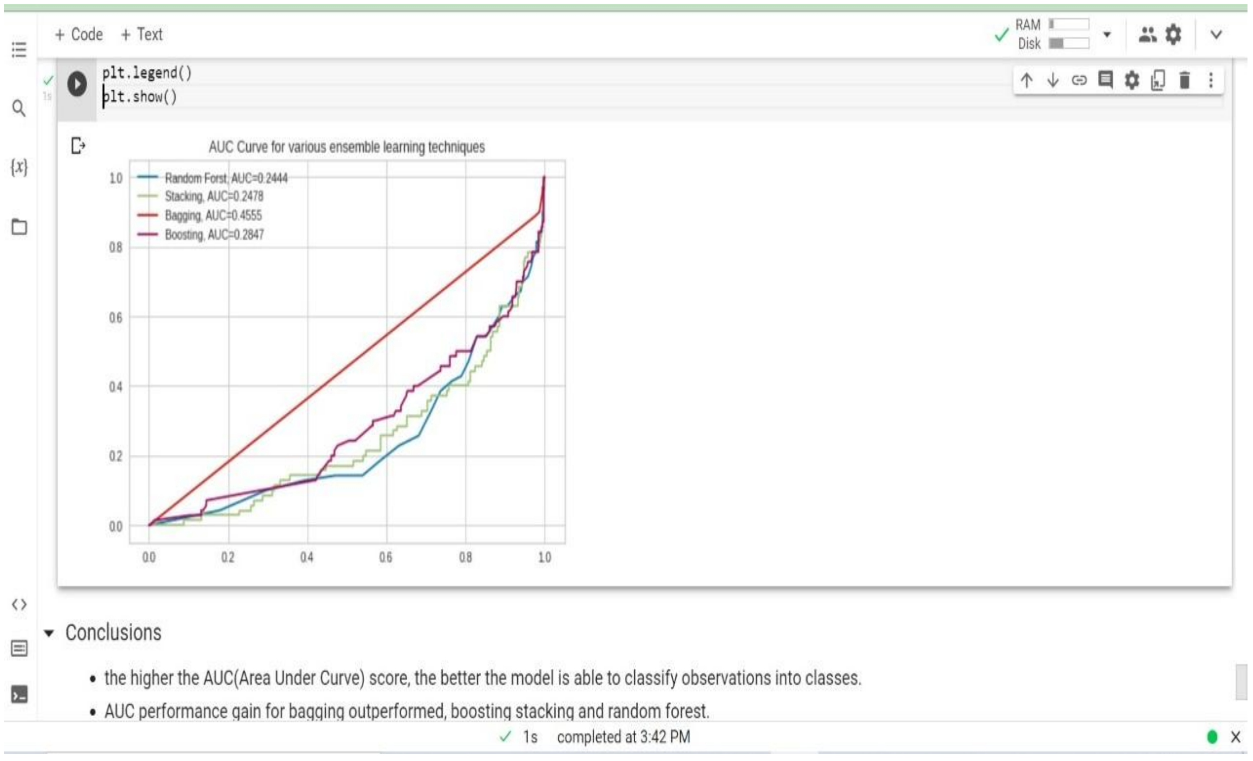


Fig 4.6: AOC of the accuracy of the JDT dataset



```
#add legend
plt.title('AUC Curve for various ensemble learning techniques')
plt.style.use('bmh')
plt.legend()
plt.show()
```

Fig 4.7: AOC of the accuracy of the MYLYN dataset



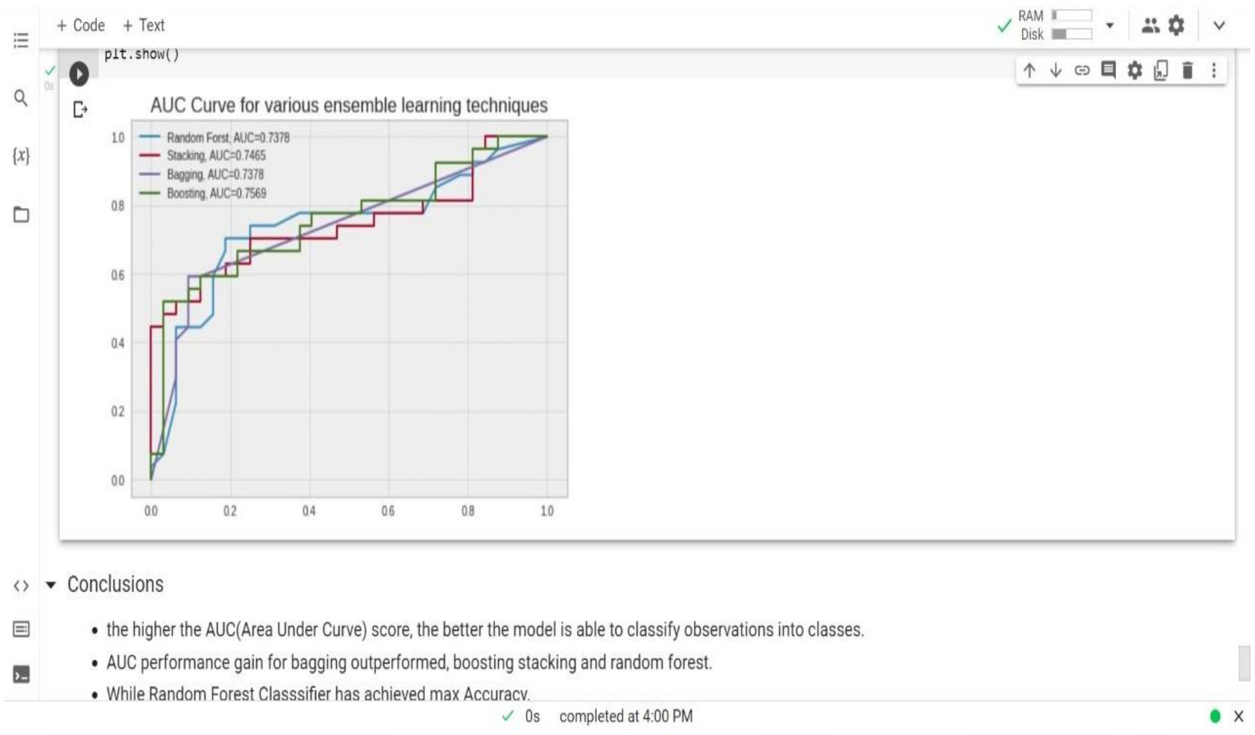
```
plt.legend()
plt.show()
```

Conclusions

- the higher the AUC(Area Under Curve) score, the better the model is able to classify observations into classes.
- AUC performance gain for bagging outperformed, boosting stacking and random forest.

Fig 4.8: AOC of the accuracy of the PDE dataset

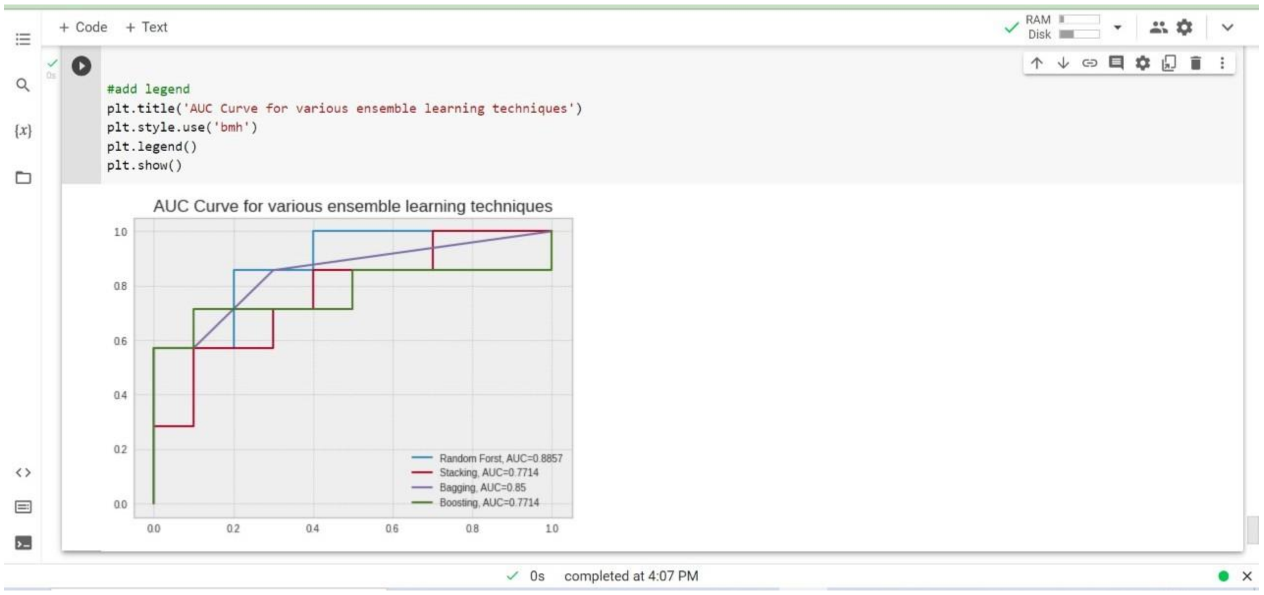
2. RELINK:



```
plt.show()
```

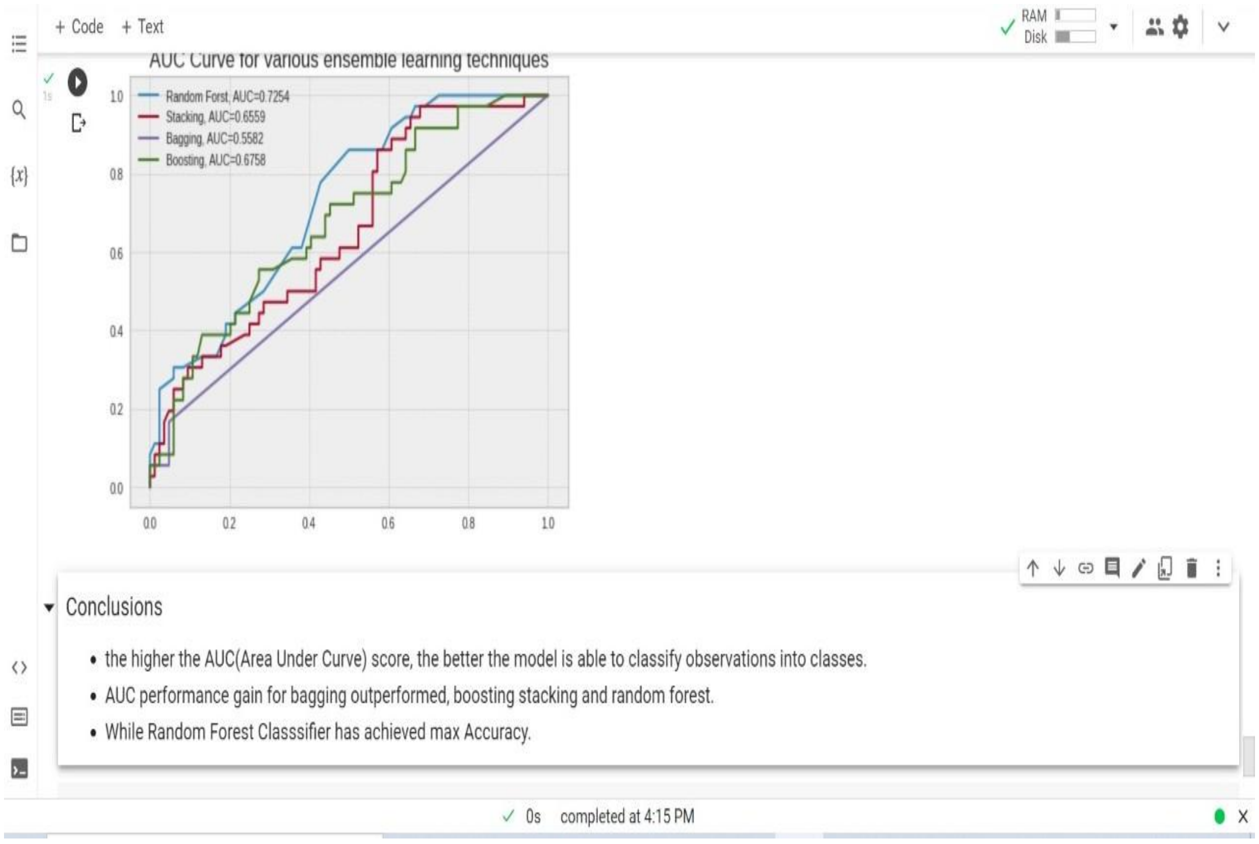
- the higher the AUC(Area Under Curve) score, the better the model is able to classify observations into classes.
- AUC performance gain for bagging outperformed, boosting stacking and random forest.
- While Random Forest Classsifier has achieved max Accuracy.

Fig 4.9: AOC Curve of APACHE dataset



```
#add legend
plt.title('AUC Curve for various ensemble learning techniques')
plt.style.use('bmh')
plt.legend()
plt.show()
```

Fig 4.10: AOC Curve of the SAFE dataset



Conclusions

- the higher the AUC(Area Under Curve) score, the better the model is able to classify observations into classes.
- AUC performance gain for bagging outperformed, boosting stacking and random forest.
- While Random Forest Classsifier has achieved max Accuracy.

✓ 0s completed at 4:15 PM

Fig 4.11: AOC Curve of ZXING dataset

CHAPTER 5

CONCLUSION, PERFORMANCE EVALUATION, AND FUTURE SCOPE

5.1 PERFORMANCE EVALUATION:

DATASET	RANDOM FOREST	BAGGING	BOOSTING	STACKING
MYLYN	67.11%	87.2%	86%	86.6%
PDE	85.5%	84.64%	84.6%	86.6%
APACHE	74%	69%	68%	92%
SAFE	70%	82%	82%	70.54%
ZXING	70%	70%	70.24%	71.84%
JDT	84.6%	83%	80%	84.3%
EQ	74.4%	66%	69%	90.3%
LUCENE	88.94%	88%	89%	90.3%

TABLE 5.1: Accuracy Percentage of all the datasets

5.2 CONCLUSIONS:

The higher the AUC(Area Under Curve) score, the better the model can classify observations. AUC Performance Gain for bagging outperformed, boosting stacking and random forest.While Random Forest Classifier has achieved max Accuracy.

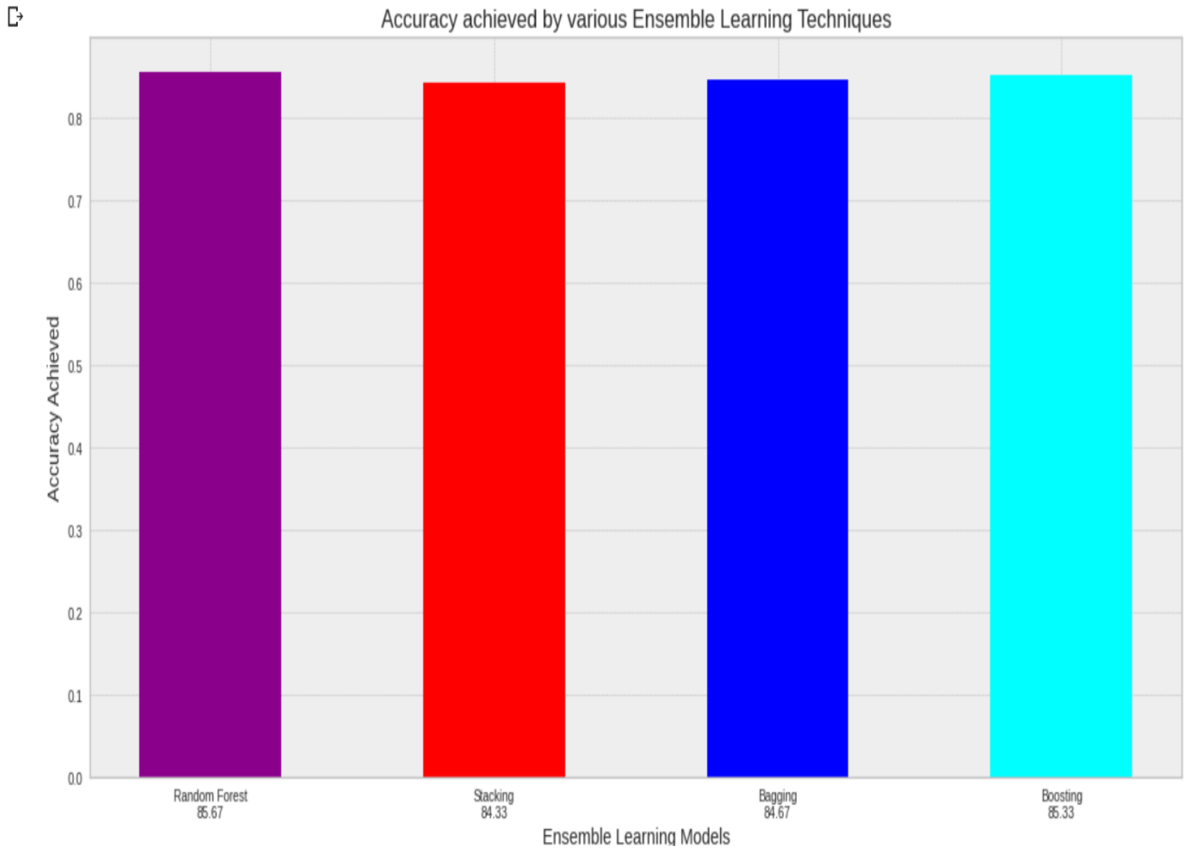


Fig 5.1: Graphical Plot of Accuracy achieved by various Ensemble Learning Techniques

5.3 FUTURE SCOPE

- One exciting subject we want to look into is how effective a single classifier technique would be if it was given more time to explore its idea space than the ensemble method requires to train several classifiers.
- Bagging and Boosting will be compared to other recently introduced approaches.
- To avoid the effect of having to weight classifiers, we plan to study the usage of Stacking and Super Learner as a technique of training a combining function.
- The goal will be to design a learner that can be turned on and off with a one-button click. We will be keeping the advantages of Boosting but also avoid overfitting on noisy datasets.

REFERENCES :

[1] A.O. Balogun, F. B. Lafenwa-Balogun, H. A. Mojeed, V. E. Adeyemo, O. N. Akande, A. G. Akintola, A.O. Bajeh, and F. E. Usman-Hamza, ‘‘SMOTE-based homogeneous ensemble methods for software defect prediction,’’ in Computational Science and Its Applications—ICCSA 2020 (Lecture Notes in Computer Science), vol. 12254, O. Gervasi et al., Eds.Cham, Switzerland: Springer, 2020, DOI: 10.1007/978-3-030-58817-5_45.

[2] H.Aljamaan and A.Alazba, ‘‘Software defect prediction using tree-based ensembles,’’ in Proc. 16th ACM Int. Conf. Predictive Models Data Anal. Softw. Eng. (PROMISE). New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 1–10, DOI: 10.1145/3416508.3417114

.

[3] Somya Goyal, Pradeep Kumar Bhatia, “Empirical Software Measurements with Machine Learning” Computational Intelligence Techniques and Their Applications to Software Engineering Problems. Boca Raton: CRC Press, pp. 49-64. 2021.

[4] H. Tong, B. Liu, and S. Wang, “Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning”, Inf. Softw. Technol., vol. 96, pp. 94-111, Apr. 2018

[5] A. Kaur and K. Kaur, ‘‘Performance analysis of ensemble learning for predicting defects in open source software,’’ in Proc. Int. Conf. Adv. Comput., Commun., Informat. (ICAC), 2014, pp. 219–225, DOI: 10.1109/icacci.2014.6968438.

[6] Li, M., Zhang, H., Wu, R., & Zhou, Z. H. (2012). Sample-based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 19(2), 201-230

[7]I. H. Laradji, M. Alshayeb, and L. Ghouti, ‘‘Software defect prediction using ensemble learning on selected features,’’ Inf. Technol., vol. 58, pp. 388–402, Feb. 2015.

[8] S. S. Rathore and S. Kumar, ‘‘Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems,’’ Knowl.-Based Syst., vol. 119, pp. 232–256, Mar. 2017.

[9] FaseehaMatloob, Shabib Aftab, Ahmed Iqbal, " A Framework for Software Defect Prediction Using Feature Selection and Ensemble Learning Techniques", International Journal of Modern Education and Computer Science(IJMECS).

[10] Z. Sun, Q. Song, and X. Zhu, ‘‘Using coding-based ensemble learn- ing to improve software defect prediction,’’ IEEE Trans. Syst., Man, Cybern. C, Appl. Rev., vol. 42, no. 6, pp. 1806–1817, Nov. 2012, DOI:
10.1109/tsmcc.2012.2226152.

[11] I. Alazzam, I. Alsmadi, and M. Akour, ‘‘Software fault proneness prediction: A comparative study between bagging, boosting, and stacking ensemble and base learner methods,’’ Int. J. Data Anal. Techn. Strategies, vol. 9, no. 1, p. 1, 2017, DOI: 10.1504/ijdats.2017.10003991.

[12] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, ‘‘Software defect prediction using feature selection and random forest algorithm,’’ in Proc. Int. Conf. New Trends Comput. Sci. (ICTCS), Oct. 2017, pp. 252–257, DOI:
10.1109/ictcs.2017.39.

[13] S. Young, T. Abdou, and A. Bener, ‘‘A replication study: Just-in-time defect prediction with ensemble learning,’’ in Proc. IEEE/ACM 6th Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng. (RAISE), May/Jun. 2018, pp. 42–47.

[14] R. Li, L. Zhou, S. Zhang, H. Liu, X. Huang, and Z. Sun, ‘‘Software defect prediction based on ensemble learning,’’ in Proc. 2nd Int. Conf. Data Sci. Inf. Technol. (DSIT), Jul. 2019, pp. 1–6, DOI: 10.1145/3352411.3352412.

[15] S. S. Rathore and S. Kumar, ‘‘An empirical study of ensemble techniques for software fault prediction,'' Int. J. Speech Technol., vol. 51, no. 6, pp. 3615–3644, Jun. 2021, DOI: 10.1007/s10489-020-01935-6.

[16] S. Mehta and K. S. Patnaik, ''Improved prediction of software defects using ensemble machine learning techniques,'' Neural Comput. Appl., pp. 1–12, Mar. 2021, DOI: 10.1007/s00521-021-05811-3.

[17] I. Alazzam, I. Alsmadi, and M. Akour, ''Software fault proneness prediction: A comparative study between bagging, boosting, and stacking ensemble and base learner methods,'' Int. J. Data Anal. Techn. Strategies, vol. 9, no. 1, p. 1, 2017, DOI: 10.1504/ijdats.2017.10003991.

[18] Z. Sun, Q. Song, and X. Zhu, ''Using coding-based ensemble learning to improve software defect prediction,'' IEEE Trans. Syst., Man, Cybern. C, Appl. Rev., vol. 42, no. 6, pp. 1806–1817, Nov. 2012, DOI: 10.1109/tsmcc.2012.2226152.

[19] Ensemble Methods | Bagging Vs Boosting Difference https://dataaspirant.com/ensemble-methods-bagging-vs-boosting-difference/

[20] I. H. Laradji, M. Alshayeb, and L. Ghouti, ''Software defect prediction using ensemble learning on selected features,'' Inf. Technol., vol. 58, pp. 388–402, Feb. 2015, DOI: 10.1016/j.infsof.2014.07.005.