ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 8, Issue 7, July 2021

An Ensemble Classification Model For Software Defect Prediction

[1] Baldeep Kaur, [2] Dr. Sumeet Kaur Sehra, [3] Dr. Daljeet Singh
[1] Student, Guru Nanak Dev Engineering College, Ludhiana
[2][3] Assistant Professor, Guru Nanak Dev Engineering College, Ludhiana

Abstract: Software developed for a specific requirement is called software product. Engineering at the same time, is related to the product development by means of explicit technical fundamentals and techniques. The software defect prediction have various phases which are include data set input, pre-processing, feature extraction and classification. The various classification schemes are applied for the software defect prediction in this research work. The classification schemes like Gaussian Naive Bayes, Bernoulli Naive Bayes, Random Forest and Decision Tree are used for the software defect prediction. To improve performance for the software defect the ensemble classification method is designed in this research work. The proposed ensemble classification method is the combination of PCA algorithm with class balancing. The proposed model is implemented in python and results are analyzed in terms of Accuracy, Precision and Recall.

Keywords: Software Defect, Gaussian Naive Bayes, Bernoulli Naive Bayes, Random Forest, C4.5, PCA, Class Balancing

1. INTRODUCTION

Software is referred not just a program code but the much above. A program signifies an executable code for some computing applications. Software can be defined as a compilation of executable programming code, related libraries and documentations. Software developed for a specific requirement is called software product. Software defect along with an intrinsic element of software product, is also an important aspect of software quality. Software defects are an unavoidable co-product of the developed software [1]. In addition to this, the guarantee of software quality assurance is not so easy and requires a lot of time too. There are different ways to define defects, such as in terms of quality. However, the defects are generally defined in the form of deviations from specifications or expectations which may be the reason of failure in functioning. In general, various software projects do not have sufficient time and workforce available for eliminating all the defects prior to the release of a specific product. This may affect the overall product quality and probably the status of an organization that deliver the product. In this condition, the potential level of several techniques with the ability of providing alternate methods for assuring the quality of software product becomes vast. The defects may be present in all products whether it is a small program or large-scale software system. A number of defects can be discovered without any complexity. But some defects hidden deeply cannot be found easily. Some defects do not harm much but some could make huge loss of property or even cause threat to life. Plenty of time and labour of developers, clients and maintenance staff, is required from the primary designing process to the ultimate software usage. The efficient prediction of software defects is a must for the assurance of software quality. Software defect prediction techniques may divert attention of quality assurance activities towards the code that is most vulnerable to defect. These techniques can provide more resources to resolve the complex issues. DeP (Defect Prediction in Software) refers to the process of determining segments of a software system that may include defects. In software engineering, prediction of software defect (Bug) is a very promising research domain. The teams of quality assurance are able to efficiently distribute available resources for testing and examining software products using the list of defect-prone software objects provided by defect prediction models. In the software lifecycle, the early use of defect prediction models enables experts to concentrate their testing team in such a way that the testing of the parts recognized as “prone to defects” can be performed with more accuracy as compared to the other portions of the software product. This provision may reduce the workforce costs throughout development and also give some relaxation to efforts made in maintenance. The construction of defect prediction models depends on two approaches [2]. The first approach makes use of measurable features of the software system known as Software Metrics, for constructing a De-model. The second approach on the other hand uses fault data from an alike software project to do so. When the defect prediction model is built, it can be implemented to future software projects. In this way, experts can detect bug-prone sections of a software system. There are mainly two categories of Software defect prediction techniques. The techniques in the first category are used for predicting the various defects in the product module/class while the techniques in the second category are used for classifying the module/class in terms of defected or non-defected.

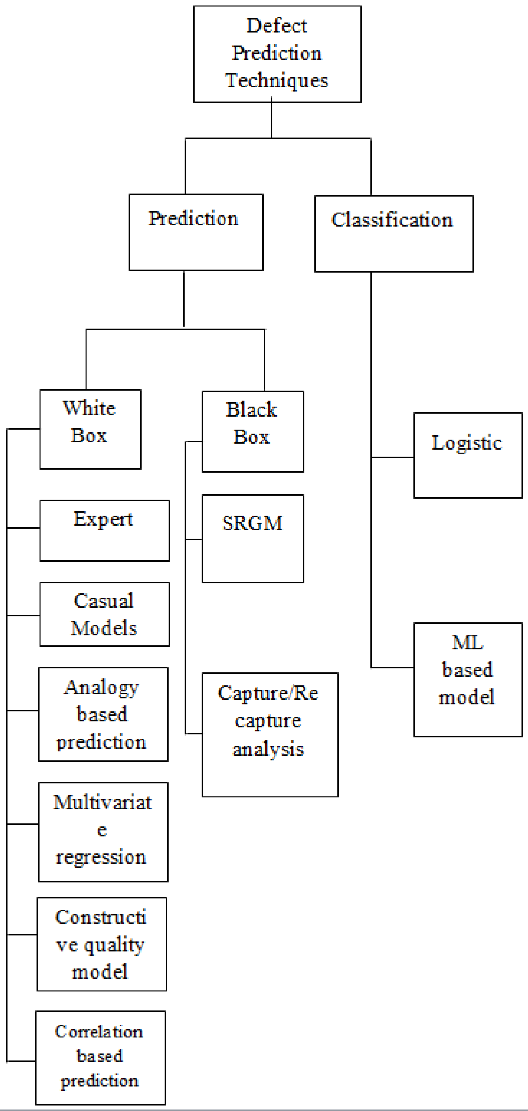


Figure 1.1: Software Defect Prediction Techniques

Machine learning models make use of data mining methods and statistical techniques-based algorithms for classifying defects. The models provide the classification outcomes of software elements as faulty or non-faulty by using some prediction variables as input information. The use of machine learning algorithms is quite popular among researchers for the classification of software parts. Figure 1.1 shows a general process of software defect prediction based on machine learning.

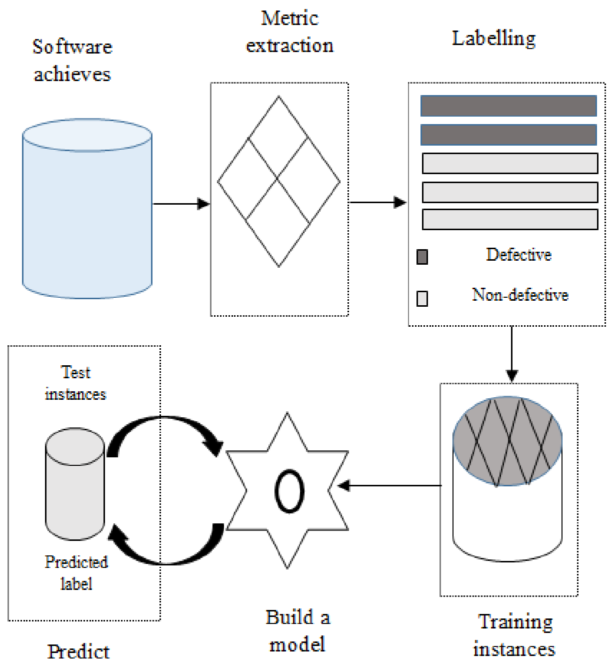


Figure 1.2: General Process of Software Defect Prediction

As shown in the figure, the first step towards the construction of a software defect prediction model is to generate data patterns from software databases that include issue tracking systems, version control systems etc. The source codes and some commit messages occur in the version control systems. The issue tracking systems on the other hand contain some defect info. The prediction granularity states that every pattern has the ability to represent a technique, a class, a source code file, a package or a change in code. A number of defect prediction attributes taken out from the software repositories generally occur in the pattern. The intricacy of software and its development process is represented by the value of the metrics. The labelling of a pattern can be labelled as defected or non-defected on the basis whether the pattern includes defects or not. Afterward, a group of training patterns can be used for constructing the defect prediction models on the basis of achieved metrics and labels [3]. At last, the prediction model can do the classification of a new patter into defective or non-defective.

II. LITERATURE REVIEW

Guo et al, 2018 emphasized on transmitting strong defect capability of classification for predicting effort-aware software defect effectively [4]. The connection between the classification performance and the cost-effectiveness curve was studied in experimental manner in which 6 open-source software data sets had carried out. The results indicated that the skewed distributions of change size were monitored enormously due to which the lack of association had occurred between performance while classifying and the potential for finding effective test orderings while detection defects. All the effort-aware approaches which connected high classification capacity for providing effort-aware performance effectively had enabled through trimming. At last, it was analyzed that the effort-aware models were governed the Effort distributions. The Trimming was proved the best practical technique for handling this issue.

Shan et al, 2014 analyzed that software security was enhanced when the testers assisted in locating the software defects in accurate way through predicting the software defect [5]. In defects dataset, the accuracy in predicting the defects was reduced by data redundancy that occurred due to the overmuch attributes. This paper suggested a LLE-SVM based model for dealing with the issue. The implementation of support vector machine was done as basic classifier in this model. The data redundancy was addressed using locally linear embedding algorithm that had potential to maintain local geometry. The technique included ten-fold cross validation and grid search optimized the parameters in support vector machine. The NASA defect data set had employed to validate the comparison of locally linear embedding and support vector machine with SVM model experimentally. It was demonstrated in the outcomes that the suggested model outperformed the SVM model and it had capability for avoiding the accuracy decrease that occurred due to the data redundancy.

Punitha et al, 2013 Assisting developers for recognizing defects of software on the basis of existing software metrics in which data mining schemes were implemented. In this way, the software quality was enhanced that ultimately initiated the reduction of the software development cost in the development and maintenance phase [6]. The defective modules had recognized that was emphasized in this research. Consequently, the examination of the scope of software was required to prioritize the defects. The test cases were carried out for implementing the test cases in the predicted modules by the developers. This suggested technique assisted for the recognition of modules in which immediate attention was necessitated and enhancing the reliability of the software faster in which the defects with higher priority were handled initially. The major purpose of this technique was that the classification accuracy of the Data mining algorithm was enhanced. First of all, the existing classification algorithms were suggested for the evaluation and initialization of this process. The Neural network algorithm was also suggested on the basis of its vulnerability with a degree of fuzziness in the hidden layer for improving the classification accuracy.

EboBennin et al, 2018 suggested a new and effectual synthetic oversampling approach named MAHAKIL used in software defect datasets. This approach was designed on the basis of the chromosomal theory of inheritance [7]. With this theory, the interpretation of two different sub-classes was performed as parents and a new instance was carried out in which various traits were inherited from each parent and contributed to the diversity in the data distribution. The MAHAKIL was compared with five other sampling approaches for which twenty releases of defect datasets were carried out from the PROMISE repository and 5 prediction models. It was demonstrated in the experiments that the prediction performance was enhanced for all the models and superior and more considerable of values were obtained using MAHAKIL as compared to other oversampling approaches on the basis of robust statistical tests .

Wu et al, 2017 emphasized on providing an effectual solution for dealing with CSDP and WSDP issues while predicting software defect [8]. The SSDL that was an effectual ML method was presented in predicting the defect and a SSDL approach was suggested for approach for CSDP and WSDP. The useful information was carried out in limited labelled defect data and a large amount of unlabelled data in the semi-supervised structured dictionary learning approach. Two public datasets were employed to conduct the experiments. The results demonstrated that a superior performance for predicting defects was obtained from the semi-supervised structured dictionary learning as compared to related SSDP techniques in the CSDP scenario.

Mausa et al, 2014 proposed a tool demonstration in which a systematic data collection process was executed for software defect prediction datasets that were carried out from the open source bug tracking and the source code management repositories [9]. This tool dealt with the major challenging issue that was to link the information of the same entity from these two sources. The interfaces were employed to bug and source code repositories using tool and the other tools were utilized to compute the software metrics. At last, the user was allowed for developing datasets in predicting the software defect although the user had not aware about the all the details behind this complex task.

Al-Jamimi et al, 2016 discussed that the Software defect prediction was one of the disciplines that assisted in predicting the defects proneness of future modules [10]. Software metrics were implemented in this type of predication. But there was an association between the predication metrics and uncertainty. Therefore, it was essential to express these metrics in linguistic terms for dealing with ambiguity and uncertainty. The software metrics and opinions of experts were two categories of knowledge that had carried out as input in prediction models. A framework was suggested to generate a fuzzy logic-based software predication model in which various set of software metrics had utilized. A generic set of metrics was offered in this paper that was utilized for predicting the software defects. The real software projects data that utilized a Takagi-Sugeno fuzzy inference engine in the prediction of software defects had implemented for the authentication of performance of suggested Fuzzy-based models and the promising validation outcomes had obtained.

Jing Sun et al, 2018 suggested a technique named GFKSDP for dealing with the issue of various distributions that located between source domain and target domain [11]. The differences of source and target domains were minimized when an infinite number of subspaces was combined that was used for characterizing the changes of geometric and statistical properties and carried out from source domain to target domain using geodesic flow kernel software defect prediction technique. The important parameters were determined in adaptive manner in this technique for reducing the computational complexity. The suggested technique outperformed the conventional studies in unsupervised learning. The AEEEM and Relink datasets were employed to conduct the experiments. The results proved that the performance of cross-project prediction was enhanced using suggested technique. This technique performed better than the techniques in unsupervised learning.

Ayon et al, 2019 recommended a technique in which GA was implemented for the selection of attributes [12]. Then, PSO was carried out for generating a cluster of selected attributes. Various NN techniques named FNN, RNN and ANN were employed for training the model. At last, precision, sensitivity, precision, NPV, F1 score, and MCC were computed. There were 5 various datasets that had carried out from NASA a promise software engineering repository in this study. The greatest accuracy results were acquired through DNN. The results of experiments represented that recommended method was perfect for predicting the defects of software.

Xia et al, 2014 emphasized on realizing the significance of various process metrics while predicting the defects for TT&C software and a series of experiments had performed for which many combinations sets of software metrics were employed [13]. The results of experiments demonstrated that the performance to forecast the defect was enhanced through the combination set of CM + MMSLC + HM and the prediction performance for TT&C software was enhanced under the influence of the history change process metrics. For the future work, the importance of each process metric was analyzed in detail.

III. RESEARCH METHODOLOGY

The proposed methodology is based on the various classifies like random forest, Gaussian Naïve Bayes, Bernoulli Naïve Bayes, Decision Tree. The ensemble classifier is generated for the software defect prediction. The ensemble classifier is the combination of Gaussian naive Bayes, Bernoulli Naive Bayes, Random Forest and C4.5. In the third phase of the research work, the feature extraction technique called PCA is applied with the ensemble classifier with class balancing. The details of each classifier are explained below:-

A. Decision Tree

DTs are trees in which instances are classified after sorting on the basis of feature values. A feature is illustrated in an instance for the classification using every node in a DT and a value which can be assumed by the node is signified using each branch. Initially, the classification of instances is done at the root node and sorted on the basis of their values of attribute. The DTs namely C5.0, Id3, or CART are utilized to handle the real-world datasets in effective manner. The C5.0 is a DT which was developed ID3 depending on the information gain due to the partiality of the ID3 of multi-valued features. C5.0 was developed for dealing with that problem with the computation of the information gain ration obtained for each feature. Afterward, the attribute consists of the maximal Information Gain ration value is chosen as a root node of the training dataset. The attribute that has the maximum gain ratio is selected for splitting so as the needed information is alleviated while predicting a given instance in the resulting separation of a feature. The evaluation of Gain Ration for attribute A is presented as:

$$GainRatio(A) = \frac{Gain(A)}{Split\ Info(A)}$$

$$Gain(A) = info(D) - Info_A(D)$$

Where D is the training dataset

$$Info(D) = -\sum_{i=1}^{n} p(ci)\log_2 P(C_i)$$

$P(C_i) = |C_{i,D}|/|D|$ where $|C_{i,D}|$is the number of the tuples included in the class C_i used in the training dataset and |D| is the number of the tuples of the training dataset, and n represents the number of the values of the class.

$$Info_A(D) = \sum_{i=1}^{n}\left(\frac{|a_{i,D}|}{|D|} \times \left(-\sum_{j=1}^{m}\frac{|C_{JD}|}{|a_{i,D}|}\log_2\frac{|C_{J,D}|}{a_{i,D}}\right)\right)$$

Where $|a_{i,D}|$is the number of the tuples of the value a_J of the attribute A in the training dataset and $|D|$denotes the tuples of the training dataset and n is the number of the values of attribute A. $|C_{J,D}|$corresponds to the number of the tuples of class $|C_J|$associated with the value ai of the attribute A and m, the number of the classes of class C.

B. Bernoulli Naive Bayes

The NB training and classification algorithms are carried out in the Bernoulli Naive Bayes for data whose distribution is done in accordance with the multivariate Bernoulli distributions. This implies that there are numerous attributes. Moreover, every attribute is deduced as a binary-valued variable. Consequently, the samples are necessitated for the class that are represented as binary-valued feature vectors. When any other kind of data is handed, a Bernoulli Naive Bayes instance are binarised its input. The decision rule for Bernoulli naive Bayes is depending on

$$P(x_i|y) = P(i|y)x_i + (1 - P(i|y))\,(1 - x_i)$$

that differs from the multinomial Naive Baye's rule that penalized the non-occurrence of a attribute $i$that is an indicator for class y in which the multinomial variant would simply ignore a non-occurring attribute. This classifier is utilized and trained using word occurrence vectors in the text classification. The Bernoulli Naive Bayes has provided the superior performance on some datasets, especially those with shorter documents. It is desirable for the computation of the models, if time allows.

C. Gaussian Naive Bayes

One typical way for handling the continuous features in the NB classification is that Gaussian distributions are utilized for the signification of the probabilities of the features conditioned on the classes. Therefore, each attribute is described through a Gaussian probability density function (PDF) as

$$X_i \sim N(\mu, \sigma^2)$$

The Gaussian PDF has the shape of a bell and is defined using the following equation:

$$N\,(\mu, \sigma^2)(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e - \frac{(x-\mu)^2}{2\sigma^2}$$

In which μ is the mean and $\sigma^2$is the variance. In NB, the parameters required are in the order of $O(n,k)$ in which n is the number of attribute and k represents the number of classes. In particular, there is a requirement of defining a normal distribution $P(X_i \setminus C) \backsim N(\mu, \sigma^2)$for every continuous attribute. The parameters of such normal distributions is acquired with

$$\mu_{X_i|C=c} = \frac{1}{N_c}\sum_{i=1}^{N_c} x_i$$

$$\sigma^2{}_{X_i|C=c} = \frac{1}{N_c}\sum_{i=1}^{N_c} x_i{}^2 - \mu^2$$

In this $N_c$is the number of instances in which $C = c$ and $N_c$is the number of total instances utilized while training. The calculation of $P(C = c)$for all the classes is easy with the help of relative frequencies such that

$$P(C = c) = \frac{N_c}{N}$$

D. Random forest

RF is an ensemble learning algorithm. This basic principle of the algorithm is that a small DT is developed with few attributes is a computationally cheap procedure. When various small, weak DTs are developed in parallel, the trees are integrated for forming a single, strong learner after averaging or obtaining the majority vote. The RFs are often investigated as the most accurate learning algorithms to date in training.

Formally, a RF is a predictor in which a collection of randomized base regression trees are comprised as $\{r_n(x, \ominus_m, D_n), m \geq 1\}, where\ \ominus_1, \ominus_2$ .... are independent and identically distributed outputs of a randomized variable $\ominus$. These integration of RTs is done for developing the aggregated regression estimate

$$\bar{r}_n(X, D_n) = \mathbb{E}_\ominus[r_n(X, \ominus, D_n)],$$

In which $\mathbb{E}_\ominus$ represents the expectation in terms of the random parameter, conditionally on X and the data set D_n. In the following, to lessen a notation a little, the dependency of the estimates would be excluded in the sample, and written to illustrate $\bar{r}_n(X)$ rather than $\bar{r}_n(X, D_n)$. In practice, Monte Carlo computed the above expectation when the M RTs are produced and the average of the individual outcomes is taken. The randomizing variable ⊖ is carried out for determining performance of the successive cuts while developing the individual trees in which selection of the coordinate to split and position of the split are comprised. The variable ⊖ is deduced as the independent of X and the training sample D_n.

E. Principal Component Analysis

PCA is a statistical strategy using which the collection of interrelated factors are changed into a set of linearly unrelated subsets which are depending on a transformation and provides the uncorrelated variables. The PCA is also called orthogonal linear transformation that assisted in generating a projection of the initial dataset to another projection system with the end objective in which the biggest variance includes a projection of the first coordinate, even as the second biggest variance consists of a projection of the second coordinate that describes that it is vertical to the first component. In essence, a linear transformation is positioned using PCA that is denoted as $z = W_k^T x$ in which $x \in R^d$, and $r < d$, for improving the variance of the data within the projected space. The $X = \{x_1, x_2, \dots \dots .., x_i\}$, $x_i \in R^d$, $z \in R^r$ and $r < d$ is used to express the data matrix and the transformation can be described with the help of a set of p-dimensional vectors of weights $W = \{w_1, w_2, \dots \dots .., w_p\}$, $w_p \in R^k$, which matches every $x_i$vector of X to a

$$t_{k(i)} = W_{|(i)}T_{x_i}$$

For boosting the variance, an initial weight $W1$has to observe to the condition defined as follows:

$$W_i = \arg max_{|w|} = \{\sum_i (x_i \cdot W)^2\}$$

A further expansion of the previous condition is provided as:

$$W_i = \arg max_{\|w\|=1} \{\|X.W\|^2\}$$
$$= \arg max_{\|w\|=1} \{W^T X^T X W\}$$

A symmetric grid such as the $X^T X$, can be analysed in efficient way, when the biggest eigenvalue of the matrix is achieved, as W is the related eigenvector. After obtaining the W_1, the primary principal component can be inferred through the projection of initial data matrix X onto the W_1 in the space which is provided in the transformation. The further segments are obtained along these lines subsequent to the subtraction of the newly obtained components.

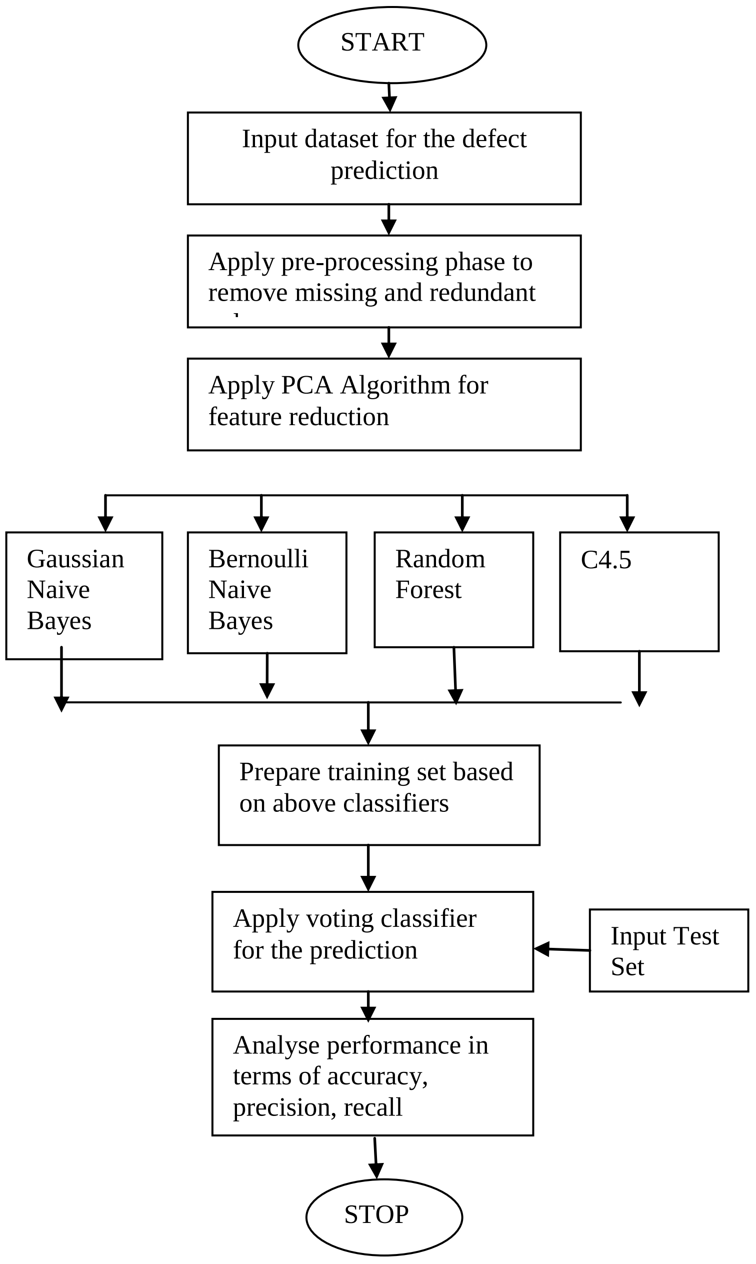
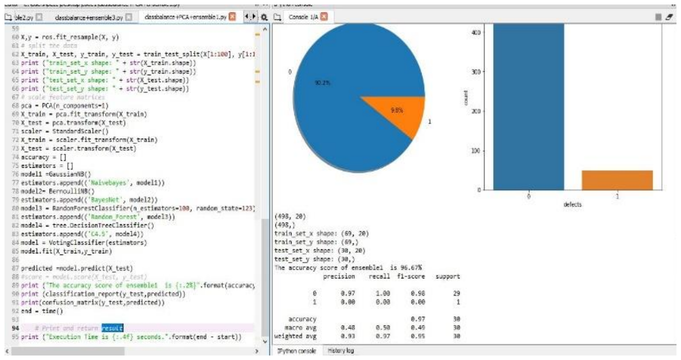

START
Input dataset for the defect prediction
Apply pre-processing phase to remove missing and redundant
Apply PCA Algorithm for feature reduction
Gaussian Naive Bayes
Bernoulli Naive Bayes
Random Forest
C4.5
Prepare training set based on above classifiers
Apply voting classifier for the prediction
Input Test Set
Analyse performance in terms of accuracy, precision, recall
STOP

Figure 3.1: Proposed Methodology

IV. RESULTS AND DISCUSSION

In this work "CM1/Software Defect Prediction" which is available from PROMISE SE Repository is examined and used. It contains 498 records and 22 attributes (5 different lines of code measure, 3 McCabe metrics, 4 base Halstead measures, 8 derived Halstead measures, a branch count, and 1 goal field). This sample data is chosen in the work because it comes from a legitimate source and is freely available.



```
X,y = ros.fit_resample(X, y)
# split the data
X_train, X_test, y_train, y_test = train_test_split(X[1:100], y[1:1
print ("train_set_x shape: " + str(X_train.shape))
print ("train_set_y shape: " + str(y_train.shape))
print ("test_set_x shape: " + str(X_test.shape))
print ("test_set_y shape: " + str(y_test.shape))
# scale feature matrices
pca = PCA(n_components=1)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
accuracy = []
estimators = []
model1 =GaussianNB()
estimators.append(('Naivebayes', model1))
model2= BernoulliNB()
estimators.append(('BayesNet', model2))
model3 = RandomForestClassifier(n_estimators=100, random_state=123)
estimators.append(('Random_Forest', model3))
model4 = tree.DecisionTreeClassifier()
estimators.append(('C4.5', model4))
model = VotingClassifier(estimators)
model.fit(X_train,y_train)

predicted =model.predict(X_test)
#score = model.score(X_test, y_test)
print ("The accuracy score of ensemble1  is {:.2%}".format(accurac
print (classification_report(y_test,predicted))
print(confusion_matrix(y_test,predicted))
end = time()

    # Print and return result
print ("Execution Time is {:.4f} seconds.".format(end - start))
```

```
(498, 20)
(498,)
train_set_x shape: (69, 20)
train_set_y shape: (69,)
test_set_x shape: (30, 20)
test_set_y shape: (30,)
The accuracy score of ensemble1  is 96.67%
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        29
           1       0.00      0.00      0.00         1

    accuracy                           0.97        30
   macro avg       0.48      0.50      0.49        30
weighted avg       0.93      0.97      0.95        30
```

Figure 4.1: PCA with Class Balancing and Ensemble Classifier

As shown in figure 4.1, the class balancing with PCA and ensemble classifier is the combination of four classifiers which are Gaussian Naive Bayes, Bernoulli Naive Bayes, Random Forest and C4.5

The results of all the models which are implemented are shown in terms of accuracy, precision and recall. The results of all the classifier are shown in table 1

Table 1. Result Analysis

Model Name	Accuracy %	Precision %	Recall %
Bernoulli NB	74	15.15	31.25
C4.5	84.67	23.08	18.75
Gaussian NB	80.67	11.76	12.50
MLP Classifier	82.67	18.75	18.75
SVC(kernel=-linear)	88.67	33.33	6.25
Random Forest	87.33	20	6.25
Proposed Model	96.67	93	97

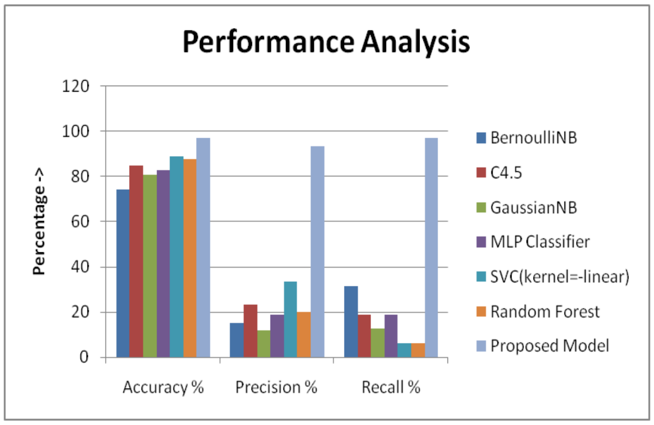


Figure 4.2: Performance of Models

The various classifiers like Bernoulli Naive Bayes, Gaussian Naive Bayes, Random Forest, Decision tree, MLP and SVM are applied for the software defect prediction. As shown in figure 4.2, when the individual classifier are applied no feature extraction algorithm like PCA or class balancing is applied for the software defect prediction. The proposed model is the combination of Bernoulli naive Bayes, Gaussian Naive Bayes, Random forest, C4.5 and also PCA is applied for the feature extraction with class balancing. The proposed Model give maximum accuracy, precision and recall as compared to other individual classifiers for the software defect prediction.

CONCLUSION

Software defect along with an intrinsic element of software product, is also an important aspect of software quality. Software defects are an unavoidable co-product of the developed software. In addition to this, the guarantee of software quality assurance is not so easy and requires a lot of time too. There are different ways to define defects, such as in terms of quality. However, the defects are generally defined in the form of deviations from specifications or expectations which may be the reason of failure in functioning. In this research work, various individual like Gaussian Naive Bayes, Bernoulli Naive Bayes, Random Forest, C4.5, SVM and MLP are implemented for the software defect prediction. The ensemble classifier is generated for the software defect prediction which is the combination of Gaussian Naive Bayes, Bernoulli Naive Bayes, Random forest and C4.5 respectively. The feature extraction technique called PCA is merged with the ensemble classifier and also class are balanced for the software defect prediction. It is analyzed that model in which ensemble classifier is applied with feature extraction and class balancing give maximum accuracy of 96.67 percent as compared to other classifiers.

REFERENCES

[1] L. Zhang and Z. Shang, "CLASSIFYING FEATURE DESCRIPTION FOR SOFTWARE DEFECT," pp. 10–13, 2011.

[2] J. Wang, B. Shen, and Y. Chen, "Compressed C4 . 5 Models for Software Defect Prediction," vol. 2, no. 1, pp. 4–7, 2012, doi: 10.1109/QSIC.2012.19.

[3] Y. Chen and B. Ge, "Research on Software Defect Prediction Based on Data Mining," vol. 1, pp. 563–567, 2010.

[4] Y. Guo and M. Shepperd, "Poster : Bridging Effort-Aware Prediction and Strong Classification - a Just-in-Time Software Defect Prediction Study," no. 1,

pp. 325–326, 2018.

[5] C. Shan, H. Zhu, C. Hu, J. Cui, and J. Xue, "Software Defect Prediction Model Based on Improved LLE-SVM," no. Iccsnt, pp. 530–535, 2015.

[6] T. Nadu, "Software Defect Prediction Using Software Metrics - A survey," pp. 2–5.

[7] K. E. Bennin, J. Keung, P. Phannachitta, and A. Monden, "MAHAKIL : Diversity based Oversampling Approach to Alleviate," vol. 5, no. 1903, p. 3182520, 2018.

[8] F. Wu *et al.*, "Cross-project and Within-project Semi-supervised Software Defect Prediction Problems Study Using a Unified Solution," pp. 39–41, 2017, doi: 10.1109/ICSE-C.2017.72.

[9] G. Mauˇ and T. G. Grbac, "Software Defect Prediction with Bug-Code Analyzer - a Data Collection Tool Demo."

[10] H. A. Ai-jamimi, "Toward Comprehensible Software Defect Prediction Models Using Fuzzy Logic," pp. 127–130, 2016.

[11] H. Iru and U. S. Hihfw, "Manifold Learning for Cross-project Software Defect Prediction," *2018 5th IEEE Int. Conf. Cloud Comput. Intell. Syst.*, pp. 567–571, 2018.

[12] S. I. Ayon, "Neural Network based Software Defect Prediction using Genetic Algorithm and Particle Swarm Optimization," *2019 1st Int. Conf. Adv. Sci. Eng. Robot. Technol.*, vol. 2019, no. Icasert, pp. 1–4, 2019.

[13] Y. Xia, "Analyzing The Significance of Process Metrics for TT & C Software Defect Prediction."