Use of Ensemble Learners to Predict Number of Defects in a Software

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE OF

MASTER OF TECHNOLOGY IN SOFTWARE ENGINEERING

Submitted By:

Mayank Yadav

2K21/SWE/13

Under the supervision of

Prof. RUCHIKA MALHOTRA

(HOD, SE, DTU)

DEPARTMENT OF SOFTWARE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering) Bawana Road, Delhi-110042

April, 2023

Department of Software Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road Delhi-110042

CANDIDATE'S DECLARATION

I Mayank Yadav, 2K21/SWE/13 of Master of Technology (Software Engineering) hereby declare that the Major Project-II Dissertation titled **"Use of Ensemble Learners to Predict Number of Defects in a Software"** which is submitted by me to the Department of Software Engineering, Delhi Technological University, Delhi in partial fulfillment of requirement for the award of degree of Master of Technology (Software Engineering) is originaland not copied from any source without proper citation. This work has not been previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi **Mayank Yadav**

Date: **2K21/SWE/13**

Department of Software Engineering

Delhi Technological University

(Formerly Delhi College of Engineering)

Bawana Road Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled “**Use of Ensemble Learners to Predict Number of Defects in a Software**” submitted by Mayank Yadav (2K21/SWE/13) to the Department of Software Engineering, Delhi Technological University in partial fulfillment of requirement for the award of the degree of Master of Technology, is a record of project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date:

Prof. RUCHIKA MALHOTRA

(Supervisor)

Head of Department

Software Engineering

Delhi Technological University

ACKNOWLEDGEMENT

First, I would like to thank the Almighty, who has always guided me to follow the right pathof the life. My greatest thanks are to my parents who bestowed the ability and strength in me to complete this work.

My thanks are addressed to my mentor Prof. Ruchika Malhotra, Department of Software Engineering who gave me this opportunity to work in a project under her supervision. It was her enigmatic supervision, unwavering support and expert guidance which has allowed me to complete this work in due time. I humbly take this opportunity to express my deepest gratitude to her.

Date:

Mayank Yadav

M.Tech (SWE)-4th Sem

2K21/SWE/13

ABSTRACT

Presently Fault detection is crucial in industry. Early discovery of faults may aid in the prevention of subsequent abnormal events. Fault detection can be achieved in a variety of ways. This research will go through the fundamental approaches. At this moment, methods for finding flaws faster than the customary time restriction are necessary. Detection methods include data and signal approaches, process model-based methods, and knowledge-based methods. Some treatments need very precise models. Early issue discovery increases life expectancy, enhances safety, and lowers maintenance costs. When choosing a fault detection system, several factors must be considered. Principal Component Analysis can help find flaws in large-scale systems. Signal models are used when difficulties arise as a result of process changes. This research includes a systematic review from the literature, along with a selection of noteworthy applications. In this research, we would want to go through different real-world scenarios that employ different defect detection methodologies. In other words, we will look at both hardware and software concerns. The first case considers fault detection, and a decision tree technique is utilized to detect these defective lines. The algorithm is designed to categorize as defective or non-faulty whenever possible. In second scenario, to discover faults in each dataset, we shall employ the "ensemble learning" learning technique. We will be working on the datasets. During testing activity, software shows occurrences of multiple defects. And, that too capable of causing instant failures; thereby decreasing the software's capability.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AUC	Area Under Curve
BNET	Bayes Network
C & K	Chidamber and Kemrer
DS	Decision Stump
DT	Decision Tree
DTABLE	Decision Table
DV	Dependent Variable
EL	Ensemble Learners
ER	Ensemble Regressor
ES	Ensemble Selection
FSS	Feature Subset Selection
GLM	Generalized Linear Model
KNN	K-nearest Neighbor
LR	Logistic Regression
ML	Machine Learning
NB	Naïve bayes
OO	Object Oriented
RF	Random Forest
SVM	Support Vector Machine

CHAPTER 1

INTRODUCTION

Prediction models are useful for software testers since they aid in quantitative planning. Furthermore, the availability of free software defect information archives has opened new areas for study, implementation, and evaluation of machine learning algorithms for software defect prediction models based on software characteristics' measurements. This part also discusses the topic of defect prediction and how it is fed nourishment by the ensemble learning to use predictive displaying of modelled information.

1.1 Software

The Software is a system of physical devices combined with hardware that enables these items to associate and trade data. Software is typically classified into three types: System software serves as the foundation for application software. System software includes device drivers, operating systems (Oss), compilers, plate formatters, word processors, and utilities that help the PC run more efficiently. It is also responsible for supervising equipment parts and providing critical non-task-specific capabilities. Typically, system software is developed in a programming language. Programming software is a collection of tools that developers may use to create hard programs. Compilers, linkers, debuggers, translators, and word processors are among the several programming software available.

Application software is designed to carry out certain tasks. Office suites, gaming applications, database frameworks, and instructional software are examples of use software.

Application software can be a single application or a collection of small projects. This is the type of software that most people think of when they hear the term “software.”

1.2 Software Defect Prediction

The study of forecasting which software “modules” are faulty is known as defect prediction. In this context, “modules” refers to a primitive unit of a running system, such as a function or a class. Hundreds to thousands of classes can be found in a typical object-oriented software project. It is usual practice to use some quality assurance (QA) methodologies to analyse the classes’ intrinsic quality in order to ensure general and project-related fitness qualities for such programs. Inspections, unit tests, static source code analyzers, and other approaches are examples of these techniques. A defect log is a record of the results of this QA.

We can use these logs to train defect predictors if the data contains not only a precise account of the encountered faults (i.e., “bugs”), but also a detailed description of static code features such as lines of code (LOC), complexity measures (e.g., McCabe’s cyclomatic complexity), and other suitable object-oriented design metrics. There are three compelling reasons to investigate defect predictors learned from static code attributes. They are simple to utilize, widely used, and useful.

Software defect (or deficiency) prediction is regarded as one of the most practical and helpful technologies for determining if a certain module is faulty or not. Software experts regard it as a critical stage for ensuring the quality of the technique or product that is to be generated.

It made light of an extremely critical role in attaining the instances, the software industry that it can't satisfy the demands within budget and on time.

1.3 Ensemble Of Machine Learning

Ensemble techniques appear to be meta-calculations that combine a few machine learning algorithms into a single predictive model to enhance predictions (voting), decrease predilection (boosting), or decrease difference (sacking). Ensemble learning is a strategy that includes specific regressors in which an indicator is built utilizing a pack of regressors that may be of the same kind and at that point new information focuses are ordered by taking a weighted vote or the arrived at the midpoint of aftereffect of their yields. Base learners are a group that work together to create an ensemble model.

1.4 Predictive Modelling

To create models in defect prediction, Machine Learning (ML) approaches are used. Information, often known as recorded data, is extracted from the past and used to forecast future results. Predictive modelling is a method in which models are created to evaluate outcomes. Each model includes both free (indicators) and ward factors (result). The main goal of predictive modeling is to discover links between dependent and independent variables that affect changes in other variables as a result of one variable.

Software testing is a critical stage in the software development life cycle. Software faults produce a mismatch between real and intended output, resulting in system failure. A software defect is a problem in a software product that fails to fulfil the software need (as expressed in the requirement specifications) or end-user expectation (which may not be specified but is reasonable).

Software defect prediction (SDP) is a method for predicting a software system's fault-prone module. After identifying the faulty module, the project manager might assign additional testing team members to other faulty software modules. It lowers the overall cost of software development.

1.5 Organization Of the Thesis

In this theory, we intend to identify the optimal strategies for dealing with the SDP issue. New tactics are studied and compared to traditional indicators. The flow section contains the examination's broad outline.

The writing overview underlying the assessment is covered in Chapter 2.

Furthermore, Chapter 3 discusses the flaws of ensemble learning as well as the basic models used in the evaluation.

Chapter 4 includes a detailed representation of the OO measures that were used in the dataset.

Chapter 5 introduces the group of learners who were used for activities such as stacking, boosting, and so on.

Chapter 6 summarizes the findings and visually depicts the conclusions obtained from the research.

CHAPTER 2

LITERATURE REVIEW

The writing studies that have been reviewed present generally persuasive models for the prediction of defects. The underlying work in software defect prediction focuses mostly on the use of quantifiable processes. The next section discusses the evaluations that were used in this research. Numerous software thinkers have investigated fault prediction in software. In any event, in this section, we will look at studies that use ML algorithms to predict defects based on OO measures.

Chidamber and Kemerer (CK) [2] metrics are used to determine if a piece of code adheres to OO standards. Gyimothy et al. [1] used Decision Tree (DT) and AI techniques such as computed relapse and neural system to uncover the association between CK values and prediction of defects. The research by Singh et al. [10] supported using these estimates to relinquish inclined software elements. The focus also proposed doing a large number of research to determine the predictive limit of ML computations in this area. There was a link formed between the factual model and the ML procedures in that evaluation, and it was assumed that ML strategies perform better than traditional factual calculations in the domain of predictive demonstrating.

Another ongoing focus by Malhotra [3] evaluated the Android ensemble computations capacity of 18 ML. The results reveal that some computations, such as Logiboost and Nave Bayes, are more popular than others. In addition, MLP exhibited usefulness in the prediction of defects space.

Catal et al. [5] studied the fake invulnerable acknowledgement framework to authorize NASA KC1 information collecting. Malhotra et al. [4] evaluated the fitness of 17 AI algorithms for predicting defects based on the arrivals of the Xerces informative index. The considerations used ML methods to examine the relationships between defect prediction and OO measures. Malhotra [25] investigated numerous AI approaches for predicting defects. The concept of informative indexes for predicting abnormality is skewed. By skewed, we mean that information is unequally distributed; for example, non-flawed modules outnumber faulty ones. Non defective modules are negative models (or negative class or dominating part class) in machine learning writing, whereas faulty modules in information preparation are specific models (or positive class or on the other hand minority class).

This is referred to as the issue of class disparities. Class awkwardness degrades the presentation of AI algorithms greatly. Seiffert et al. [24] presented prerequisites as one solution to this problem.
Zhou and Leung [6] assessed the usefulness of CK measures for forecasting defects in terms of defect severity. They approved two severity levels on NASA's KC1 informational dataset using approaches such as Random Forest, Naive Bayes, and LR. It has been determined that the measuring number of learners is by all accounts of little consequence in the prediction of faults.

The results showed that the CK measures have significant limitations in order to predict class with high severity errors. Similarly, the models created using ML processes achieved minimal execution. Chug and Singh [10] examined five artificial intelligence (AI) computations used to predict early software flaws, including Artificial Neural Network (ANN), Linear Regressor (LC), Decision Tree (DT), Particle Swarm Optimization (PSO), and Nave Bayes (NB).

The results of this investigation demonstrate that, in terms of prediction exactness, the straight regressor outperforms other calculations, although ANN and DT calculations have the lowest mistake rate.

CHAPTER 3

Techniques and Object-Oriented Metrics

3.1 Ensemble Learning Motivation

There are several motivations pushing the deployment of collective AI approaches. Ensemble models have shown to be quite effective in inspiring model correctness and presentation. Some AI approaches do a local search rather than a global search, which commonly gets stuck in local optima.
For example, the computation for the decision tree develops the tree using a parting rule for greedy strategies.

However, an ensemble built from a few diverse starting points by performing a close search would often yield a better estimate of the true unlabeled sample than any of the methods considered alone. A learning calculation may be thought of as searching for a space H of theory in order to identify the best hypothesis in space.

Nonetheless, the factual issue emerges that the amount of information available to create the model is insufficient in comparison to the extent of the speculation space. Without appropriate information, a broad variety of theories may be found in a learning calculation in H, which when combined with preparation knowledge delivers equivalent precision for the most part. By combining all of these precise models, the computation can have weighted-normal of their votes, lowering the likelihood of selecting the incorrect predictor for prediction.

3.2 Base Learners

Ensemble models are built employing the standard basic learners but brings about better precision furthermore, execution of the model. The models used in this study are depicted below, along with their pros and shortcomings.

3.2.1 Support Vector Machine

A Support Vector Machine (SVM) is interpreted provisionally as a biased regressor by a different hyperplane; nevertheless, we may understand SVM as the computation that arranges new models develops a perfect hyperplane provided the managed learning information (marked prepared information). This hyperplane is a line that divides a plane of information into two zones in the two-dimensional space where it resides on each side. It mostly uses the bit trick to arrange information that cannot be ordered directly.

The calculation's primary goal is to predict a plane that increases class separation in order to reduce the likelihood of overfitting and misclassification of the incoming information point.

In real life, a tree has a few comparisons and seeks to affect a broad variety of ML methods crossing both relapse and organization. A decision tree may also be used in predictive analysis to show decisions and judgment-making openly as well as clearly.

As the name indicates a tree-like structure is used for settling on choices where at each node we are anticipated to decide on options which in the end prompts the ideal.

3.2.2 Tree Regressors

Tree regressors come under supervised learning concept. The platform is provided through sklearn. It is a software that helps us learn machine learning concepts. It is a library to be more precise. We can apply regression through tree like structures provided under the names such as decision tree (DT), bagging ensemble, forest of randomised trees, adaboost, gradient tree boosting and histogram based gradient tree boosting. Let us look at these concepts in some detail to make things more sensible.

Decision trees can be utilised very well in classification problems. But research has shown much promise offered by them when it comes to problems based solely on regression. A one-dimensional regression tree (decision tree) can be drawn to fit a curve based on observations which might have some noisy instances as well. There might be sighting of overfitting when it comes to training data. A bagging based regressor is highly useful when it comes to a comparison between base estimator and a highly advanced learner.

Generally, base regressors are fitted onto data randomly chose out of the complete instance set. At last, we aggregate this information obtained from base learners results to make our much more efficient predictions.

3.2.3 Decision Tree (DT)

Decision tree is a supervised learning approach that uses basic decision rules learned from data attributes to predict continuous output values. It generates a tree-like model in which each node represents a feature, each branch represents a decision rule, and each leaf represents a result. Decision tree regression is capable of handling both numerical and categorical data, as well as multi-output situations. Overfitting, instability, and poor extrapolation are possible drawbacks.

3.2.4 Logistic Regression (LR)

A supervised learning approach that may be utilised for classification problems is the logistic regressor. It calculates the likelihood of an occurrence, such as voting or not voting, based on a collection of independent factors. The logistic function is used to translate the linear combination of independent variables to a probability value between 0 and 1. It can handle binary and categorical data, as well as multi-class categorisation. It may, however, suffer from overfitting, multicollinearity, and outliers.

On the other hand, Linear regression is a supervised learning approach for regression applications. It simulates the linear connection between one or more independent factors and a dependent variable1. It fits the data points with a straight line and predicts the value of the dependent variable based on the independent factors. It is capable of handling both continuous and categorical data, as well as performing multiple linear regression. It may, however, exhibit multicollinearity, heteroscedasticity, and nonlinearity.

3.2.5 Naïve Bayes (NB)

A supervised learning method that may be utilised for regression applications is the Naive Bayes. Given an example with various qualities, it represents the conditional probability of a numeric goal value. It employs the Bayes theorem and the naïve assumption of conditional independence between any two qualities given a goal value. It is capable of handling both numeric and nominal characteristics, as well as performing multiple regression. It may, however, be plagued by unrealistic independence assumptions, data sparsity, and zero-frequency issues.

3.2.6 Extra Trees Regressor (ET)

The extra trees regressor is a supervised learning technique for regression applications. It simulates the average prediction of a group of randomised decision trees (also known as extra-trees) on different sub-samples of the dataset. Its construction differs from that of traditional decision trees. When determining the optimal split to divide a node's samples into two groups, random splits are drawn for each of the max features randomly selected features, and the best split among them is picked. The extra trees regressor can handle both continuous and categorical data, as well as multiple regression. It may, however, suffer from excessive volatility, complexity, and poor interpretability.

3.3 Metrics

As the use of measures has grown in popularity, so has the appearance of the specific set of dimensions. The goal of these measures is to produce excellent data that can be used to evaluate the perplexing frameworks. Coupling measurements are one of the models. Coupling is the use of techniques or attributes described by another class within a class. If a class collaborates with other classes, a subsystem or framework might be used to highlight the plan's multiple character. These are sometimes referred to as coupling metrics.

Weighted Methods per Class (WMC) statistic measures the approaches used in a class and is calculated by taking into account the cyclomatic complexity of all listed methods used in the class. A class with new methods will be adequate for the program area, restricting class reusability and covering the support, reusability, and under standardization aspects of the quality model.

Lines of CODE (LOC) is the number of dynamic code physical lines (executable lines) in one of the strategy's codes. Size may be calculated in a variety of ways. This includes inspecting all physical code lines, articulations, and so on.

Data Access Metric (DAM) represents the ratio of private (secured) collateral to the total number of reported features in the class. DAM requires a high value. It has a scale of 0 to 1.

Depth of Inheritance Tree (DIT), as the name suggests, it simply tells us how many levels of inheritance are there inside a structure of class.

Number of Children (NOC), as the name suggests, it simply tells us how many children class are there of any parent class.

Coupling Between Object Classes (CBO), as the name suggests, it simply tells us many classes a particular class is coupled with.

Response For a Class (RFC), as the name suggests, it simply tells us how many methods can be made to execute when a message is to be responded to.

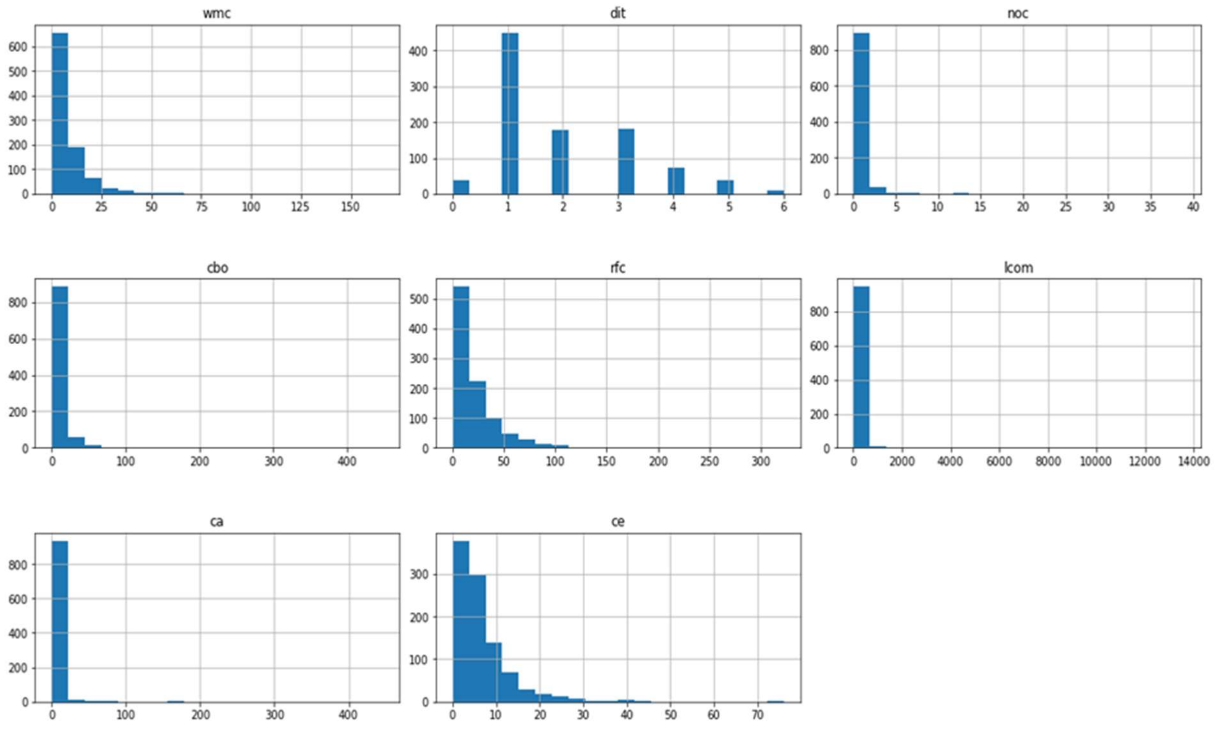

Figure 3.1 Dataset visualization is done to get visual representation of the various metrics present in the dataset description.

CHAPTER 4

ENSEMBLE OF REGRESSOR MODELS

A regressor ensemble seemed to outperform a single regressor model. As a general rule, the single regressor framework encounters overfitting and bias in the regressor. Ensemble regressors have demonstrated a typically outstanding performance in defeating such circumstances.

We evaluated the display of various arrangements of regressor systems on the Ant dataset alongside its diverse forms in this study.

4.1 Bagging

Bagging is a method of condensing Bootstrap Aggregating. Breiman demonstrated bagging. The idea behind sacking is simple: the ensemble is made up of regressors that rely on the bootstrap copies of the preparation set. The individual regressor's outputs are combined using the blend rule of bigger part voting. In this case, bootstrap testing is used, and a subset of information focuses are picked at random from a space of information focuses with names.

The key hidden rule is that the instances are obtained with substitution, thus we can state that an information point that has been obtained previously has the same possibility of being obtained again as other information foci that have not been obtained previously.

Algorithm for Bagging:

1.Boundaries are instated

- D = φ, where D is the ensemble.
- L, the quantity of regressor to prepare.
- Parameters are instated

2. For k = 1 to L

- Take a bootstrapped test Sk from Z.
- Build a regressor Dk by utilizing Sk as the preparation set.
- Add regressor to the current ensemble.

3. Bring D back.

4. For Testing Phase, Run D1 upto DL on the info X.

5. The class having lion's share of votes is marked as the regression of that example.

4.2 Boosting

The basic idea underlying the ensemble model's operation is to increase the predictive intensity of the model by repeatedly incorporating each regressor. When a regressor enters an ensemble at a certain stage, it is prepared on an informative collection that is randomly evaluated from the preparation informational index. Test appropriation begins with a consistently steady mode and progresses to the prediction of difficult information focuses.

Boosting brings together powerless students. On the other hand, we may argue that basic students are created by combining AI computations with a different appropriation to create a solid regressor with solid regulations. Each time a fundamental learning calculation is used, a new standard is produced.

At the end of the day, boosting is an iterative strategy in which the primary calculation is prepared for the entire dataset and the resulting calculations are built by fitting the residuals of the primary calculation, thereby providing more significant load to the perceptions that the previous model failed to predict. AdaBoost is a variant of the boosting ensemble inclining strategy. AdaBoost stands for Adaptive Boosting.

Finally, we shall highlight previous progress until the restriction of the ensemble learning calculation is met or higher accuracy is achieved. To begin, we assign equal loads to all of the information foci. If there is any off-base forecast, i.e., the blunder of prediction because of the primary calculation of essential characterization, we give more weight to those perceptions with mistake of prediction. The following computation is used to discover the base at that location.

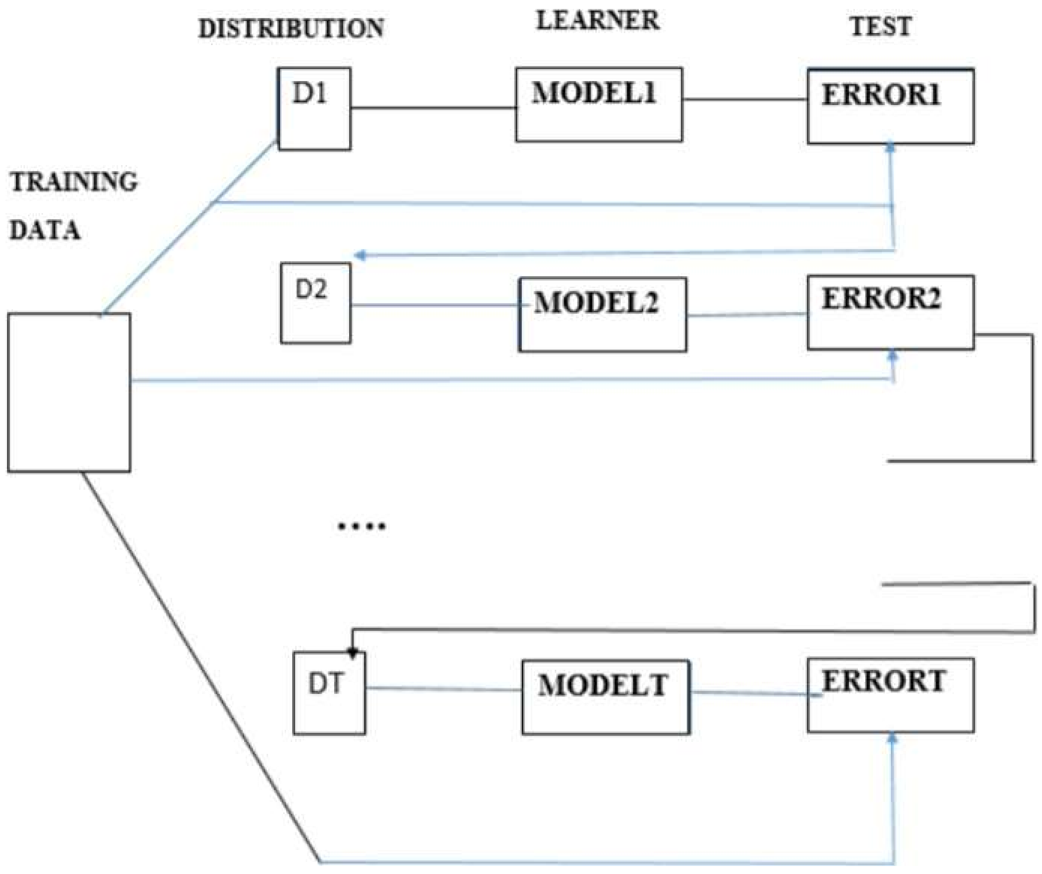

Fig 4.1: Learning with Boosting [7]

Figure 4.1 describes how the boosting process works. First training data is chosen. Out of this training data various distributions are obtained. These distributions are further utilized in different learner models. These different models will produce various errors. These errors will like multiple sets of test data.

In this we have picked 4 diverse base regressors for the boosting strategy. Following are the regressors:

1. Supported Decision Tree
2. Supported SVM
3. Supported Logistic Regression
4. Supported Naïve Bayes

Algorithm for Boosting:

1. Info:

- A marked dataset with N information focuses (if class name sets).
- A learner model (NN, DT, SVM).

2. Learning stage (Training of the Model)

- D T base models are developed on T exceptional inspecting appropriations based on the preparation dataset.
- An example dissemination Dt is worked for model t by adjusting the testing conveyance Dt-1 obtained from the t-1 th phase. Information foci that were incorrectly identified in previous endeavors are likely to have bigger burdens in new shaped data.

3. Characterization step

- The mark respect is gained based on a weighted majority of the class.

4.3 Voting

Voting is a well-known way of ensemble decision-making. Casting a vote connects the option from various models in light of a blend preclude that goes to be an alternate mix of probability evaluations. Models can be of numerous types, such as choices from a single model regressor, a homogeneous model regressor ensemble, or even a heterogeneous model of ensemble.

The approach used in polling a vote technique is simple and similar to the ensemble strategy of dominating portion voting a vote used in other ensembles such as stacking or AdaBoost.

The main distinction is that in Bagging or AdaBoost, choosing a vote plot acts as a mix rule for the last relationship, whereas in voting a vote ensemble technique, voting a vote alludes to a class or participant who collects names from various sources and uses likelihood measurement for official outcome making. In this study, we chose three unique base regressors for the voting technique.

Voting ensemble regression is a technique that combines the predictions from multiple regression models and averages them to get a final prediction1. It is a type of ensemble learning method that can improve the accuracy and robustness of regression models2. Diverse predictors utilize the various predictions obtained from a single new instance. These diverse predictors help generate multiple predictions and thus, a prediction called ensemble prediction is obtained.

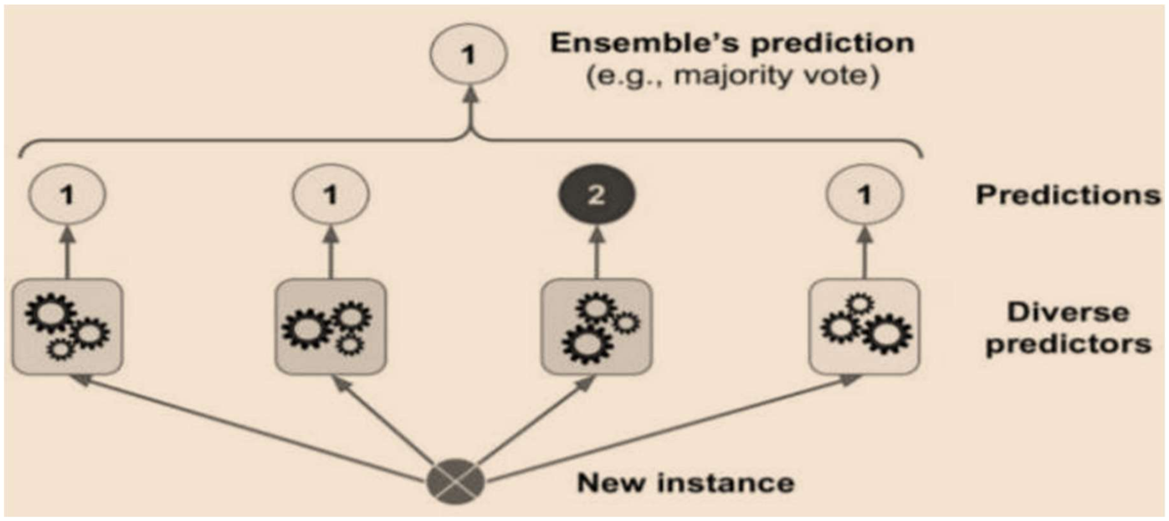

Fig 4.2: Learning with Voting
[7]

4.4 Stacking

Stacking is a machine learning (ML) process and an alternate model in ensemble considering as it considerably attempts to improve the ensemble's exactness and therefore the presentation by working on the errors. It addresses the issue of regressor inclination with regard to information used for preparation and center to learn and use these decided inclinations to enlarge the grouping, which is known as stacking generalization.

Wolpert suggested Stacked Generalization (or stacking) in 1992, claiming that it "It is a method of combining many models to provide a meta-learner concept. Despite the fact that it is an appealing notion, it is used in writing less frequently than bagging and boosting ".

In AI, ensemble techniques employ n(n>1) models to achieve more order accuracy than any of the constituent models could achieve. We have taken help of scatter plots to effectively represent the comparison between actual and predicted bug count. Furthermore, higher severity level defects may be anticipated so that asset allocation can be managed properly. Figure 4.3 shows how the underlying step generates a collection of base level-1 regressors. In the following stage, a meta-level regressor called the meta-learner is used to combine the predictions of the base level1 regressor.

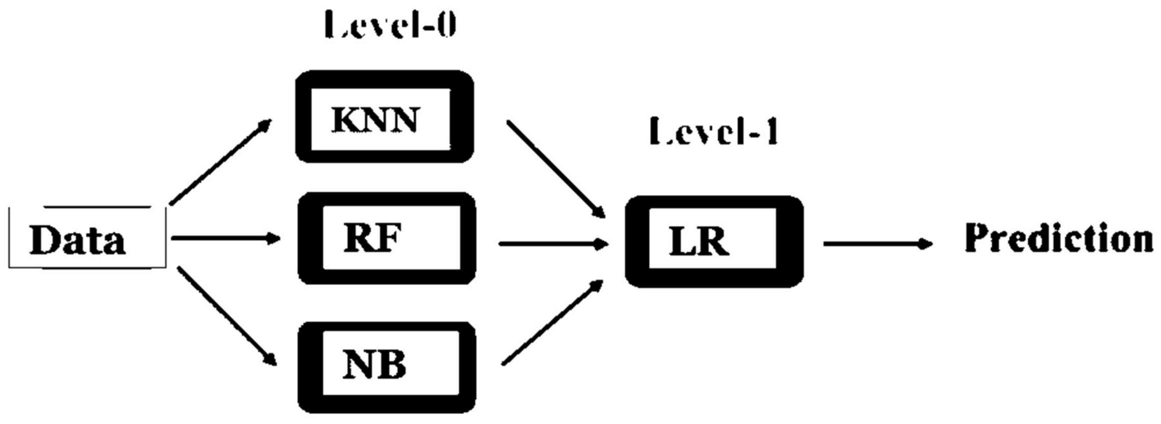

Fig 4.3: Learning with Stacking [7]

CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1 Data Description

In this study, we used multiple Ant dataset versions to run the analysis on the PROMISE vault with varying sizes of various modules. The table depicts the dataset representation. This dataset is often used for bug prediction testing using software object organized measurements. Because the information contains the inclusion of bugs existing in the section of defects, it should be preprocessed for the simultaneous order used in the study. The data set 'Ant' has a double section, viz bug, which indicates whether or not a class has an error. As a result, the section has been renamed defects. If the estimation of the twofold section is zero, 'her' are no defects. Furthermore, if the double section value is 1 or higher, the class will have defects. Table 5.1 summarizes the information, as well as its variations and the damaged and non-defective modules. The information that is shown by the software release section has five various versions.

Table 5.1 Dataset Description

SOFTWARE RELEASE	TOTAL INSTANCES	DEFECTIVE INSTANCES	NON-DEFECTIVE INSTANCES
ANT 1.3	125	20	105
ANT 1.4	178	40	138
ANT 1.5	293	32	261
ANT 1.6	351	92	259
ANT 1.7	745	166	579

5.2 Results

Accuracy is the primary execution measure used. It measures the number of correct instances expected over the total number of tests. For example, if the regressor is correct 90% of the time, it means that it correctly predicts the count 90% of the time out of 100 cases.

We have chosen the ant dataset for our experimentation work for this research. This dataset belongs to the PROMISE repository of Eclipse based Java systems. These are public datasets which have been commonly utilised by practitioners in this area of research work.

Table 5.2 displays dataset description (in percentage).

DATASET	**DEFECTIVE MODULES (%)**
Ant 1.3	14
Ant 1.4	74
Ant 1.5	55
Ant 1.6	20
Ant 1.7	15

The principle is to get insights needed for studying defective software. By analysis of the dataset chosen we can easily remove the instances that we won't need. So, that's what has been performed first of all.

We selected some of the available versions of the ANT dataset. This was done to provide diversifying results when it comes to the software defect prediction process. It has been made sure that the first phase would focus solely on cleaning of the data chosen.

The two chosen tree regressor models- Decision Tree (DT) Regressor and Extra Tree (ET) Regressor; are observed on the basis of their R2 Scores in the **Table 5.3** below:

Table 5.3 R2 Scores

S. No.	**Dataset Release**	**Decision Tree Regressor**	**Extra Tree Regressor**
1	Ant 1.3	0.99	0.93
2	Ant 1.4	0.63	0.69
3	Ant 1.5	0.61	0.67
4	Ant 1.6	0.91	0.79
5	Ant 1.7	0.96	0.73

The two tree regressors are also observed on the basis of the mean squared error. This is done due to the reason that we had to find that what difference exists between actual and predicted outcomes, if we square them.

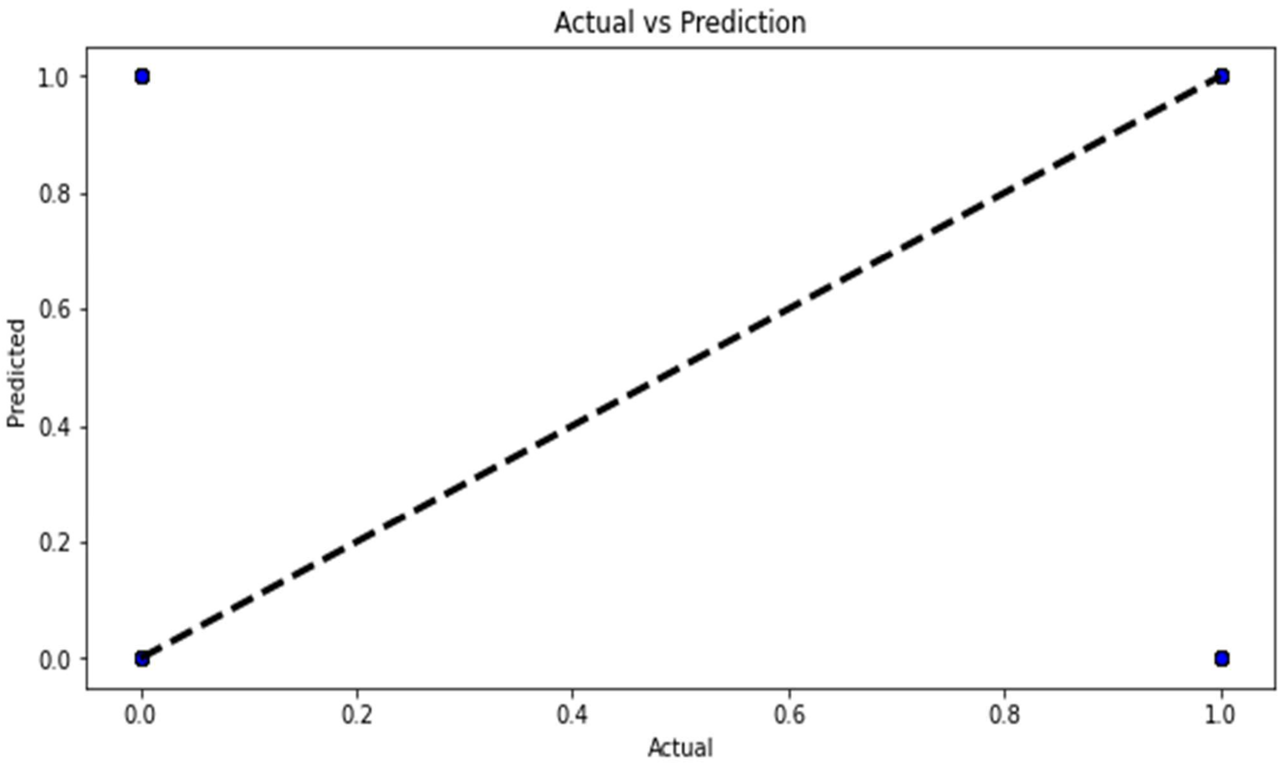

Fig. 5.1 Graph between actual and predicted counts.

Figure 5.2 shows that how the graph plotted between actual and predicted defect counts in a particular dataset can vary. As per this graph, results are not shown yet because both counts aren't marked.

Mean Squared Error (MSE) values have been calculated and mentioned as in a tabular manner via **Table 5.4**.

Table 5.4 Mean Squared Error (MSE).

S. No.	**Dataset Release**	**Decision Tree Regressor**	**Extra Tree Regressor**
1	Ant 1.3	0.077	1.083
2	Ant 1.4	0.408	1.406
3	Ant 1.5	0.975	1.262
4	Ant 1.6	0.537	1.083
5	Ant 1.7	0.949	1.244

A good way to compare models has already been shown above by the help of performance metrics. Corresponding tables haven been drawn in order to represent observations made. But, the task in hand was to make actual predictions and then make comparisons with the actual outcomes or value of the respective bug count.

The reason being that scatter plots are useful when it comes to representing task if large amount of data. We can also easily correlate between the actual and predicted value of the bugs with the help of scatter plots.

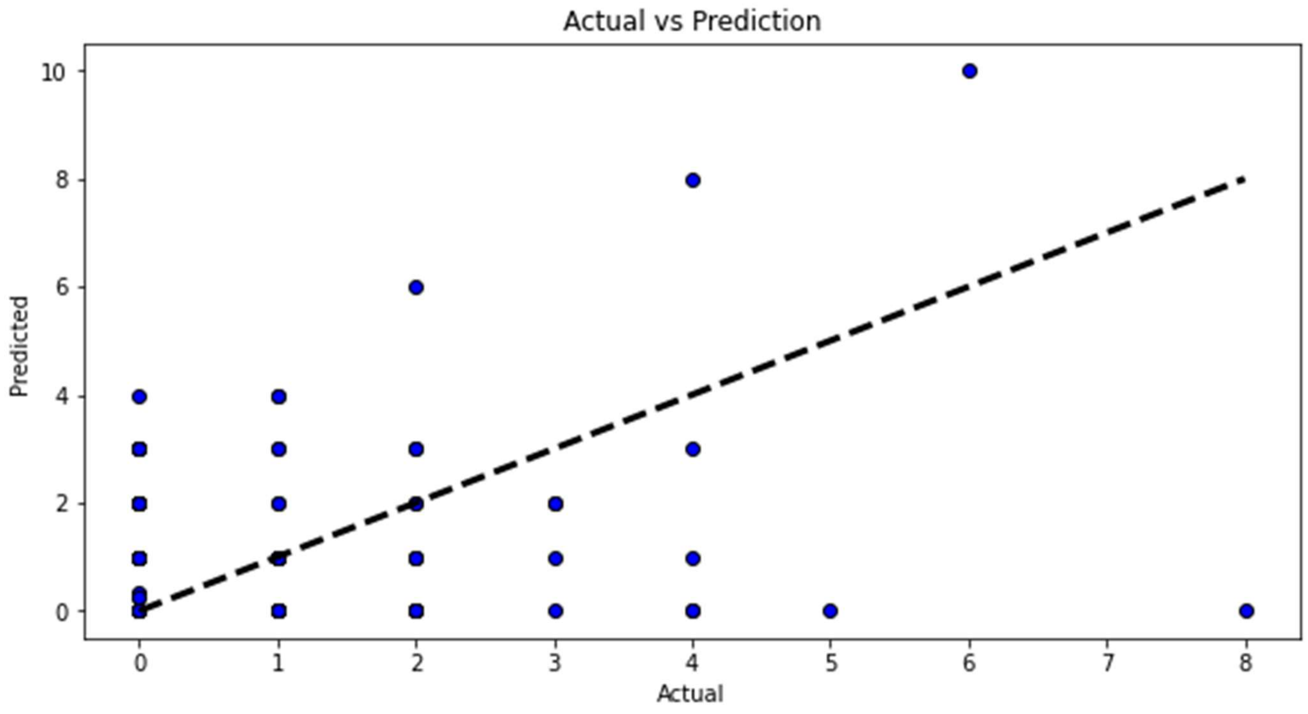

Figure 5.2 Ant 1.4 predicted vs actual count.

Figure 5.2 shows that how actual counts are varying with the predicted counts in a particular dataset. Here, ANT 1.4 dataset has been taken and respectively it can be seen that predicted counts are not linearly varying or are matching with the actual counts. Like predicted 2 but actual 3 defects are present.

Table 5.5 displays accuracy of several datasets across multiple methods. The table clearly shows that when we employ ensemble in all datasets, there is an increase in accuracy. The single regressor models are compared to the ensembles in which they were used as base learners. For example, in Ant 1.5, the accuracy is 79.26 while using DT, but it rises to 89.75 when using ensemble.

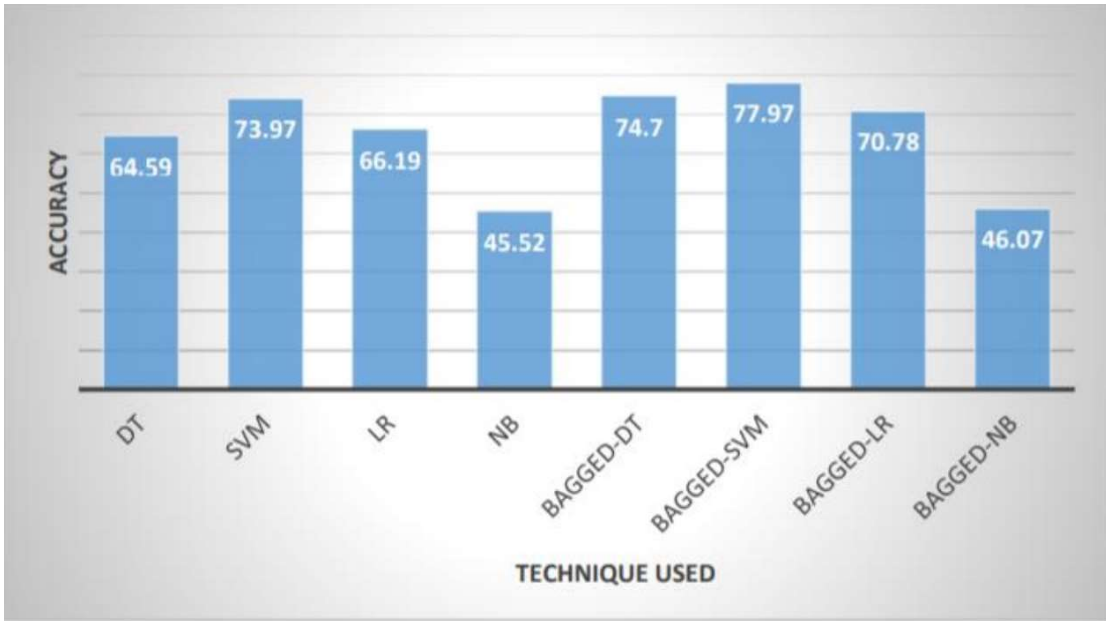

Figure 5.3 Accuracy using Bagging, Ant 1.5

Figure 5.3 depicts graphically how ensemble models of base learners improved when compared to regular learners on the Ant 1.5 dataset. When compared to DT, SVM, and NB, we can see that stacked DT beats and in single base learners the precision of LR is high.

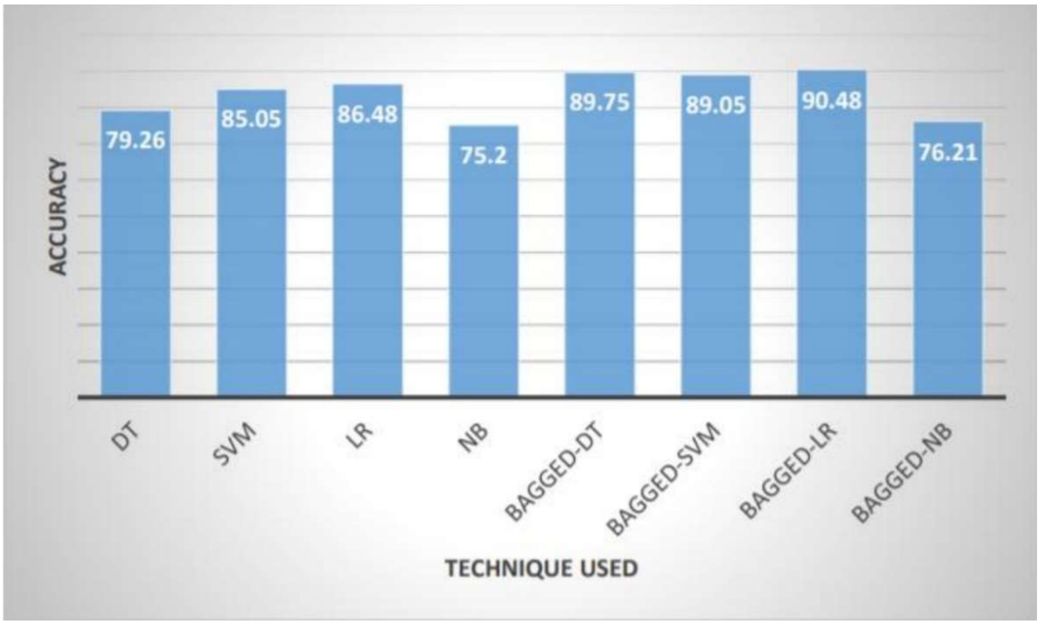

Figure 5.4 Accuracy using Bagging, Ant 1.6

Figure 5.4 depicts graphically how ensemble models of base learners improved when compared to regular learners on the Ant 1.6 dataset. When compared to DT, SVM, and NB, we can see that stacked DT beats and in single base learners the precision of Bagged-LR is high.

As of hardware fault detection we were successfully able achieve our objective of exploring the power line dataset through visual exploration. Also, machine learning models were able to predict the faulty lines as they were intended to be able to. Future study might help uncover search tactics for the prediction of defects and how their capabilities can be used to broaden the representation of the SDP model.

When compared to the other approaches, we can see that casting a vote ensemble is outflanking. It depicts graphically how the ensemble models of various base learners have increased accurateness as compared to ordinary learners on the Ant 1.3 dataset, with the exception of the LR where the AUC score is the same.

We have shown that supported DT beats and single base learners demonstrate a high level of NB as compared to DT, LR, and SVM. When we compare the ensemble tactics to one another, we may divide them into two categories:

1. **Homogenous Ensembles:** In these the models are of same kind.
2. **Heterogeneous Ensembles:** In these the level models need not to be of same kind

It depicts accuracy of bagging and bosting over distinct variants. shows the positions obtained by the Friedman test. This will display the packed results. The position of bagged LR is particularly notable here, i.e., bagged LR is the most appropriate bagging process for our problem out of all other procedures. The heterogeneous ensembles on the distinct versions of the dataset are depicted. The outcome can be seen in terms of increased accuracy of the same model when "bagging" is applied.

To be precise a relevant increase of 14% can be seen in one of the cases where ant 1.4 is worked upon by a DT regressor. Logistic regression has not been that better but when ensemble is created its accuracy is increased i.e., from as low as 72% to as high as 87%. A lot of promise has been observed in Naïve Bayes regressor. This is because it shows potential on this particular dataset (ant) with high accuracy of 80% and when ensembled up to 90%.

Table 5.5 displays how these multiple techniques do vary their accuracies in predicting the results when associated with application on different versions of data. We have utilized the Camel dataset as well and its multiple versions. This has been done to showcase how varying techniques can prove better a particular version of data.

Table 5.5: Accuracy obtained using cross-validation

Technique	Ant 1.3	Ant 1.4	Ant 1.5	Ant 1.6	Ant 1.7
Decision Tree	57.75	75.16	64.59	79.26	71.81
Naïve Bayes	83.00	73.20	60.35	75.20	65.35
Linear Regression	75.26	77.72	66.19	86.48	63.56
Bagged-Decision Tree	88.57	83.07	74.70	89.75	81.61
Bagged-Naïve Bayes	82.19	81.15	46.07	76.21	72.47
Bagged-Linear Regression	81.02	89.09	70.78	90.48	80.76

CHAPTER 6

CONCLUSION AND FUTURE WORK

This research focus on the Accuracy and AUC score, which are the OO execution metrics of two homogenous ensemble learners boosting and sacking with variations in their base learners. This study was conducted on four primary learners in relation to open-source Java ventures using ANT forms in relation to SDP. When compared to individual learners, our large discoveries in bagging and boosting have more accuracy. Showing only bagged SVM beat on three out of five datasets, but bagged DT and dismissed LR beat on ANT 1.5 and 1.6 separately. Given that boosting only helped SVM beat on two of the five datasets, it helped LR beat on the remaining three viz. ANT 1.5, 1.6 & 1.7.

When compared to the base learners that were used in them, our diverse ensembles stacking and voting beat. We detect an increase in both accurateness and AUC score with the aid of groups of learners bagging, bosting, voting, and stacking. As a result, they aided in picking up the exhibition increase in basic learners. Using the Friedman Test, we discovered that bagging and boosting worked better using Logistic Regression as the base learner. Fault detection is presently being monitored using real-world scenarios.

We have seen how "ensemble learning" is used in this defect detection and removal procedure. Some machine learning methods were discussed briefly here. We also can observe how such fault detection technologies may be used to oversee electrical cables that typically have several problems during their lifetime. In addition, we observed how a software might contain flaws that can be found using an ensemble learning technique.

We may be able to estimate the number of errors in a particular software system by doing suitable dataset exploration and approach selection. Different detecting methods were considered. Methods based on data were less complicated. Ensemble Models were more appropriate. Regressors like Decision Tree (DT) have shown high accuracy. It has been observed that ensembling has proved beneficial.

REFERENCES

[1] T. Gyimothy, R. Frenc and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open-Source Software for Fault Prediction," *IEEE Transaction on Software Engineering*, vol. 31, pp.897-910, 2005.

[2] S. Chidamber and C. Kemerer "A Metric Suite for Object Oriented Design." *IEEE Transactions on Software Engineering,* vol.20, no.6, pp. 476-493, 1994.

[3] R. Malhotra, "An Empirical Framework for Defect Prediction Using Machine Learning Techniques with Android Software," *Applied Soft Computing*, vol. 49, pp. 134-1050.

[4] R. Malhotra, S. Shukla and G. Sawhney, "Assessment of Defect Prediction Models Using Machine Learning Techniques for Object Oriented Systems", *2016 5th International Conference on Reliability, Infocom Technologies and Optimization,* 2016.

[5] C. Catal, B. Diri and B. Ozumut, "An Artificial Immune System Approach for Fault Prediction in Object-oriented Software," *In 2nd International Conference Dependability Compution System*, pp. 1-8, 2007.

[6] Y. Zhou and H. Leung, " Empirical Analysis of Object-Oriented Design Metrics for Predicting High Severity Faults," *IEEE Transactions on Software Engineering,* vol. 32, no. 10, pp. 771-784, 2006.

[7] B. Adhikari, "Heart Disease Prediction Using Ensemble Model," *2nd International Conference on Sustainable and Expert Systems,* Genetic Programming, vol. 98, pp. 658-664, 1998.

[8] G. G. Robertson and R. L. Riolo, "A tale of two classifier systems," Machine Learning, vol. 3, pp. 139-159, 1988.

[9] R. L. Riolo, "Lookahead planning and latent learning in a classifier system," From animals to animats, pp. 316-326, 1991.

[10] P. Singh and A. Chug, "Software Defect Prediction Analysis Using Machine Learning Algorithms", *In International Conference on Computing and Data Science,* 2017, pp.775-781.

[11] N. Q. M. Noor, N. N. A. Sjarif, N. H. F. M. Azmi, S. M. Daud and K. Kamardin, "Hardware Trojan Identification Using Machine Learning-based Classification," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, pp. 23-27, 2017.

[12] K. K. Nagwanshi and S. Dubey, "Statistical Feature Analysis of Human Footprint for Personal Identification Using BigML and IBM Watson Analytics," *Arabian Journal for Science and Engineering*, pp. 1-10, 2018.

[13] W. Z. Liu and A. P. White, "The importance of attribute selection measures in decision tree induction," Machine Learning, vol. 15, pp. 25-41, 1994.

[14] M. Li, L. Zhen and X. Yao, "How to read many-objective solution sets in parallel coordinates [educational forum]," *IEEE Computational Intelligence Magazine*, vol. 12, pp. 88-100, 2017.

[15] M. Kessel, P. Ruppel and F. Gschwandtner, "BIGML: A location model with individual waypoint graphs for indoor location-based services," *PIK-Praxis der Informationsverarbeitung und Kommunikation*, vol. 33, pp. 261-267, 2010.

[16] A. Hussain and R. Vatrapu, "Social data analytics tool (sodato)," *in International Conference on Design Science Research in Information Systems*, 2014.

[17] R. E. Hoyt, D. H. Snider, C. J. Thompson and S. Mantravadi, "IBM Watson analytics: automating visualization, descriptive, and predictive statistics," *JMIR public health and surveillance*, vol. 2, p. e157, 2016.

[18] J. H. Holmes, D. R. Durbin and F. K. Winston, "The learning classifier system: an evolutionary computation approach to knowledge discovery in epidemiologic surveillance," *Artificial Intelligence in Medicine*, vol. 19, pp. 53-74, 2000.

[19] J. H. Holland, L. B. Booker, M. Colombetti, M. Dorigo, D. E. Goldberg, S. Forrest, R. L. Riolo, R. E. Smith, P. L. Lanzi, W. Stolzmann and S. W. Wilson, "What Is a Learning Classifier System," *in Learning Classifier Systems*, Berlin, 2000.

[20] J. H. Holland, K. J. Holyoak, R. E. Nisbett and P. R. Thagard, Induction: Processes of inference, learning, and discovery, *MIT press*, 1989.

[21] J. S. Hallinan, "Data mining for microbiologists," *in Methods in Microbiology*, vol. 39, Elsevier, 2012, pp. 27-79.

[22] D. E. Goldberg and J. H. Holland, "Genetic Algorithms and Machine Learning," Machine Learning, vol. 3, pp. 95-99, 01 10 1988.

[23] R. L. De Mántaras, "A distance-based attribute selection measure for decision tree induction," Machine learning, vol. 6, pp. 81-92, 1991.

[24] C. Seiffert, T. M. Khoshgoftaar and J. V. Hulse, "Improving Software Quality Predictions with Data Sampling and Boosting," *IEEE Transactions on Systems Man and Cybernetics Part A*, vol. 39, no. 66, pp. 1283-1294, 2009.

[25] R. Malhotra, "Systematic Literature Review of Machine Learning Techniques for Software Fault Prediction," *Applied Soft Computing*, vol. 27, 2015, pp. 504-518.

[26] M. S. Acharya, A. Armaan and A. S. Antony, " A Comparison of Regression Models for Prediction of Graduate Admissions," *in ICCIDS 2019: IEEE International Conference on Computational Intelligence in Data Science,* Chennai India, 2019.

List of Publications

[1] M. Yadav and R. Malhotra, “A Systematic Review on Fault Detection in Hardware and Software Systems" in Ch. 14, IOP Publishing, 2023.

[2] M. Yadav and R. Malhotra, "Analyzing Performance of Tree Based Regressors on Defect Prediction Datasets," in Vol. 22, No. 1, Telematique, 2023, pp. 242-252.

[3] M. Yadav and R. Malhotra, "Descriptive Analysis of Defect Prediction Techniques," in Vol. 20, No. 20, Neuroquantology, 2022, pp. 11944-11950.

[4] M. Yadav, "A Security-focused Review on Virtual Machines," in Vol. 5, No. 1, JIER, 2022, pp. 5-10.