M. Tech (Software Engineering) SAKSHI 2019

Software Defect Prediction Using Ensemble of Machine Learning Techniques

A DISSERTATION

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

MASTER OF TECHNOLOGY
IN
SOFTWARE ENGINEERING

Submitted by
Sakshi
2K17/SWE/15

Under the supervision of

DR. RUCHIKA MALHOTRA
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road,
Delhi-110042

JUNE, 2019

Department of Computer Science and Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road Delhi-110042

CANDIDATE’S DECLARATION

I Sakshi, 2K17/SWE/15 of Master of Technology (Software Engineering) hereby declare that the Major Project-II Dissertation titled **“Software Defect prediction using Ensemble of Machine Learning Techniques”** which is submitted by me to the Department of Computer Science and Engineering, Delhi Technological University, Delhi in partial fulfillment of requirement for the award of degree of Master Of Technology (Software Engineering) is original and 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 **Sakshi**

Date: **2K17/SWE/15**

Department of Computer Science and Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Bawana Road Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "**Software Defect prediction using Ensemble of Machine Learning Techniques**" submitted by Sakshi (2K17/SWE/15) to the Department of Computer Science and 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 **DR. RUCHIKA MALHOTRA**

Date: **(Supervisor)**

Associate Professor

Discipline of Software Engineering

CSE Department

Delhi Technological University

(Formerly Delhi College of Engineering)

Shahbad, Daulatpur, Bhawana Road Delhi-110042

ACKNOWLEDGEMENT

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

My thanks is addressed to my mentor Dr. Ruchika Malhotra, Department of Computer Science and 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:

Sakshi

M.Tech (SWE)-4th Sem

2K17/SWE/15

ABSTRACT

Now a days research on software defect prediction has attracted many researchers because it helps in creation of successful software. Additional advantage is that it helps in reduction of the software development cost and facilitates procedures to identify the reasons for determining the percentage of defect-prone software in future. For specific types of machine learning, there is no conclusive evidence that will be more efficient and accurate in predicting software defects. Some of the previous related work, however, proposes the learning techniques of the ensemble as a more precise alternative. This work introduces the resample technique with three types of ensemble learners; boosting, bagging, stacking and voting using four base learners on different versions of same dataset repository provided in the PROMISE repository. Results indicate that accuracy has been improved using ensemble techniques more than single leaners.

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	Genralized Linear Model
KNN	K-nearest Neighbour
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 project managers as they help in quantitative planning and management of project. Further, the availability of public software metrics data repositories has opened new areas for research in development, application and evaluation of machine learning techniques for software defect prediction models based on software metrics. Further this chapter explains about the problem of defect prediction and how that is catered by the ensemble learning using predictive modelling.

1.1 DEFECT PREDICTION

Software defect (or fault) prediction is considered to be one of the most cost-effective and also very useful tool which let us know if a particular module is having defect or not. Software practitioners see it as a crucial stage for ensuring the quality of the process or the product which is to be developed. It played a very crucial role in bringing down the claims about the software industry that it is unable to meet the requirements in the budget and on time.

The enormous investment and money spent on software engineering development leads to an increase in software system maintenance costs. Today, the enormous size of the software developed is becoming increasingly complex. A large number of program codes, too. To this end, the probability of software deficiencies has been increased and methods of quality assurance are not sufficient to overcome all software deficiencies in huge systems. Identifying which modules are most likely to be defective in the software can therefore leads to reduction in the limited resources as well as development time.

One of the effective way to enhance the quality of software is to predict software defects, which is also an effective way to relieve the effort of inspecting or testing software code. Under this circumstance, only part of the software artifacts need to be inspected or tested and the remaining ones ignored.

Resolving a defect, or fault, leads to exponential increases if it penetrates to the subsequent stages of a software development lifecycle. The advantage of identifying software faults in the initial stages not only yields fewer faults and an enhanced software w.r.t quality, but also helps in developing of a cost-effective model. In addition, we only focus on the weaker modules during testing and maintenance phases which eventually leads to effective development of model. Thus, the limited resources of an organization could be reasonably allocated with the objective of detection and correction of the maximum number of software defects. Therefore, this topic of prediction of the software faults has been analyzed extensively and a lot of methods have been suggested to address this problem.

1.2 ENSEMBLE OF MACHINE LEARNING

Ensemble methods seem to be meta-algorithms that are fusion of several techniques of machine learning into one predictive model to improve predictions (voting), decrease bias (boosting), or decrease variance (bagging). Ensemble learning is a method that involves certain classifiers in which a predictor is constructed using bunch of classifiers may or may not be of same kind and then by taking a weighted vote or the averaged result of their outputs, new data points are classified.

A number of learners are used to make an ensemble model, called base learners. An ensemble model's ability to generalize is usually better than base learner’s ability. What makes it appealing is the ability of ensemble models to boost weak learner’s performance and also to make them strong learners that produce far more precise predictions. Therefore, base learners in ensemble learning are also designated to as “weak learners”. Worthy of note, that although weak learners are the basis for most theoretical analysis, base learners or the weak learners used in actual scenario don’t have to be weak every time, as the quality and prediction base learners that are not so weak is often much better.

To obtain better predictive performance, multiple learning algorithms are used than could be obtained from any of the constituent learning algorithms alone. To evaluate an ensemble model's diction, more computation is needed than evaluating a single model's predictive power.

1.3 PREDICTIVE MODELLING

Statistical methods or Machine Learning (ML) techniques are used to create models in predictive modelling. Data used to know as historical data, is extracted from the past and is used to predict future results. Predictive modeling is a process in which models are created to estimate results. Each model consists of independent variables (predictors) and dependent variables (outcome). The primary aim of predictive modelling is to discover relationships between both the independent and dependent variables that cause changes in other variable because of one variable.

1.4 ORGANIZATION OF THE THESIS

In this thesis we aim to find the best methods for the problem of the SDP. New methods are explored and compared with the conventional predictors.

The current chapter contains the general summary of the research. Chapter 2 contains the literature survey behind the research. Further, Chapter 3 introduces about the crucks of ensemble learning as well as the base classifiers that are used in the study. It also contains the detailed description of the OO metrics that have been used in the dataset. Chapter 4 presents the ensemble learners that were used for the imprudent in the accuracies like bagging, boosting etc. Chapter 5 summarizes the results obtained and graphically shows the observations that are drawn from the study. Chapter 6 presents the conclusion and the future scope of the research project.

CHAPTER-2

LITERATURE REVIEW

The literature studies that have been surveyed suggests extensively effective models for the prediction of defects. The initial work in prediction of software defects focus mainly about the use of statistical techniques. The summary of the studies that were used in this research are discussed below.

2.1 LITERATURE REVIEW

Many software studies have investigated fault prediction in a software. But here we will only consider those studies that use ML techniques to predict defects based on OO metrics. The purpose of Chidamber & Kemerer (CK)[2] metrics is to measure whether or not a piece of code follows OO principles. Gyimothy et al.[1] used Decision Tree(DT) and machine learning techniques such as logistic regression and neural network to find the relationship among CK metrics and prediction of defects.

The study by Singh et al.[15]advocated using these algorithms to defect-prone software parts. The study also recommended carrying out a large number of studies to determine the predictive capacity of ML algorithms in this domain. In that study there was a comparison made between the statistical model and the ML techniques and it was concluded that ML techniques perform better than the traditional statistical algorithms in the domain of predictive modeling. Another recent study by Malhotra [3] evaluated Android package algorithms capability of 18 ML. The results indicated that some algorithms like Logiboost and Naïve Bayes proved to be superior than others. Along with that MLP also proved good for the domain of the defect prediction.

Catal et al. [5] analyzed the artificial immune recognition system to validate the data set of NASA KC1. Malhotra et al. [4] statistically evaluated the aptness of 17 machine learning algorithms for predicting defects and evaluated their validation on the releases of the Xerces data set. The above studies used ML techniques to examine the dependencies among the defect prediction and the OO metrics.

In the prediction of defects, a number of machine learning techniques were investigated by Malhotra [25]. The nature of data sets for defect prediction is skewed. By skewed we mean that data is imbalanced i.e. the non-faulty modules are high in number as compared to faulty ones. Non-defective modules are negative examples (or negative class or majority class) in terms of machine learning literature, and defective modules in training data are positive examples (or positive class or minority class). This is referred to as the problem of class imbalance. Class imbalance greatly degrades the performance of machine learning techniques. Seiffert et al. [24] suggested data sampling to be one the solution for this problem.

Zhou and Leung [6] assessed the usefulness of CK metrics for predicting defects in terms of severity of defects. They validated two severity levels on NASA's KC1 data set using techniques such as Random Forest, Naive Bayes, LR. It has been evaluated that the metric number of children seems to be of little importance in the prediction of defects. The results showed that the CK metrics had certain limitations to predict class with high severity errors. Also, low performance was achieved by the models built using ML techniques.

Chug and Singh [11] reviewed five machine learning algorithms used to predict early software deficiencies, i.e. Artificial Neural Network (ANN), Linear Classifier (LC), Decision Tree (DT), Particle Swarm Optimization (PSO) and Naïve Bayes (NB). Results of this study show that, in prediction accuracy, the linear classifier is better than other algorithms, but the lowest error rate is for ANN and DT algorithms. NASA dataset such as inheritance, cohesion and Line of Code (LOC) metrics are the popular metrics used.

However, there were considerably fewer studies about the usage of these ensemble learning methodologies specifically for the defect data where we can predict the defects. The results produced using ensemble learning methods are presented in this thesis and we also include a comparative analysis with the previously deployed machine learning techniques.

CHAPTER-3

TECHNIQUES AND OBJECT ORIENTED METRICS

In recent times, the ensemble learning techniques are gaining importance in the field of defect prediction because of its increased accuracy and performance. Also, the results that are obtained in the research have validated that they are having higher validation accuracy as compared to single base learners in SDP. The independent variables which are used for predicting defect prone classes in this study are OO metrics that are explained in detail under this module and the dependent variable (DV) is fault proneness which is binary in nature.

3.1 ENSEMBLE LEARNING MOTIVATION

The motivation behind using the ensemble machine learning techniques are due to several reasons. Ensemble models have been proved very effective to uplift the accuracy and the performance of the models.

Some machine learning techniques perform a local search rather than finding the global optima, which quite often gets trapped in local optima. For example, the algorithm for the decision tree uses a splitting rule for greedy methods to grow the tree. By contrast, an ensemble built from several different starting points by running a local search usually tend to provide a better prediction of the true unlabeled sample than any of the individual classifiers taken separately.

A learning algorithm can be viewed as searching for a space H of hypothesis in order to identify the best hypothesis in space. However, the statistical issue arises that the amount of data available to train the model is too small compared to the size of the hypothesis space. Without sufficient data, many different hypotheses can be found in a learning algorithm in H which when used with training data usually provides the same accuracy. By creating an ensemble of all these precise models, the algorithm can have weighted-average of their votes and provides a reduction in the likelihood of selecting the inappropriate classifier for prediction.

3.2 BASE CLASSIFIERS

Ensemble models are created using the conventional base learners but results in improved accuracy and performance of the model. The models used in this study have been described below along with their pros and cons.

3.2.1 SUPPORT VECTOR MACHINES

A Support Vector Machine (SVM) is interpreted experimentally as a discriminatory classifier by a separate hyperplane alternatively we can understand SVM as, given the supervised learning data (labeled training data), the algorithm that classifies new examples produces an ideal hyperplane. This hyperplane itself is a line that separates a plane of data points into two areas into the two dimensional spaces where it sits in each class on either side.

It uses the kernel trick generally to classify data that cannot be classified linearly. The algorithm's main objective is to predict a plane that maximizes class distance in order to reduce the possibility of overfitting and reduce the likelihood of misclassification of the new data point.

Figure 3.1 shows that the selection of optimal hyperplane should be done in such a way that the margin can be maximized between the support vector

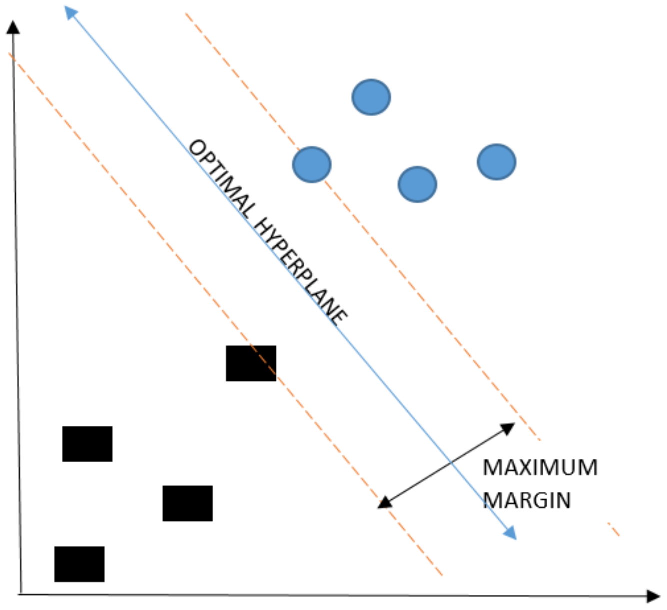


Fig 3.1: SVM Hyperplane

3.2.2 NAÏVE BAYES

Naive Bayes is a sort of classifier that makes the use of Bayes theorem. It calculates membership likelihoods for each class, such as the likelihood that a particular class belongs to a given record or data point. The most probable class is the class with the highest probability. This is also referred to as Maximum A Posteriori (MAP). Equation i represents the relation between hyp and evd.

Maximum(P(Hyp/Evd)) =

Maximum((P(Evd/Hyp)*P(hyp))/P(Evd)) = **….(i)**

Maximum(P(Evd/Hyp)*P(Hyp))

where Hyp: Hypothesis

Evd: Evidence

So, the crucks of naïve Bayes are that the classification of the Naïve Bayes depends as a simple classifier on the Bayes rule theorem of conditional probability [7]. It assumes the values of attributes are independent and unrelated, it is called the model of independent feature. In many of the applications, Naïve Bayes uses the maximum probability methods to estimate its parameters [8].

3.2.3 DECISION TREES

In real situations, a tree has several similes and tries to turn out to have impacted a wide range of ML techniques spanning both regression and classification. A decision tree can also be used in predictive analysis to portray judgements and judgment-making visually as well as explicitly. As the name suggests a tree-like structure is used for making decisions where at every node we are supposed to make decisions which eventually leads to the desired output.

The significance of the feature is evident and interactions can be easily perceived. This technique is more usually referred to as data learning decision tree and tree is called classification tree as the objective is to categorize tuple as class 1 or class 2. Regression trees are defined in the very same way, only predicting continuous values such as a house's price. In terms of views, both CART or Classification & Regression Trees are referred to as Decision Tree algorithms.

3.2.4 LOGISTIC REGRESSION

Binary dependent variable(DV) are those where the output can only take binary values and are used to represent such positive/negative outcomes.

Multinomial logistic regression is used to analyze cases where there are more than 2 outcomes of dependent variables. In the regression model used in Logistic Regression, the DV is categorical.

Generalized linear model (GLM) is a super algorithm class that includes Iinear regression. In 1972, Nelder and Wedderburn proposed a model with the aim of providing a means to use linear regression to problems that were not directly suited to linear regression application. With the help of logistic function,the relationship between the categorical dependent variable and independent variables is measured using logistic regression.

The fundamental equation of GLM is given by ii:

$$\boxed{\mathbf{g(E(y)) = \alpha + (\beta * x1) + (\Upsilon * x2) \qquad \ldots.(ii)}}$$

where g() is the link function , E(y) represents the expectation of the target variable and **$\alpha + (\beta*x1) + (\Upsilon*x2)$** is the linear predictor and **α,β,Υ** are to be predicted. The link function is used for the linkage of expectation of y to that of linear predictor.

Logistic Regression used by Basili et al. [9] to determine the dependencies among metrics and class defectiveness. They also used the univariate method to assess each metric in isolation. This was enhanced by performing multivariate regression to evaluate those metrics ' predictive ability.

Briand et al. [10] statistically developed the use of the sub-set of metrics in predictive failures and reached the conclusion that coupling and inheritance measures are closely linked, whereas cohesion has no major impact.

3.2.5 RANDOM FOREST

The random forest starts with the random selection of n features from the complete N features. In the subsequent stage, by making the usage of best split approach, we use the randomly selected' n' features to find the root node. The next stage, we'll use the best split approach to calculate the daughter nodes. The initial 3 sequential stages are reiterated until a root node forms the tree and the target is the node of the leaf.

At last, to create ' n ' randomly created trees, we repeat one to four stages, this randomly created trees form the random forest.

There are several unpruned regression or classification trees in random forests. Using random selection of features, these trees are induced from training data bootstrap samples. Each data sample in the random forest is fed down each of the trees in classification problems. Then, the latter outputs the class that received most of the votes from the individual trees as its decision class.

In order to predict the class using the random forest algorithm, we need to cross the test traits through the rules of each trees that have been created randomly. Suppose we were forming 50 random decision trees to form the RF for the same test feature, each RF will predict different targets. Then every predicted target vote will be considered.

3.2.6 KNN

One of the non-parametric ML methods used for regression and classification is the k-nearest neighbor’s algorithm. The input comprises of the k nearest sample training data points into the feature space in both cases. The function is only locally approximated and deferred all computation until classification, k-NN is a type of instance-based learning, or lazy learning. On comparision with other ML algorithm this is the simplest one.

The output is the label value of the object in the regression K-NN. For calculating its value all we need to do is to take an k-nearest neighbors and average the label values of them. The output is a class membership in the classification algorithm k-NN. To find that to which class this sample corresponds to we have to take the majority of vote form the k-nearest samples and the most prominent class is assigned to that sample point also the value for the k is usally kept small and tends to be positive all time. If we equate k to 1 then the class of its neighbor is used as class of sample because it is closest to that.

3.3 OBJECT ORIENTED METRICS

As the use of OO metrics has gained widespread recognition, the advent of the specific set of metrics have also gained the popularity. The goal of these metrics is to produce high quality results that can be used for the assessment of the complex systems.

One of the example is coupling metrics. Coupling is called the use of methods or attributes defined by another class in a class. If a class interacts with other classes, a subsystem or system can be used to indicate the complexity of the design. These are commonly known as coupling metrics.

Some of the metrics that were used in the study are listed below:

1. **Weighted Methods per Class (WMC**):

   This metric measures the strategies used in a class and calculated by adding up the cyclomatic complexities of all the listed methods used in a class. A class with more methods will become adequate for the program domain, thus restricting class reusability and covering the quality model's maintenance, reusability and under-standardization characteristics.

2. **Depth of Inheritance Tree (DIT):**

   This metric shows the inheritance levels in the class design and, in the inheritance hierarchy, denotes the length of the path from a given class to the root class that seems to be the longest. However, inheritance promotes a class's reusability, but makes it more complex to maintain and debug. This metric focuses not only on the characteristics of efficiency and reusability of a quality model, but also on testability and understandability.

3. **Number of Children (NOC):**

   This metric refers to the inheritance feature, similar to DIT, and calculated by counting the number of inherited immediate child classes from a given class. However, a large NOC value enhances reusability, but makes it hard to test a class. Consequently, this metric relates the characteristics of a quality model's reusability, efficiency, and testability.

4. **Coupling between Object Classes (CBO):**

   This metric reveals one class's dependence over other design classes. Such dependence may occur because of the mechanism or inheritance that passes the message. It is combined with other classes by summing the number of distinct classes related to non-inheritance. This metric influences the characteristics of a quality model's reusability and efficacy.

5. **Response for a Class (RFC):**

   This measure is for the request (message), an object is for other objects and calculated as the number of methods in the set of all methods implemented in the classes that can be called remotely in reaction to a message sent. This conclude the sum of the number of local methods and methods that can be remotely called. Invoking a large number of methods in response to a message makes it more difficult to test and debug. This metric serves a quality model's comprehensibility, maintenance, and testability characteristics. RFC is given by equation iii.

RFC=|RS| where RS is response set **….(iii)**

RS can be expressed as :

RS={ M } U {Ri}, where Ri is the set of methods called by i and M is the set of all methods.

6. **Lack of Cohesion of Methods (LCOM):**

This is one the metric that shows inner cohesion inside class design components. It is estimated by counting the method pairs that do not share the same class instance variables in a class with zero similarity. An improved cohesiveness promotes encapsulation and determines the characteristics of a quality model's efficiency and reusability.

7. **Coupling Afferent (Ca):**

This is the count of the classes in number that are present in some other external packages which is depending upon modules or classes within the package, basically it is an indicator for the responsibility of the package. 0 is more desirable and 1 is undesirable. Afferent implies Incoming.

8. **Coupling Efferent (Ce):**

The number of classes in other packages that the classes in the package depend upon is an indicator of the package's dependence on externalities. 0 is more desirable and 1 is undesirable. Efferent implies Outgoing.

9. **Number of Public Methods (NPM):**

The NPM metric simply counts all the methods in a class that are declared as public.

10. **Lack of cohesion in methods (LCOM 3):** It is estimated by counting the method pairs that do not share the same class instance variables in a class with zero similarity.it is the improved version of LCOM1.

The variation range of LCOM3 is 0-2. Equation iv represents the formula for the same.

$$\text{LCOM3}= \frac{(\frac{1}{a}\sum_{j=1}^{a} \mu(Aj))-m}{1-m} \qquad \textbf{…. (iv)}$$

where m- number of methods in a class

a-number of variables (attributes in a class)

$\mu(A)$- number of methods that access a variable.

11. **Lines of Code (LOC):**

The size of a method is used by developers and maintainers to evaluate the ease of understandability, reusability and maintenance of the code. LOC is the number of active code physical lines (executable lines) in one of the method's code. Size can be measured in a variety of ways. This includes counting all physical code lines, number of statements, etc.

12. Data Access Metric (DAM):

This metric is the ratio of private (protected) attributes to the total number of declared attributes in the class. A high value is desired for DAM. It ranges from 0 low to 1 high.

13. Measure of Aggregation (MOA):

This metric measures the extent of the part-whole relationship performed through the use of attributes. The metric is a count of the number of data statements (class fields) whose types are classes defined by the user.

14. Measure of Functional Abstraction (MFA):

This metric is the ratio of the number of methods that a class inherits to the total number of methods available through the class's member methods. The constructors are ignored as well as the java.lang.Object (as parent). It ranges from 0 low to 1 high.

15. Cohesion Among Methods of Class (CAM):

This metric calculates the relationship between class methods based on the method parameter list. The metric is calculated by summing the number of different types of parameters of method in each method divided by multiplying the number of different types of method parameters in the whole class and number of methods. It is preferred to have a metric value close to 1.0. It ranges from 0 low to 1 high.

16. Inheritance Coupling (IC):

This metric provides the number of parent classes that are coupled to a given class. If one of its inherited methods functionally depends on the new or redefined methods in the class, a class is coupled to its parent class. If one of the following conditions is met, a class is coupled to its parent class:

a) A redefined method calls one of its inherited methods and uses a parameter defined in the redefined method.
b) A variable (or data member) defined in a new / redefined method is used b y one of its inherited methods.
c) A redefined method is called by one of its inherited methods.

17. Coupling Between Methods (CBM):

The metric measures the total number of new / redefined methods that are combined with all the inherited methods. When one of the conditions specified in the IC metric definition holds a coupling.

18. Average Method Complexity (AMC):

The average method size for each class is measured by this metric. A method's size is equal to the method's number of java binary codes.

19. The McCabe's cyclomatic complexity (CC):

In a method (function) plus one, it is equal to the number of different paths, which is called cyclomatic complexity. The complexity of the cyclomatic process is defined as:

CC= No. of EDGES – No. of NODES + No. of CONNECTED COMPONENTS

where edges, nodes will correspond to the graph whose complexity is to be obtained.

CHAPTER-4

ENSEMBLE OF CLASSIFIER MODELS

A classifier ensemble came into being to overcome the drawbacks of a single classifier model. More often than not, the single classifier system faces the problem of overfitting and bias in the classifier. Ensemble classifiers have demonstrated a very good performance to overcome such scenarios.

In this study, we evaluated the performance of different set of classifier strategies on the Ant dataset along with their diverse versions. We compared the responses of different methods with regard to the metric of accuracy performance.

A classifier ensemble came into being to overcome the drawbacks of a single classifier model. Most often, the single classifier system faces the problem of overfitting and bias in the classifier.

4.1 Bagging

Bagging is an abbreviation for Bootstrap Aggregating. Bagging was introduced by Breiman. The idea of bagging is simple, that is, the ensemble consists of classifiers that are based on the training set's bootstrap replicas. The outputs of the individual classifier are combined using the combination rule of majority voting.

In this bootstrap sampling is used in which subset of data points are selected at random from the space of data points with labels. The main underlying principle is that the samples are picked up with replacement, so we can say that a data point that has been picked up earlier has equal probability of being chosen again like other data points which were not picked up earlier.

These samples or we can say that the bootstrapped samples are then forwarded to an aggregator which counts the vote that how much vote a class is having. The class with majority votes is considered to be the predicted label for that unknown sample.

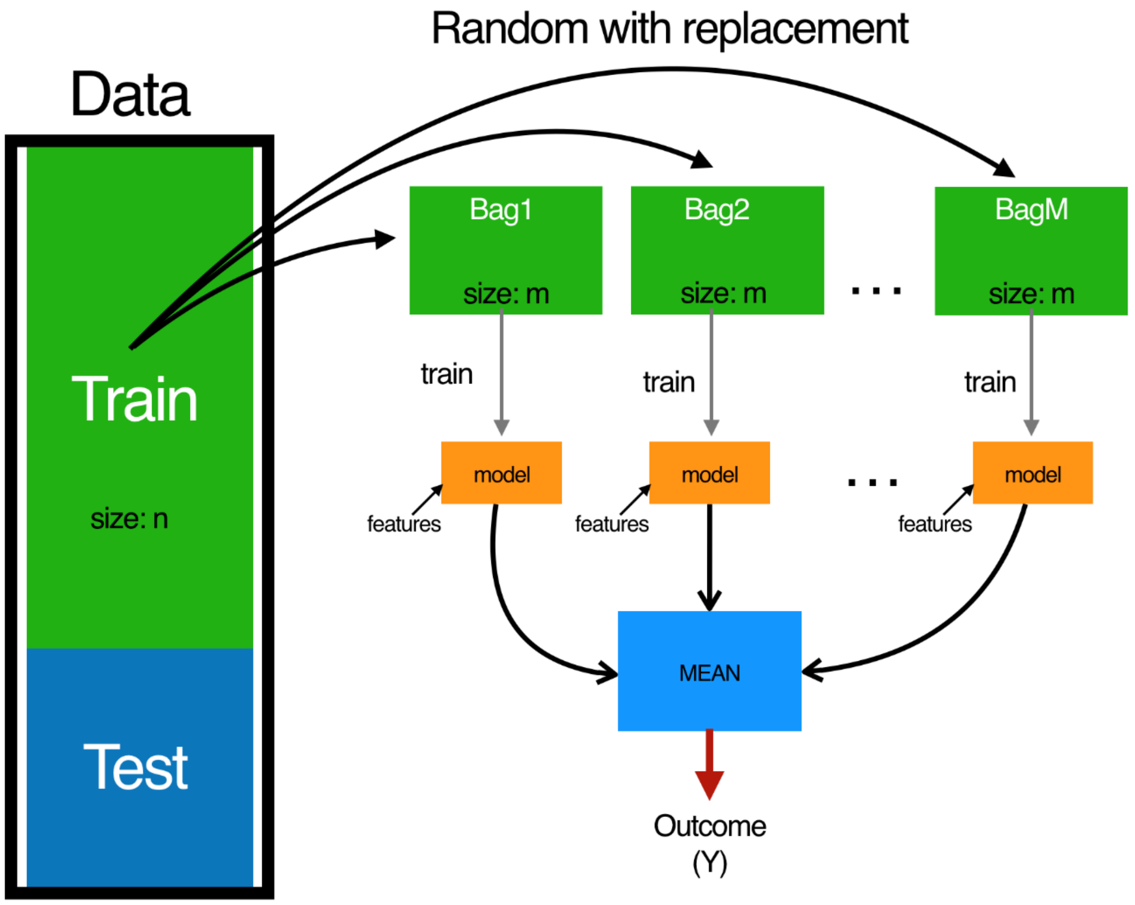


Fig 4.1: Learning with Bagging

We can have more than one bootstrapped sample which will be used for the training purpose. In our study the models that we have used are the homogenous models. The performance of the bagging classifiers can be improved by varying the base classifiers. In this we have chosen 4 different base classifiers for the bagging technique. Following are the classifiers:

1. Bagged Decision Tree
2. Bagged SVM

3. Bagged Logistic Regression
4. Bagged Naïve Bayes

Algorithm for Bagging:

1. Parameters are initialized
   - D = φ, where D is the ensemble.
   - L, the number of classifier to train.
   - Parameters are initialized

2. For k = 1 to L
   - Take a bootstrapped sample S_k from Z.
   - Build a classifier D_k by using S_k as the training set.
   - Add classifier to the current ensemble.

3. Return D.

4. For Testing Phase, Run D_1 upto D_L on the input X.

5. The class having majority of votes is labeled as the category of that sample.

4.2 Boosting

The basic idea behind the working of ensemble model is to enhance the predictive power of the model by adding one classifier iteratively at a time. At a particular stage when a classifier enters an ensemble then it is trained on a data set randomly sampled from the training data set. Sample distribution starts with uniformly stable mode and converges its growth towards increasing the prediction of difficult data points.

Boosting brings together weak learners. Alternatively, we can say base learners are created using machine learning algorithms with a different distribution to form a strong classifier with strong rules. Each time a basic learning algorithm is applied, it generates a new rule. In other words, boosting is an iterative technique in which the first algorithm is trained on the whole dataset and the subsequent algorithms are constructed by fitting the residuals of the first algorithm, thus giving greater weight to the observations poorly predicted by the previous model.

AdaBoost is a variant of boosting ensemble leaning method. AdaBoost is acronym for Adaptive Boosting. Initially we start by assigning equal weights to all the data points. If there is any wrong prediction i.e., the error of prediction due to the first algorithm of basic classification, then we pay more attention to those observations with error of prediction. Then comes the application of the next algorithm for learning the base.

Finally, we will iterate the previous step until the limit of the basic learning algorithm is reached or higher accuracy is achieved. Lastly, it combines the weak learner's outputs and creates a strong learner that ultimately increases the model's predictive power.

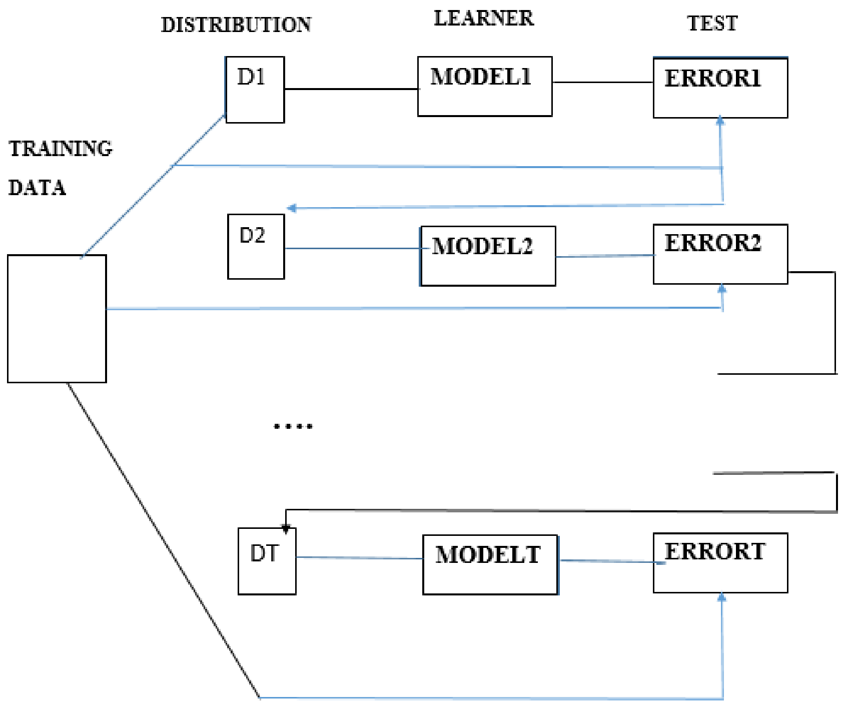


Fig 4.2: Learning with Boosting

In this we have chosen 4 different base classifiers for the boosting technique. Following are the classifiers:

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

Algorithm for Bagging:

1. **INPUT:**
    - A labeled dataset with N data points (provided class label pairs).
    - A learner model (NN, DT, SVM).

2. **Learning stage (Training of the Model)**
    - Depending upon the training dataset D T base models are trained on T different sampling distributions.
    - By doing the modification in the sampling distribution D_{t-1} obtained from t-1th step a sample distribution D_t is built for model t. Data points the were incorrectly identified in the previous attempt are likely to have higher weights in new formed data.

3. **Classification step**
    - According to weighted majority of the class the label value is obtained.

4.3 Voting

Voting is a popular approach of ensemble. Voting combines the decision from multiple models based on a combination rule that turns out to be a different combination of estimates of probability. Models can be of different types, i.e. decisions from either single model classifier, homogeneous model classifier ensemble, or even decisions from any other heterogeneous model of ensemble. The scheme used in voting method is very straight forward and much like the combination technique of majority voting that is used in any other ensembles like bagging or AdaBoost.

In this work, we use voting method with a hard vote probability estimate for experiments. The main difference is that in Bagging or AdaBoost voting scheme acts as a combination rule for final decision making, whereas in voting ensemble method, voting refers to a class or learner who receives the labels as inputs from different sources and uses probability estimates for final decision making.

In this study we have chosen 3 different base classifiers for the voting technique. Following are the classifiers:

1. Decision Tree
2. SVM
3. Logistic Regression

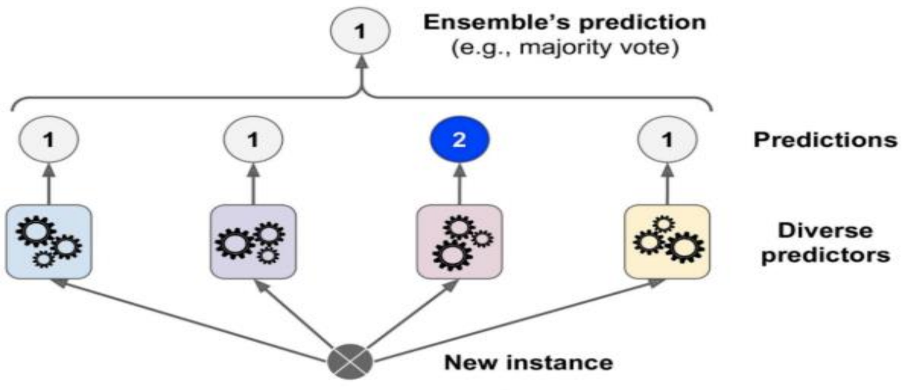


Fig 4.3: Learning with Voting

4.4 Stacking

Stacking is a technique of machine learning(ML) and it is a different model in ensemble studies as it majorly seeks to upgrade the ensemble's accuracy and subsequently the performance by working upon the errors. It addresses the problem of classifier bias with keeping regards to data used for training and focus to learn and use these calculated biases to increase the classification and is considered to be stacked generalization.

Wolpert first proposed Stacked Generalization (or stacking) in 1992, and said that "It is a way to combine multiple models that introduces a meta-learner concept. Even though it is an attractive idea, it is less used in literature than bagging and boosting".

Ensemble methods use n(n>1) models in machine learning to achieve increased classification accuracy than any of the constituent models could obtain. It basically deals with combining the predictions of multiple classifiers that were generated on a common single dataset using different base classifiers. A group of base level-1 classifiers or predictors is generated in the initial stage. A meta-level classifier called the meta-learner is used that combines the predictions of the base level-1 classifier is learned in the second phase.

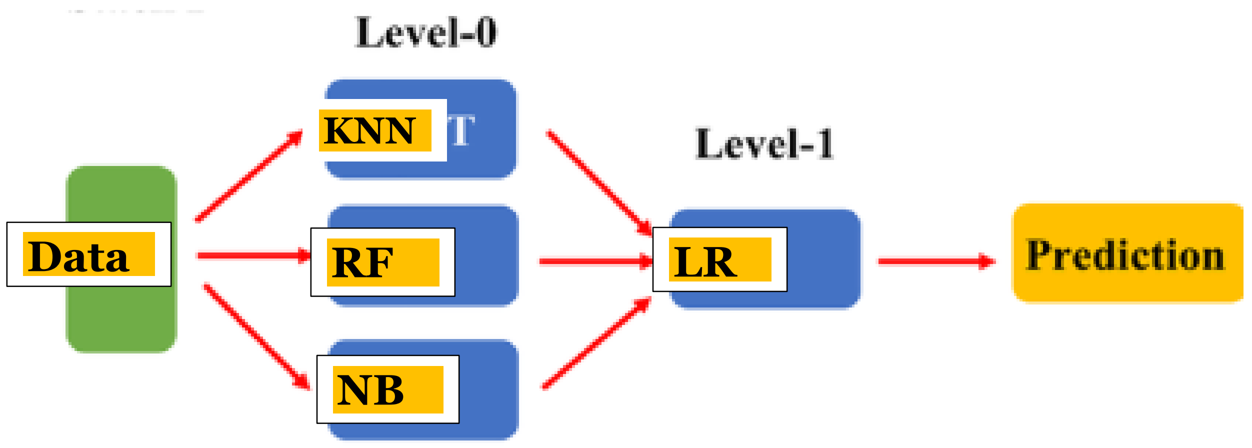


Fig 4.4: Learning with Stacking

In this study we have chosen 3 different base classifiers for the Stacking technique, along with one meta-learner. Following are the classifiers:

1. K- Nearest Neighbors
2. Naïve Bayes
3. Random Forest

Meta Learner: Logistic Regression.

CHAPTER-5

IMPLEMENTATION AND RESULTS

In this work, area under curve (AUC) and the accuracy are used as an evaluation metric to compare the relative change in between ensemble models and single classification models. Results are evaluated over 5 defect prediction data sets. First the data was preprocessed and then was used for analysis using cross-validation technique. The data was fed to the model which was implemented using python.

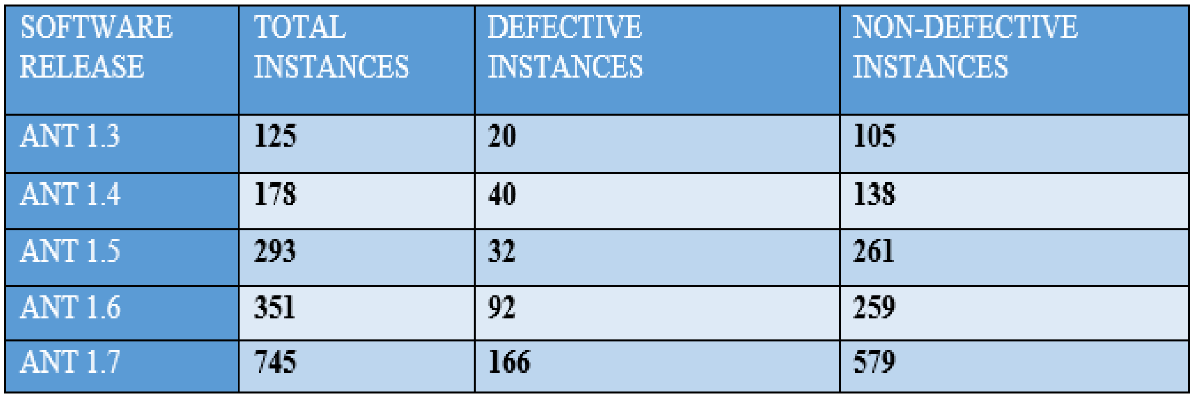
5.1 Data Description

In this study, we have selected different versions of Ant datasets to perform the experiments using the PROMISE repository with different sizes of a number of modules. The dataset description is displayed in the table. This dataset is commonly used for analysis of bug prediction using the software object oriented metrics. Since the data is containing the count of the bugs present in the column of defects therefore the data needs to be preprocessed for the binary classification used in the study.

The data set 'Ant' contains of a binary column, viz bug, this gives us Indication if a class is having defect or not. so the column has been renamed as defects. If the value of the binary column shows 0, then there are no defects. And if the binary column value shows 1 or higher, the class will be having the defects.

Table 5.1 explains the data along with its versions and also the defective and non-defective modules. There are 5 versions/releases of the data which is shown by software release column.

Table 5.1: Data 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 MODEL FLOW

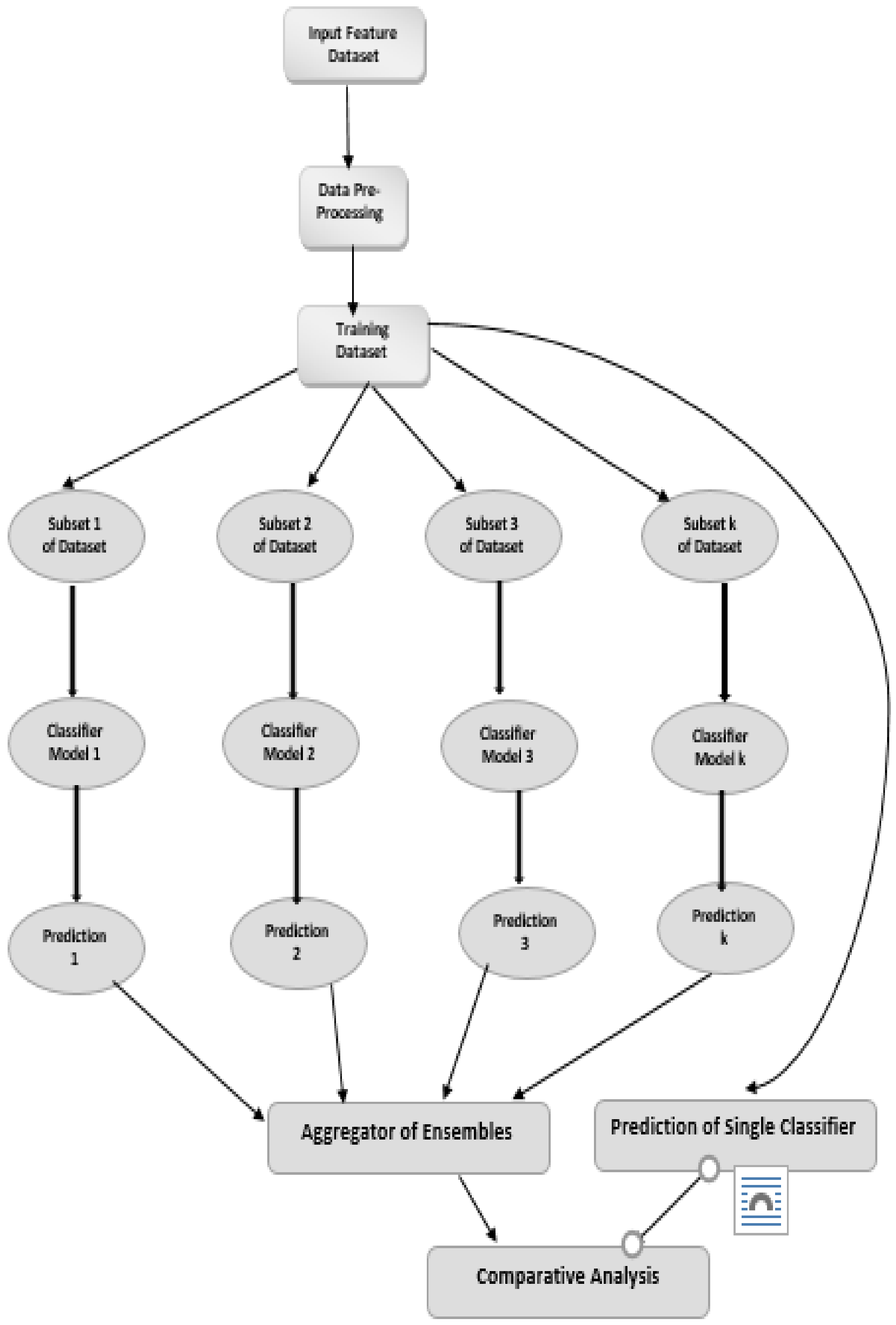


Fig 5.1: Flow chart of the comparative analysis

5.3 RESULTS

The first performance metric used is the Accuracy. It measures the number of correct samples predicted over the total number of samples. For example, if the classifier is correct for 90 percent, it means that it correctly predicts the class for 90 of them out of 100 instances.

Table 5.2 shows the accuracies of the different versions of dataset over different techniques. The table clearly depicts that there is a hike in accuracy when we use bagging ensemble in all the datasets. The single classifier models are compared with the bagged version where they have been used as base learners. For e.g. In Ant 1.5 using single learner DT the accuracy is 79.26 but using bagged-DT accuracy rise up to 89.75.

Table 5.2: Accuracy using 10-fold cross-validation using Bagging

TECHNIQUE USED	VERSIONS OF DATA SET				
	ANT 1.3	ANT 1.4	ANT 1.5	ANT 1.6	ANT 1.7
DT	70.55	64.59	79.26	70.34	74.25
SVM	80.03	73.97	85.05	69.76	73.71
LR	76.7	66.19	86.48	75.21	78.27
NB	73.5	45.52	75.2	74.34	76.8
BAGGED-DT	79.29	74.7	89.75	80.34	81.48
BAGGED-SVM	84.03	77.97	89.05	73.76	77.57
BAGGED-LR	80.7	70.78	90.48	78.35	82.27
BAGGED-NB	79.16	46.07	76.21	78.05	81.06

Figure 5.2 shows graphically that ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.3 dataset. We have observed that bagged-SVM outperformed and in single base learners the performance of SVM is high as compared to DT, LR and NB.

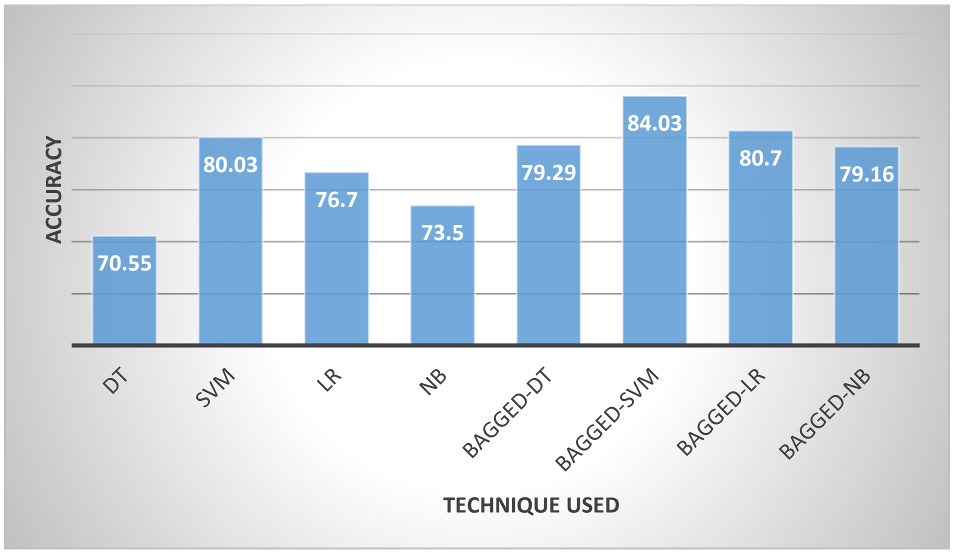


Fig 5.2: Accuracy Using Bagging, ANT 1.3

Figure 5.3 shows graphically that bagged ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.4 dataset. We can observe that bagged-SVM outperforms and in single base learners the performance of SVM is high as compared to DT, LR and NB.

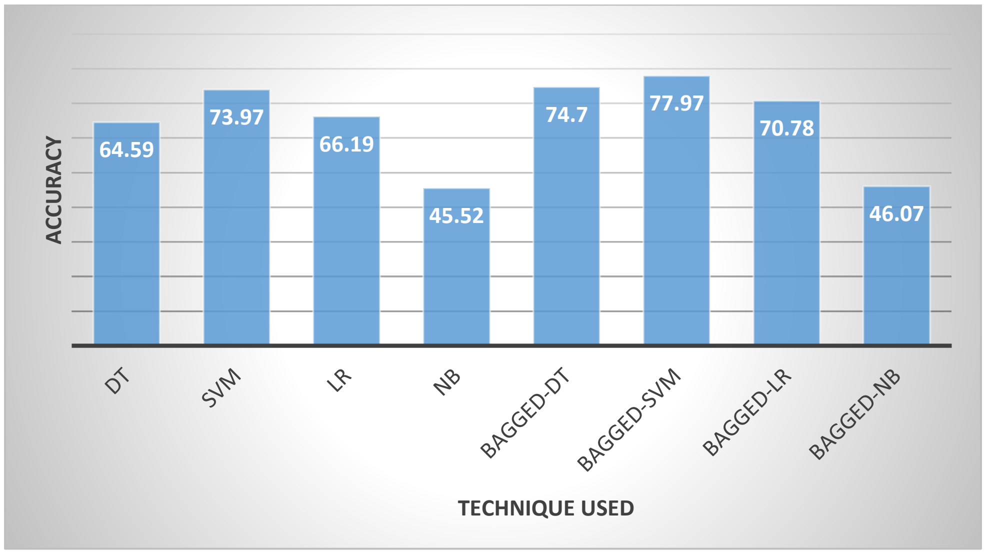


Fig 5.3: Accuracy using Bagging, ANT 1.4

Figure 5.4 shows graphically that ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.5 dataset. We have observed that bagged-LR outperforms also in single base learners the performance of LR is high as compared to DT, SVM and NB.

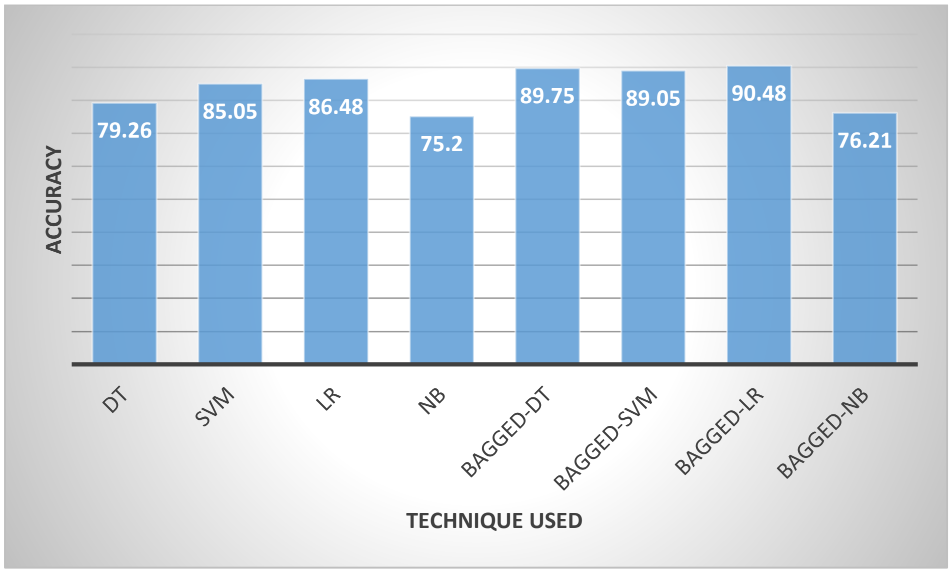


Fig 5.4: Accuracy using Bagging, ANT 1.5

Figure 5.5 shows graphically that ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.6 dataset. We can observe that bagged-DT outperforms and in single base learners the performance of LR is high as compared to DT, SVM and NB.

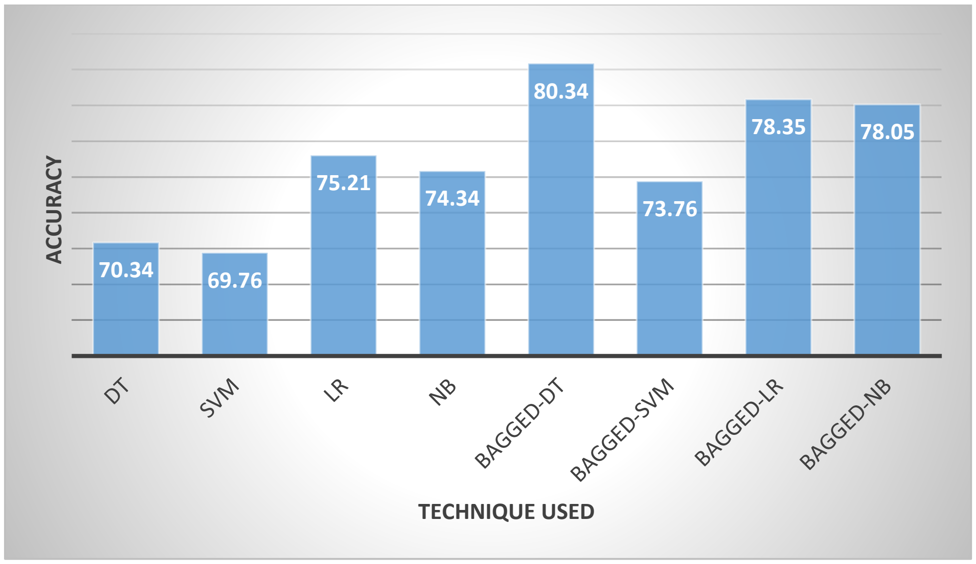


Fig 5.5: Accuracy using Bagging, ANT 1.6

Figure 5.6 shows graphically that ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.7 dataset. We can observe that bagged-LR outperforms and in single base learners the performance of LR is high as compared to DT, SVM and NB.

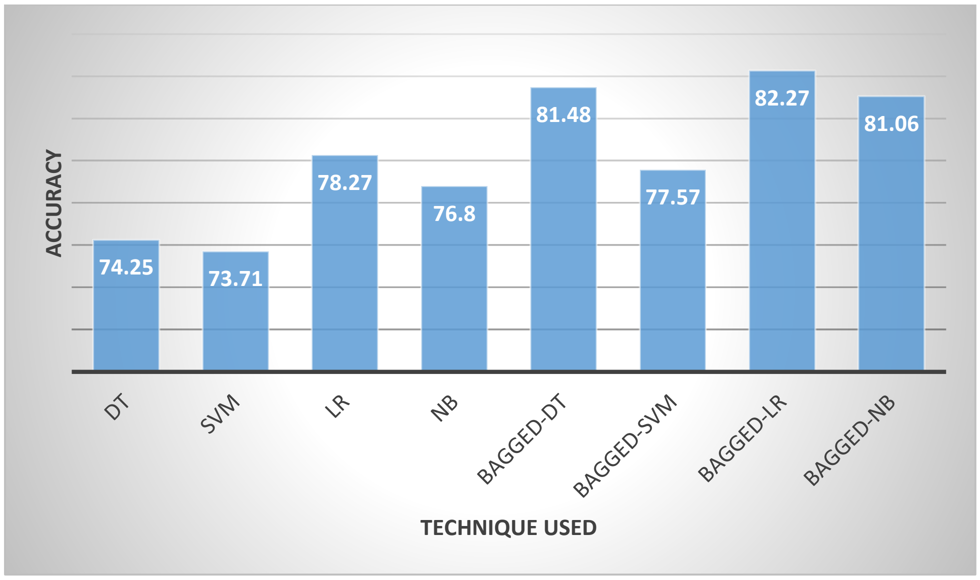


Fig 5.6: Accuracy using Bagging, ANT 1.7

Table 5.3 shows the accuracies of the different versions of dataset over different techniques. The table clearly depicts that there is a hike when we use boosting ensemble. The single classifier models are compared with the bagged version where they have been used as base learners using 10-fold cross validation. For e.g. In Ant 1.5 using single learner DT the accuracy is 79.26 but using boosted-DT accuracy rise up to 83.95.

Table 5.3: Accuracy using 10-fold cross-validation using Boosting

TECHNIQUE USED	VERSIONS OF DATA SET				
	ANT 1.3	ANT 1.4	ANT 1.5	ANT 1.6	ANT 1.7
DT	70.55	64.59	79.26	70.34	74.25
SVM	80.03	73.97	85.05	69.76	73.71
LR	76.7	66.19	86.48	75.21	78.27
NB	73.5	45.52	75.2	74.34	76.8
BOOSTED-DT	80.19	70.75	83.95	74.36	75.3
BOOSTED-SVM	84.03	77.41	89.05	73.79	77.71
BOOSTED-LR	82.37	71.27	89.45	79.18	82.28
BOOSTED-NB	74.23	63.98	83.25	61.81	74.23

Figure 5.7 shows graphically that boosted ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.3 dataset. We have observed that boosted-SVM outperforms and in single base learners the performance of SVM is high as compared to DT, LR and NB.

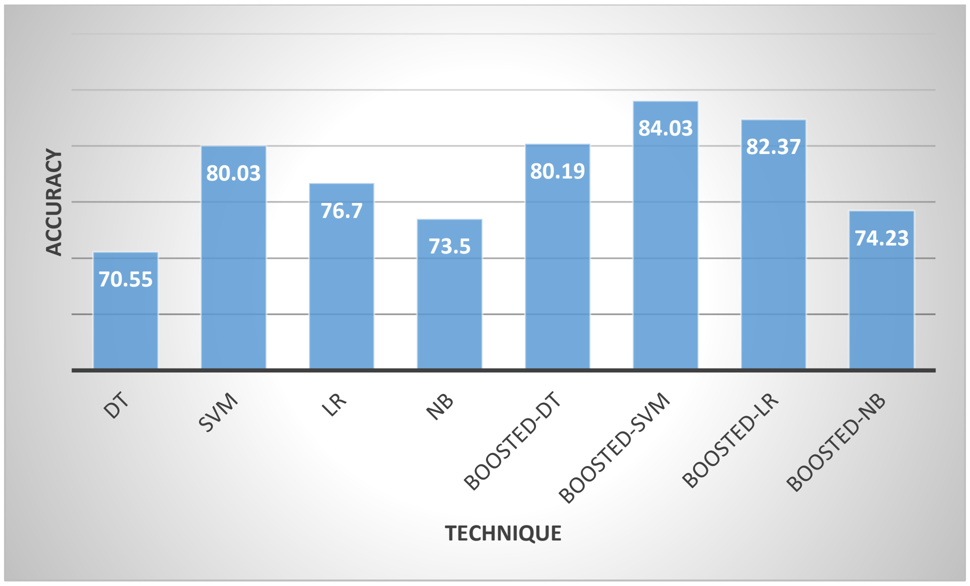


Fig 5.7: Accuracy using Boosting, ANT 1.3

Figure 5.8 shows graphically that boosted ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.4 dataset. We can observe that boosted-SVM outperforms and in single base learners the performance of SVM is high as compared to DT, LR and NB.

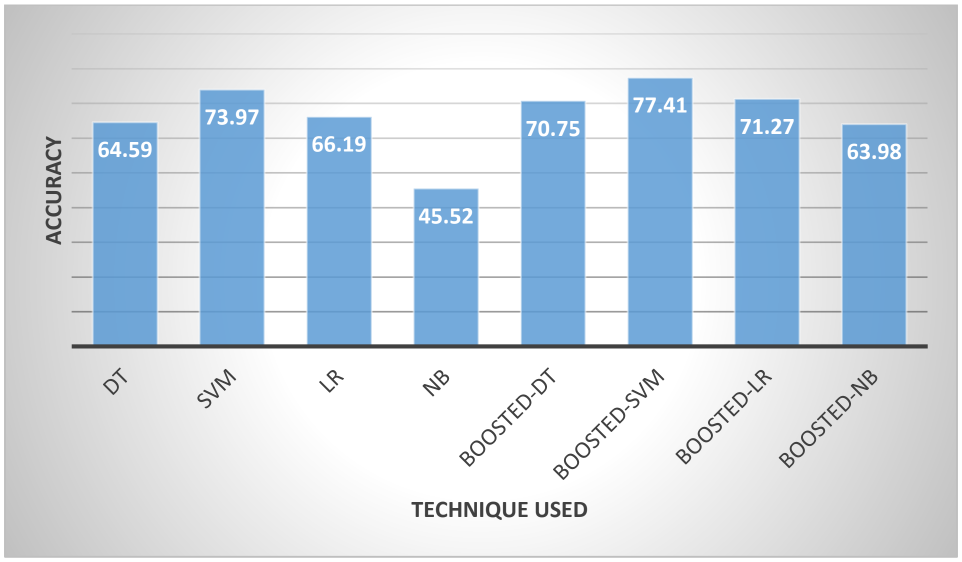


Fig 5.8: Accuracy using Boosting, ANT 1.4

Figure 5.9 shows graphically that boosted ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.5 dataset. We can observe that boosted-LR outperforms and in single base learners the performance of LR is high as compared to DT, SVM and NB.

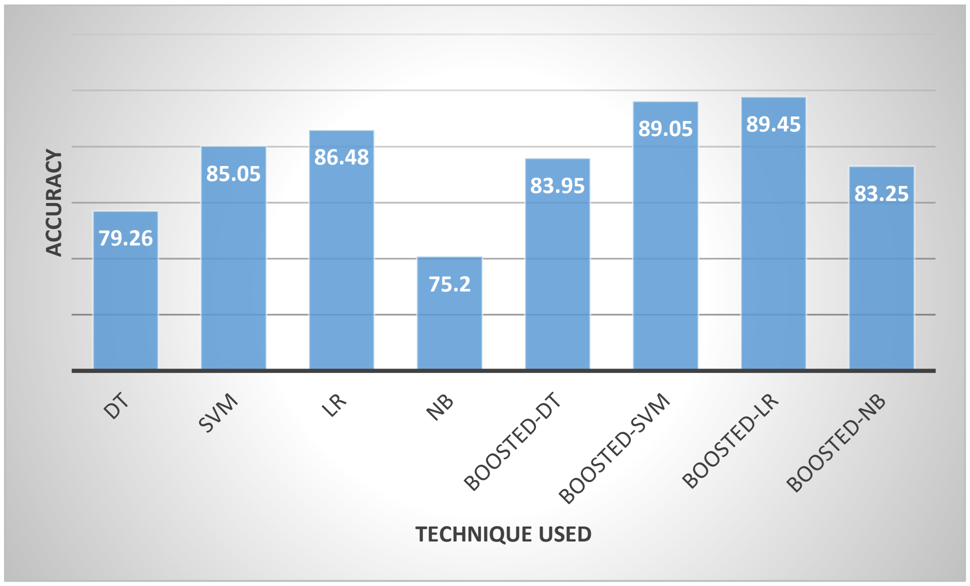


Fig 5.9: Accuracy using Boosting, ANT 1.5

Figure 5.10 shows graphically that boosted ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.6 dataset. We can observe that boosted-LR outperforms and in single base learners the performance of LR is high as compared to DT, LR and NB.

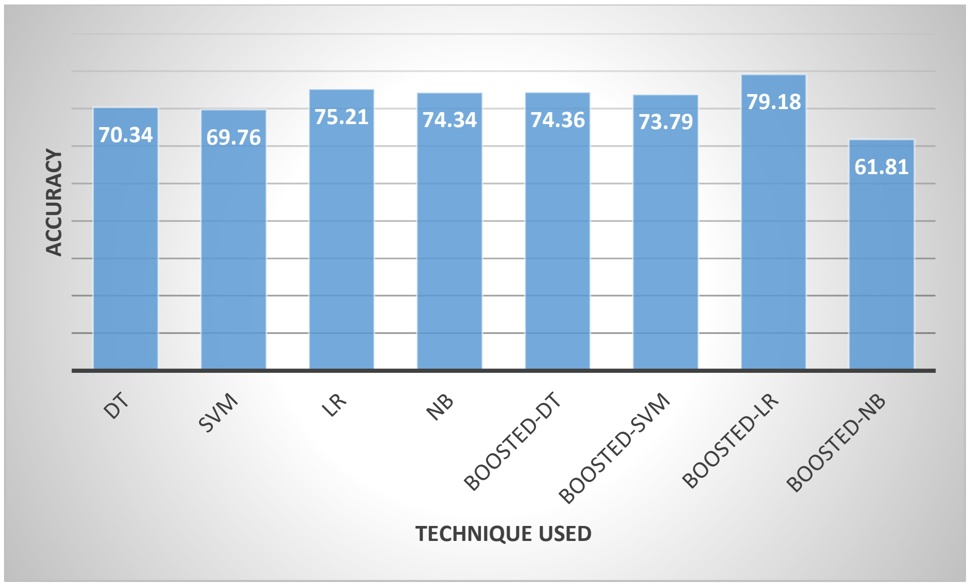


Fig 5.10: Accuracy Using Boosting, ANT 1.6

Figure 5.11 shows graphically that boosted ensemble models of base learners have increased the accuracy as compared to conventional learners on Ant 1.7 dataset. We can observe that boosted-LR outperforms and in single base learners the performance of LR is high as compared to DT, SVM and NB.

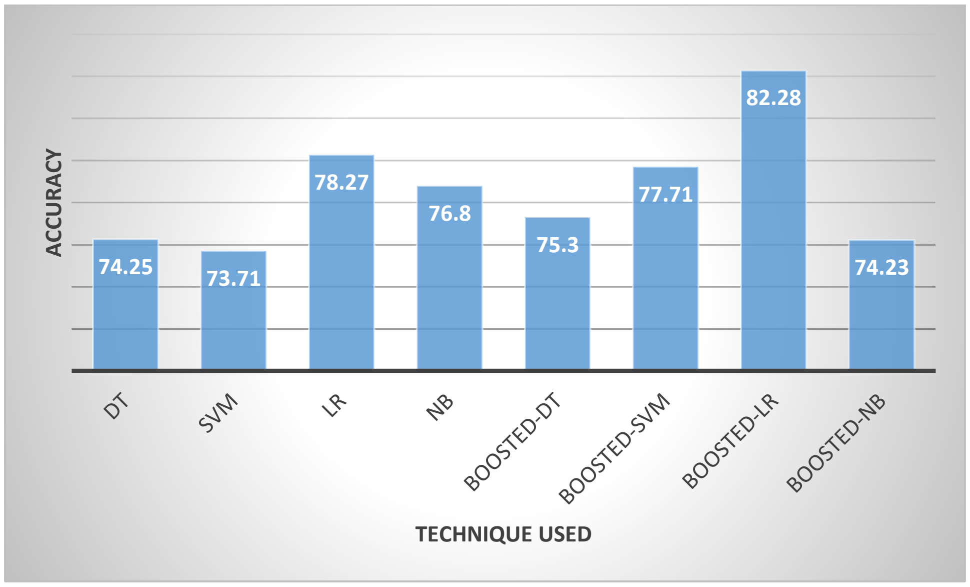


Fig 5.11: Accuracy Using Boosting, ANT 1.7

Table 5.4 shows the accuracies of different versions of dataset over different techniques. The table clearly depicts that there is a hike when we use stacked model ensemble. The single classifier models which are used for comparison are KNN, RF, NB where they have been evaluated using 10-fold cross-validation. For e.g. In Ant 1.5 using single learner KNN, RF and NB the accuracies are 83.96,88.4 and 74.09 respectively but using stacked model accuracy rise up to 89.76.

Table 5.4: Accuracy using Stacking (10-fold cross-validation)

TECHNIQUE USED	VERSIONS OF DATA SET				
	ANT 1.3	ANT 1.4	ANT 1.5	ANT 1.6	ANT 1.7
KNN	79.94	73.63	83.96	74.92	76.64
RF	79.17	74.19	88.4	74.92	76.77
NB	75.91	48.26	74.09	78.07	79.79
STACKED MODEL	83.13	75.81	89.76	81.49	80.03

Figure 5.12 shows graphically that using stacked ensemble model the accuracy has been increased as compared to the conventional learners on all versions of the dataset. We have observed that NB is not able to provide a great accuracy. On the other hand, RF is second best after the stacking model which is best of all.

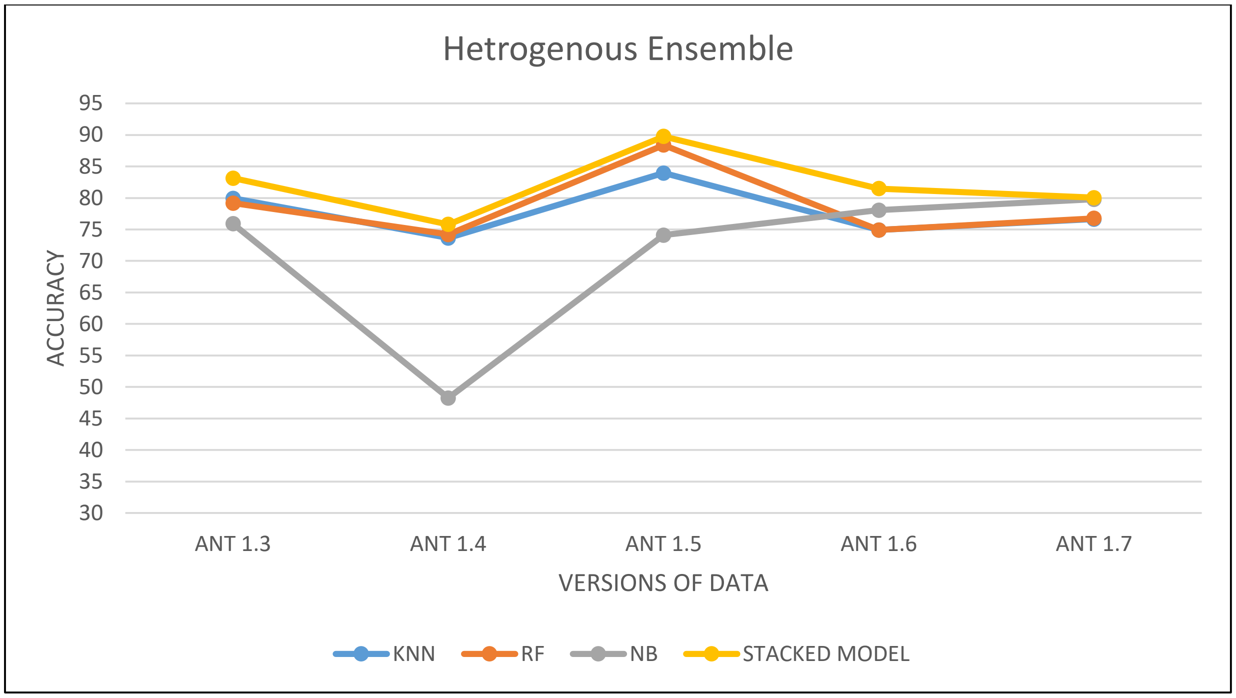


Fig 5.12: Accuracy using Stacking Model

Table 5.5 shows the accuracies of different versions of dataset over different techniques. The table clearly depicts that there is a hike in accuracy when we use voting model ensemble. The single classifier models which are used for comparison are KNN, RF, NB where they have been evaluated using 10-fold cross-validation. For e.g. In Ant 1.5 using single learner KNN, RF and NB the accuracies are 86.48, 79.26 and 85.05 respectively but using voting model accuracy rise up to 89.14.

Table 5.5: Accuracy using Voting (10-fold cross-validation)

TECHNIQUE USED	VERSIONS OF DATA SET				
	ANT 1.3	ANT 1.4	ANT 1.5	ANT 1.6	ANT 1.7
KNN	76.7	66.19	86.48	75.21	78.27
RF	70.55	64.59	79.26	70.34	74.25
NB	80.03	73.97	85.05	69.76	73.71
VOTING MODEL	84.1	76.43	89.14	79.26	81.74

Figure 5.13 shows graphically that stacked ensemble model has increased the accuracy as compared to conventional learners on all versions of the dataset. We can observe voting ensemble is outperforming when compared with rest of the techniques.

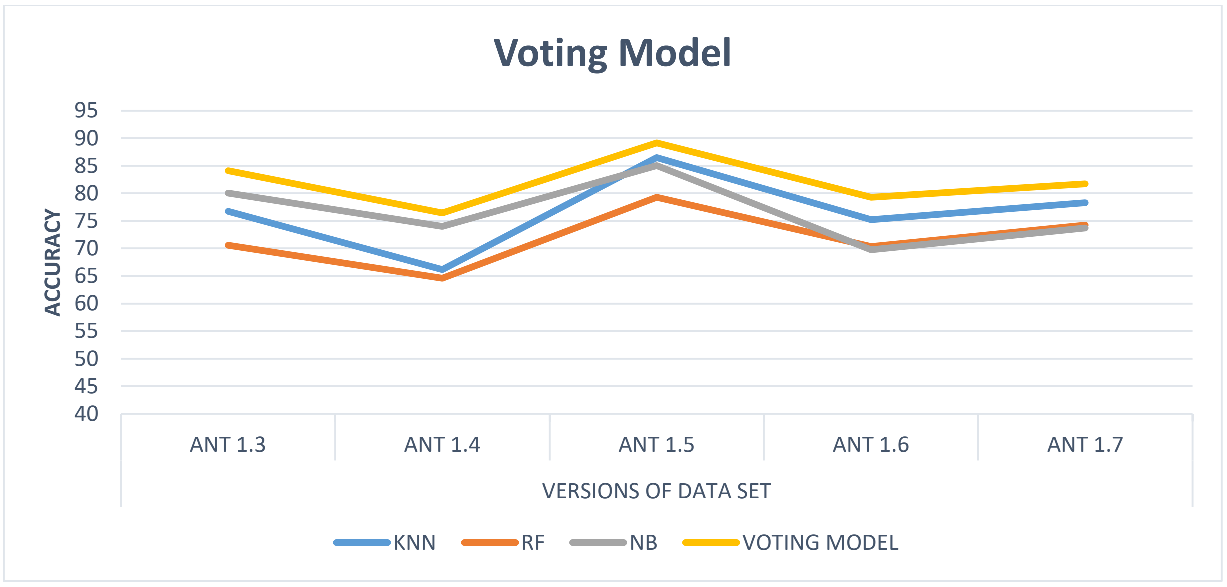


Fig 5.13: Accuracy using Voting Model

Another measure that have been used is AUC score, which explains the capability of classifier model that how it is good for classifying between classes. The higher the AUC, the better the model. By analogy, it is used to differentiate between defective point and no defective points.

Table 5.6 shows the AUC score on Ant 1.3 version of dataset over different techniques. The table clearly depicts that there is an increase in score when we use bagging/boosting ensemble. The single classifier models are compared with the ensemble models where the comparison is between the respective base learners. For e.g. in DT bagging and boosting have increased to .83 and .65 respectively.

Table 5.6: AUC score on Ant 1.3 for homogenous models

TECHNIQUE USED	ANT 1.3			
	DT	SVM	LR	NB
BOOSTING	0.65	0.82	0.7	0.82
BAGGING	0.83	0.74	0.67	0.82
INDIVIDUAL MODEL	0.62	0.7	0.67	0.73

Figure 5.14 shows graphically that boosted ensemble models of different base learners have increased the accuracy as compared to conventional learners on Ant 1.3 dataset. We have observed that boosted-SVM outperforms also in single base learners the performance of SVM is high as compared to DT, LR and NB.

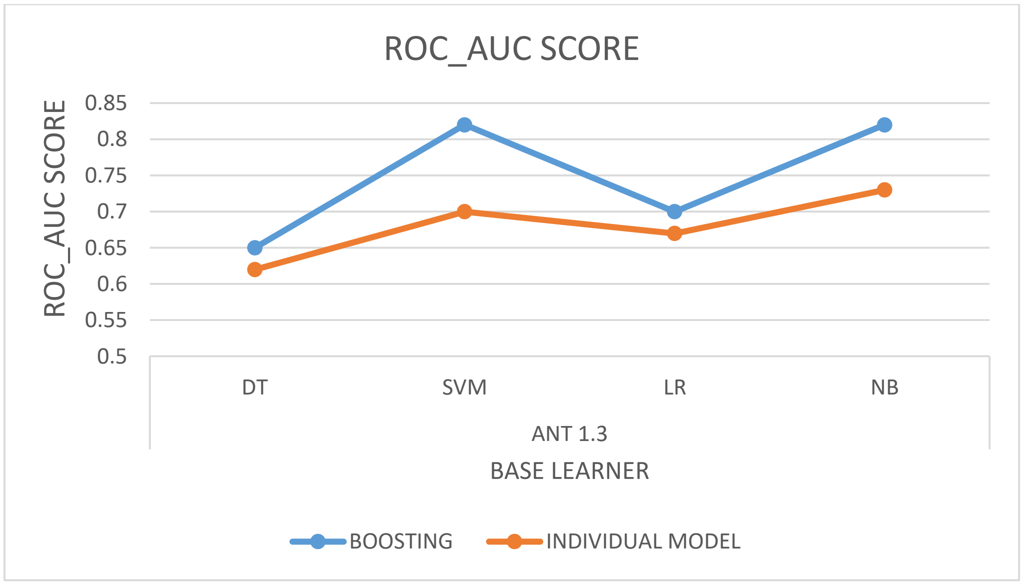


Fig 5.14: AUC score of Boosting v/s individual model for ant 1.3

Figure 5.15 shows graphically that bagged ensemble models of different base learners have increased the accuracy as compared to conventional learners on Ant 1.3 dataset except the LR where the AUC score is same. We have observed that boosted-DT outperforms and in single base learners the performance of NB is high as compared to DT, LR and SVM.

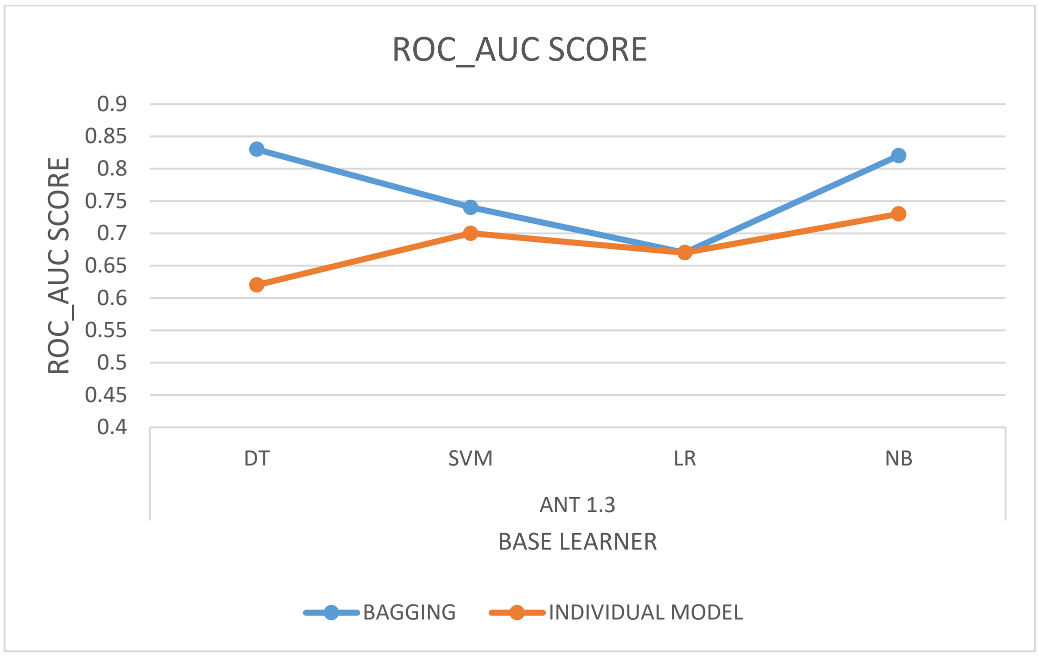


Fig 5.15: AUC score of Bagging v/s individual model for ant 1.3

Table 5.7 shows the AUC score on the Ant 1.6 version of dataset over different techniques. The table clearly depicts that there is increase in score when we use bagging/boosting ensemble. The single classifier models are compared with the ensemble models where the comparison is in between the respective base learners. For e.g. in DT bagging and boosting have increased to .88 and .72 respectively.

Table 5.7: AUC score on Ant 1.6 for homogenous models

TECHNIQUE USED	ANT 1.6			
	DT	SVM	LR	NB
BOOSTING	0.72	0.82	0.87	0.89
BAGGING	0.88	0.83	0.86	0.85
INDIVIDUAL MODEL	0.67	0.79	0.84	0.8

Figure 5.16 shows graphically that boosted ensemble models of different base learners have increased the accuracy as compared to conventional learners on Ant 1.6 dataset. We can observe that boosted-NB outperforms and in single base learners the performance of LR is high as compared to DT, SVM and NB.

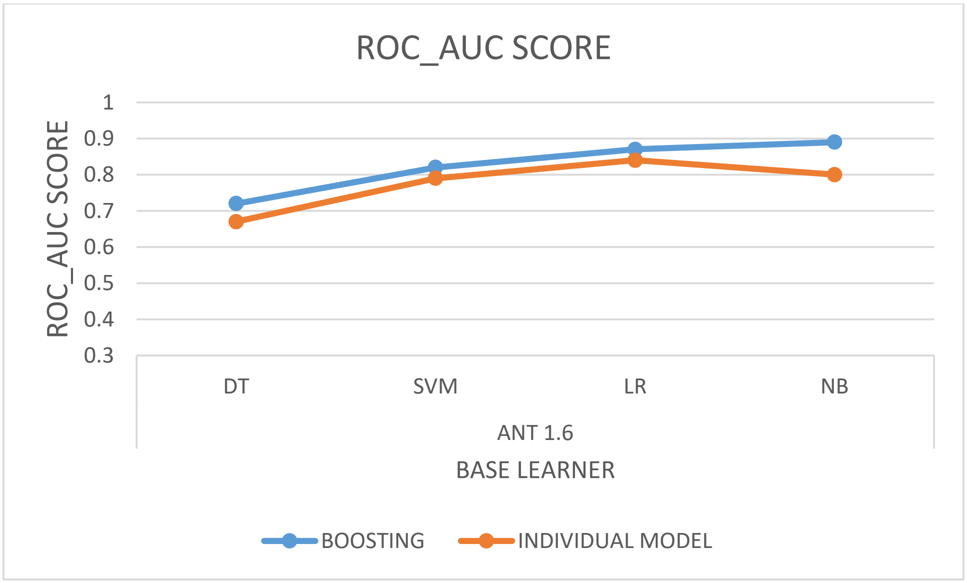


Fig 5.16: AUC score of Boosting v/s individual model for ant 1.6

Figure 5.17 shows graphically that bagging ensemble models of different base learners have increased the accuracy as compared to conventional learners on Ant 1.6 dataset. We can observe that boosted-DT outperforms also in single base learners the performance of LR is high as compared to DT, SVM and NB.

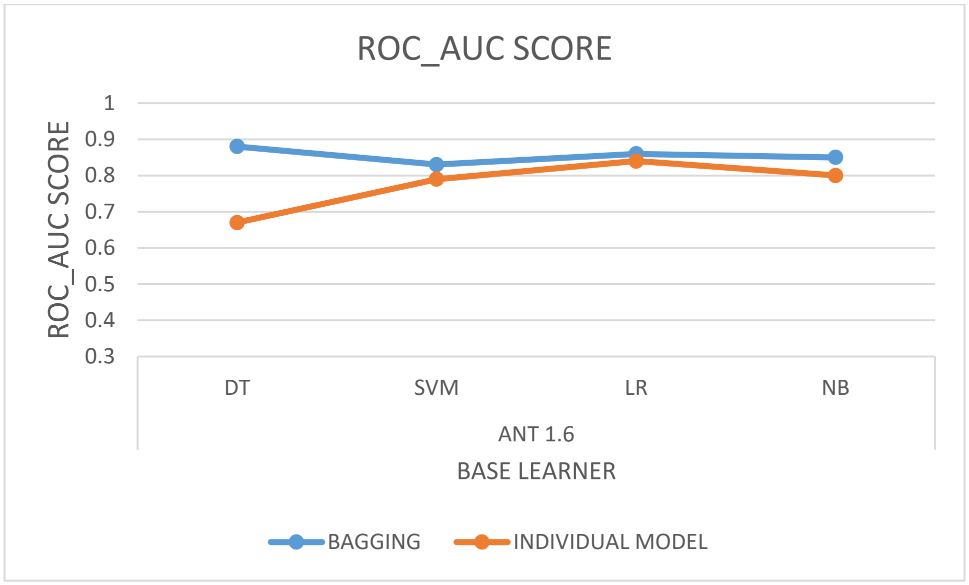


Fig 5.17: AUC score of Bagging v/s individual model for ant 1.6

On comparing the ensemble methods against each other we can segregate them all in two categories as follows:

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

Figure 5.18 shows graphically the averaged accuracies of bagging and bosting over different versions of ant dataset.

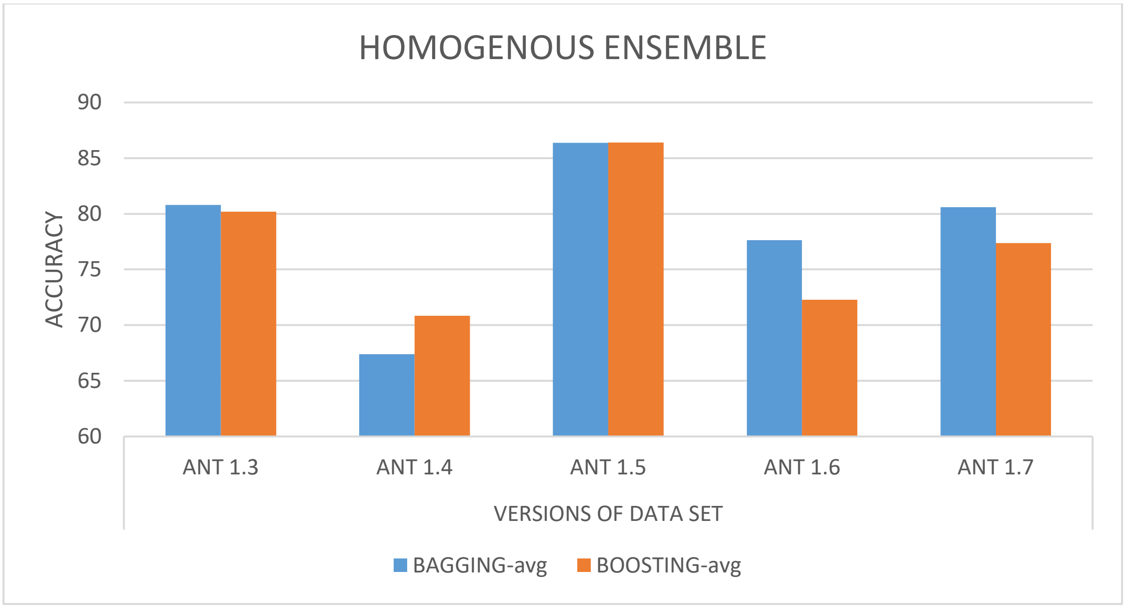


Fig 5.18: Averaged accuracies of Bagging v/s Boosting

Figure 5.19 depicts graphically the accuracy of the heterogeneous ensembles on the different versions of the ant dataset.

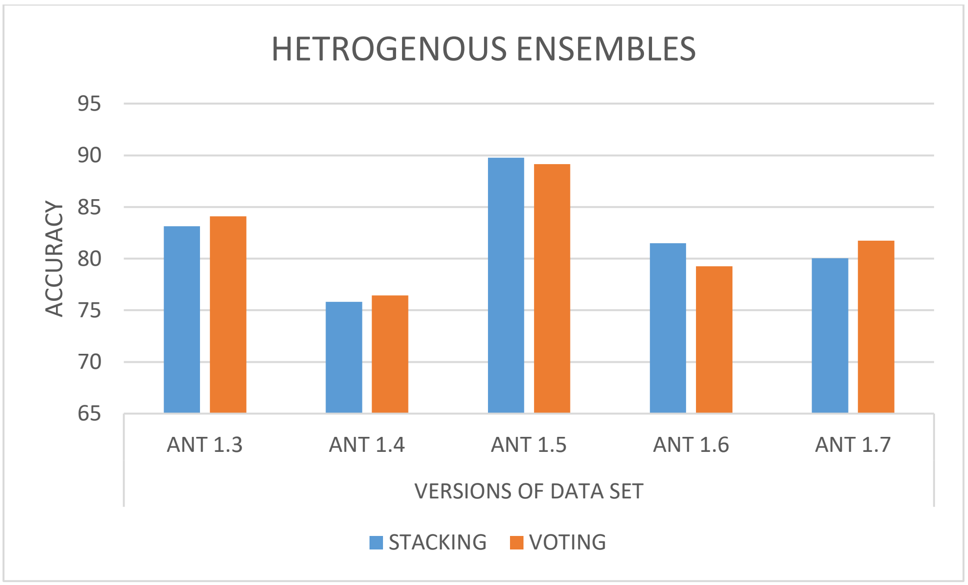


Fig 5.19: Averaged accuracies of Stacking v/s Voting

For statistically analyzing the results Friedman test has been used. Friedman test is a distribution-free test used to compare multiple treatments on the same subjects. Friedman test in this study finds out if there are any statistically significant differences between the accuracy of these techniques when predicting defectiveness of software.

Table 5.8 depicts the ranks that were obtained using the Friedman test. This table shows the results for bagging. Here the rank of bagged-LR is highest i.e., bagged-LR is best suited technique for our problem out of all other bagging techniques.

Table 5.8: Results of Friedman Test-Bagging

TECHNIQUE	**MEAN_RANK**
BAGGED-DT	3.00
BAGGED-SVM	2.40
BAGGED-LR	3.20
BAGGED-NB	1.40

Table 5.9 depicts the ranks that were obtained using the Friedman test. This table shows the results for boosting. Here the rank of boosted-LR is highest i.e., boosted-LR is the best suited technique for our problem out of all other boosting techniques.

Table 5.9: Results of Friedman Test-Boosting

TECHNIQUE	**MEAN_RANK**
BOOSTED-DT	2.20
BOOSTED -SVM	3.20
BOOSTED -LR	3.60
BOOSTED -NB	1.00

CHAPTER-6

CONCLUSION AND FUTURE WORK

This study empirically analyzes the Accuracy and the AUC score which are the OO performance metrics of two homogeneous ensemble learners boosting and bagging with variations in their base learners. This study to 4 base learners in w.r.t open source Java projects on ANT versions in relevance to SDP. Our significant findings are bagging and boosting are having increased accuracy as compared to individual learners. On considering bagging only bagged-SVM outperformed on 3 datasets out of five datasets whereas bagged-DT and bagged-LR outperformed on ANT 1.5 and 1.6 respectively.

On considering boosting only boosted-SVM outperformed on 2 datasets out of five datasets whereas boosted-LR outperformed on ANT 1.5, ANT 1.6 and 1.7.

Our heterogeneous ensembles stacking and voting outperformed when compared with the base learners that were used in them.

With the help of ensemble learners bagging, bosting, voting and stacking we find increase in both accuracy as well as AUC score. Thus they helped to gain the performance increase in base learners. Using Friedman Test, we statistically analyzed that bagging and boosting performed better with Logistic Regression as its base learner.

Future work may involve exploration of search-based techniques for the prediction of defects and their capabilities can be used to increase the performance of the model for SDP. In addition to this higher severity level defects can also be predicted so that resource allocation can be managed efficiently.

REFERENCES

[1] T. Gyimothy, R. Ferenc and I. Siket, “Empirical validation of object-oriented metrics on open source software for fault prediction,” *IEEE Transactions 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. 1034-1050.

[4] R.Malhotra, S. Shukla.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 (ICRITO) (Trends and Future Directions)*, Sep 7-9, 2016, AllT, Amity University Uttar Pradesh , Noida,India.

[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] C. Toon and S. Verwer, “Three naive Bayes approaches for discrimination-free classification,” *Data Mining and Knowledge Discovery*, Vol. 21, No. 2, 2010, pp. 277-292.

[8] A. Kaur and R. Malhotra, “Application of random forest in predicting fault-prone classes,” *ICACTE*, vol.8, 2008, pp. 37-43.

[9] V.R Basili, L.C Briand, and W.L Melo. “A validation of object-oriented design metrics as quality indicators,” *Software Engineering, IEEE TRANSACTIONS*, vol.22, no.10 pp.751–761, 1996.

[10] L.C Briand, , J.W Daly, and V. Porter, “Exploring the relationships between design measures and software quality in object- oriented systems,” *Journal of Systems and software*, vol. 51, no. 3, pp. 245–273, 2000.

[11] P. Singh and A. Chug, “Software defect prediction analysis using machine learning algorithms,” *In International Conference of Computing and Data Science*, 2017, pp. 775-781.

[12] Z .Sun, Q. Song, X. Zhu, “Using coding-based ensemble learning to improve software defect prediction,” *IEEE Transactions on Systems*, vol.43, no.6, 2012, pp. 313-325.

[13] T. Wang, W. Li, H. Shi and Z. Liu, “Software defect prediction based on classifiers ensemble,” *Journal of Information Systems*, vol.8, no.16, 2011, pp.4241-4254.

[14] A. Kaur and K. Kamaldeep, “Performance analysis of ensemble learning for predicting defects in open source software,” *International Conference on Advances in Communication and Informatics*, 2014, pp. 219-225.

[15] P. Singh and A. Chug, “Software defect prediction analysis using machine learning algorithms,” *In International Conference of Computing and Data Science*, 2017, pp. 775-781.

[16] H. Shamsul, L. Kevin and M. Abdelrazek, “An ensemble oversampling model for class imbalance problem in software defect prediction,” *IEEE Access 6*, 2018.

[17] A. Abdel Aziz, N. Ramadan and H. Hefny, “Towards a Machine Learning Model for Predicting Failure of Agile Software Projects,” *International Journal of Computer Application*, vol.168, no.6, 2017.

[18] S. Wang and X. Yao, “Using class imbalance learning for software defect prediction,” *International Conference of Computing and Data Science* vol. 62, no. 2, 2013, pp. 434-443.

[19] H. Yuan, C. Van Wiele and S. Khorram. “An automated artificial neural network system for land use/land cover classification from Landsat TM imagery,” *MDPI*, Vol.1, No.3, 2009, pp. 243-265.

[20] T. Kavzoglu, and I. Colkesen, “A kernel functions analysis for support vector machines for land cover classification,” *International Conference on Advances in Communication and Informatics*, vol.11, no.5, 2009, pp. 352-359.

[21] A. Kaur, K. Kaur and D. Chopra “An empirical study of software entropy based bug prediction using machine learning,” *International Conference on Distributed systems*, vol.8, no.2, 2017, pp. 599-616.

[22] J. Chen, H. Huang, S. Tian, and Y. Qu. “Feature selection for text classification with Naïve Bayes,” Expert Systems with Applications, vol. 6, no. 3, 2009, pp. 5432-5435.

[23] C. Toon and S. Verwer, “Three naive Bayes approaches for discrimination-free classification,”*Data Mining and Knowledge Discovery*, vol. 21, no. 2, 2010, pp. 277-292.

[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, pp. 504-518, 2015.

[26] I. Myrtveit, E. Stensrud and M. Shepperd, “Reliability and validity in comparative studies of software prediction models,” *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 380–391, 2005

[27] W. Dai, Y.E. Shao, C.J.J Lu, “Incorporating feature selection method into support vector regression for stock index forecasting,” *Neural Computing and Applications*, vol. 23, no. 6, pp. 1551-561, 2012.

[28] W. Chen and C.J. Lin, “Combining SVMs with various feature selection strategies,” *IEEE Transactions on Software Engineering*,2005.

[29] C.L. Huang, M.C. Chen and C.J. Wang, “Credit scoring with a data mining approach based on support vector machines,” *Expert Systems with Applications*, vol. 33, no. 4, pp. 847-856, 2007.

[30] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[31] J. Platt, “Fast training of support vector machines using sequential minimal optimization,” In *Advances in Kernel Methods—Support Vector Learning, Cambridge, MA: MIT Press*, 1999, pp. 185–208.

[32] S.K. Shevade, S.S Keerthi, C.K. Bhattacharyya and R.K. Murthy, “Improvements to the SMO Algorithm for SVM Regression,” *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1188-1193, 2000.