International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 02 | February - 2024 SJIF Rating: 8.176 ISSN: 2582-3930

Software Defect Prediction Using Ensemble Modelling and Class Balancing

Ms. Pooja Gupta[1], Ms.Komal Ahuja[2]

M.Tech , Computer Science Engineering, Global Research institute of Management & Technology, Radaur

Assistant Professor, Computer Science Engineering, Global Research institute of Management & Technology, Radaur

Abstract

Software product is a kind of software whose development is done to accomplish a precise requirement. Meanwhile, engineering is branch which is related to develop a product based on explicit technical fundamentals and techniques. There are diverse phases executed to predict the defect in software such as to employ the data for input, pre-process it, extract the attributes and classify the defect. This research work presents numerous algorithms, namely gaussian naive bayes (GNB), Bernoulli NB, random forest (RF) and multi-layer perceptron (MLP), for predicting the software defect. This work focuses on developing an ensemble algorithm to enhance the efficacy of predicting the defects. This ensemble consisted of Principal Component Analysis (PCA) algorithm with class balancing. Python is executed for simulating the introduced model. Diverse parameters such as accuracy, precision and recall are employed for analyzing the results.

Keywords—Software Defect, Gaussian Naive Bayes, Bernoulli Naive Bayes, Random Forest, PCA, Class Balancing

1. Introduction

Due to the growing complexity of modern software and the increased risk of failures, ensuring reliability has become a crucial focus. Organizations like Google employ code review and unit testing to detect issues in new code and enhance reliability. However, testing every code unit is impractical, and human code reviews are labour-intensive [1]. With limited funding for software projects, it is beneficial to identify potential issues early. Consequently, software defect prediction algorithms are commonly employed to automatically detect possible flaws, enabling developers to make efficient use of their resources. Software defect prediction involves creating classifiers that analyze data such as change history and code complexity to identify code segments with potential flaws. This practice allows code reviewers to allocate their efforts strategically and receive warnings about potentially buggy code regions based on the prediction results [2]. These code sections could include modifications, files, or procedures. In the typical fault prediction process, there are two main stages: feature extraction from source files and the creation of a classifier using various machine learning techniques. Previous research focused on enhancing the precision of predictions has primarily involved manually crafting discriminative features or combining features [3]. Examples include Halstead features based on operators and operands, McCabe features based on dependencies, and CK features for object-oriented programs. However, traditional hand-crafted features often overlook the intricate semantics and well-defined syntax concealed in the Abstract Syntax Trees (ASTs) of programs.

Abstract Syntax Trees (ASTs) offer structural information that details the interactions between adjacent tokens or nodes to execute specific functions [4]. Even a slight change in local structure can lead to a wide range of program

outcomes, including crashes. Consequently, conventional forecasting approaches may yield insufficient results. Recently, machine learning has proven to be a powerful tool for automated feature development due to its ability to effectively capture highly complex non-linear features. The process of pinpointing code sections that might harbour faults is termed "software defect prediction", aiding developers in optimizing their testing resources by prioritizing a review of potentially problematic code [5]. Modern large-scale software heavily relies on defect prediction to ensure its reliability. Figure 1 illustrates a commonly employed file-level fault prediction procedure.

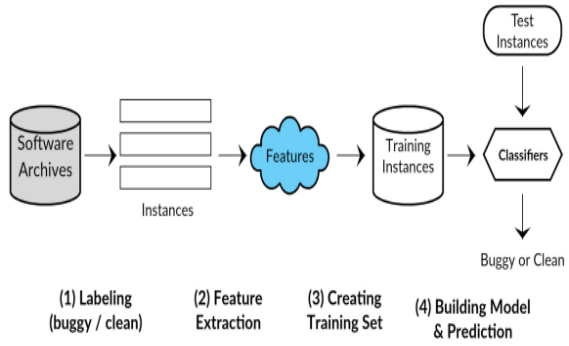

Figure 1: Defect Prediction Process

In the depicted procedure, the initial stage involves collecting source code files (instances) from software archives and classifying them as either clean or buggy [6]. The labelling procedure depends on the number of post-release flaws in each file. If a file has at least one post-release bug, it is classified as buggy; otherwise, it is marked as clean. The second phase involves preparing features for each file. Conventional features, as defined in prior research, can be categorized into two groups: process metrics (like change histories) and code metrics (such as McCabe features and CK features) [7]. The examples with matching characteristics and labels are then used to train machine learning algorithms, such SVM, Naive Bayes, and Dictionary Learning, to create predictive classifiers. Ultimately, fresh examples are added to the trained classifier, enabling it to forecast whether the files are error-free or not. The test set consists of the cases used to assess the learnt classifier, whereas the training set consists of the instances used to develop the classifier [8]. SVM is mostly used to determine the best method for separating data into two classes. In other words, the decision determines the hyperplanes or limits them for this purpose. In high-dimensional space, it works well. When there are more dimensions than there are samples, it works well. To ensure the resilience of the decision to new information, the boundary line needs to be positioned as close as possible to the boundaries of the two classes [9]. Support points refer to those points that are closest to this boundary line.

Decision nodes in a decision tree consist of basic elements such as branches and leaves. The input space of a Decision Tree (DT) is divided into intersensory areas, and data points are identified by assigning a value or label to each region. The structure of a DT is observable, and its mechanism is transparent [10]. Typically, DTs go through two phases: first, a large tree is constructed, and in the second phase, it is pruned to prevent overgrowth issues. Following that, the tree is applied to the categorization and pruning processes. Named after the mathematician Thomas Bayes, the Naive Bayes classification algorithm employs a set of computations grounded in probability concepts to ascertain the class of the provided data [11]. In NB classification, the system is trained with a specific dataset, and a class of data must be provided for instruction. Probability operations are then applied to the trained data, allowing the system to process newly supplied test data and determine its category based on previously derived probability values. The accuracy of determining the true type of test data tends to improve with more data being incorporated into the training process [12].

In the K-Nearest Neighbour algorithm, the number of elements to be considered during the categorization phase depends on the specified k value. The algorithm involves training data, and when a new value is introduced, distances are calculated to the k nearest neighbours. Distance calculation methods, such as Manhattan and Euclidean distances, can be applied in this process. The algorithm comprises five stages [13]. Initially, the value of k is determined, followed by the computation of Euclidean distances between the target object and other items. The closest neighbours are identified by sorting distances based on minimum values. The categories associated with the closest neighbours are then combined, and the neighbour type that best fits the situation is chosen. It is crucial to select the k value wisely and have a

substantial training set for optimal results [14]. The Random Forest Classifier method aims to enhance the classification value by generating multiple Decision Trees (DT) during the classification process. These individually produced DTs collaborate to construct a decision forest. In this approach, each DT comprises randomly selected portions of the dataset to which they are associated. This method yields results quite rapidly. In the Adaboost algorithm, one approach involves focusing the pre-predictor more on missing education data to correct its predictions. The process begins by training and estimating the classifier training kit before constructing an Adaboost Classifier (AC) [15]. The "Relative Weight" of incorrectly identified training data is then increased. These higher weights are utilized to train and re-estimate the second classifier. The weights are updated once more and remain constant. Estimates are generated using bagging or pasting techniques after all the forecasters have undergone training, considering each forecaster's accuracy ratio. The Gradient Boosting Classifier (GBC) is a machine learning method designed for addressing gradient boosting, regression, and classification problems. It creates a model by combining weak prediction models, typically Decision Trees (DT) [16]. The primary objective of any supervised learning algorithm, including GBC, is to identify and minimize a loss function.

2. Literature Review

M. Ali, et.al (2024) suggested an intelligent ensemble-based framework in which various classification methods were integrated to predict software defects [17]. A two-phase procedure was executed for detecting defective modules. The initial phase was emphasized on 4 supervised machine learning (SML) methods such as Random Forest (RF), Support Vector Machine (SVM), Naïve Bayes (NB), and Artificial Neural Network (ANN). An iterative parameter optimization was utilized for optimizing these methods for attaining superior accuracy. The next phase aimed to incorporate the accuracy of every method into a voting ensemble for predicting defects in software. This framework made the process of predicting software defects more accurate and reliable. The suggested framework was simulated on 7 datasets taken from NASA MDP repository. The experimental results depicted the superiority of suggested framework over existing methods to predict software defects.

Y. Al-Smadi, et.al (2023) introduced an innovative method in order to predict software defects in which 11 machine learning (ML) techniques were implemented over 12 diverse datasets [18]. Four diverse meta-heuristic algorithms: particle swarm optimization (PSO), genetic algorithm (GA), harmony algorithm (HA), and ant colony optimization (ACO), were implemented to select features. Furthermore, the synthetic minority oversampling technique (SMOTE) was implemented for dealing with imbalanced data. Additionally, the decisive features were highlighted through Shapley additive explanation (SAE) framework. The experimental outcomes depicted that the gradient boosting (GB), stochastic gradient boosting (SGB), decision trees (DTs), and categorical boosting (CB) algorithms were performed more effectively and offered an accuracy and ROC-AUC over 90%. Moreover, the supremacy of last algorithm was proved against other methods for predicting software defects.

C. Yu, et.al (2021) emphasized on deploying homomorphic encryption (HE) to predict defect, and formulated a new technique called HOPE [19]. An algorithm approximation (AA) method was presented for approximating the sigmoid function and selecting the Paillier homomorphic encryption (PHE) algorithm to execute Logistical regression (LR). The real time projects were considered as experimental subjects for generating MORPH dataset for computing the formulated technique. Thereafter, 3 control groups were deployed for simulating 3 diverse scenarios on the basis of whether the client led to transmit the encrypted data to server and whether formulated technique was employed in server. According to results, in case of deployment of original LR in server for generating this technique on the encrypted data, a same performance was obtained from the trained model that results in protecting the privacy of data. Furthermore, the formulated technique was more efficient concerning least computing cost.

R. Haque, et.al (2024) presented an innovative technique known as heterogeneous cross-project defect prediction (HCDP) in which encoder networks and transfer learning (ENTL) model was implemented to predict defects occurred in

software [20]. The encoder networks (ENs) were exploited for extracting significant features from source and target data sets. Besides, the negative transfer was mitigated in transfer learning (TL) using an augmented dataset containing pseudo-labels and the source dataset. The presented model was trained on a single dataset and quantified on sixteen datasets, generated via 4 public projects. A comparative analysis was conducted on presented model against traditional methods. The cost-sensitive learning (CL) method was utilized for dealing with the imbalanced class issue. The experimental results indicated that the presented method was more robust to predict software defects with respect to PD, PF, F1-score, G-mean and AUC.

W. Wen, et.al (2022) developed a class code similarity-based cross-project software defect prediction (CCS-CPDP) method to predict software defects [21]. At first, this technique was focused on converting the code set taken from Abstract Syntax Tree (AST), into a vector set. For this, a Doc2Bow and TF-IDF (DTI) method was employed. The second task was to compute similarity amid the vector set of target projects and training projects. At last, the principle of the majority decision subordinate category was considered in K-Nearest Neighbor (KNN) to verify the number of same class occurrences of training project. The class instance was selected to refine the source project, and their prediction and computation was done on software defects. The developed method was computed against the traditional methods. According to experiments, the developed method was effective and offered higher recall and f1-score while predicting software defects in contrast to other methods.

S. Kassaymeh, et.al (2022) constructed a Salp Swarm Algorithm (SSA) with a backpropagation neural network (BPNN) for predicting software fault [22]. The SSA-BPNN, their integrated method, was presented for enhancing the accuracy to predict defects when the optimal metrics were optimized. Various datasets were applied to compute the presented method for dealing with the issue related to predict software defects with respect to various parameters, such as AUC, confusion matrix, sensitivity, specificity, accuracy, and error rate. The size and complexity of these datasets was varied. The simulation outcomes indicated that the presented method was performed more effectively in comparison with the conventional methods and yielded a higher accuracy to predict software defects. In addition, this technique was proved as an effective tool to tackle the problems related to software engineering.

W. Wen, et.al (2022) designed a cross-project defect prediction (CPDP) model known as BSLDP to predict defects occurred in software [23]. A bidirectional long and short-term memory (Bi-LSTM) algorithm was deployed with self-attention (SA) method to extract semantic information about source code files. Generally, the ALC model was executed for extracting source code semantics on the basis of source code files, and a classifier was projected depending upon the semantic information of source project and target project, called BSL, for generating a predictive framework. An equal meshing method was adopted for extracting semantic information on small fragments when the numerical token vector was split for enhancing the designed model. A PROMISE dataset was applied for computing the designed model. Based on experiments, the designed model was capable of predicting defects and enhanced the F1 up to 14.2%, 34.6%, 32.2% and 23.6% against 4 techniques respectively.

L. Yang, et.al (2021) investigated a hybrid of particle swarm optimization (PSO) and Salp Swarm Algorithm (SSA) called SSA-PSO for augmenting the convergence prior to individual update of SSA [24]. Meanwhile, a novel maximum likelihood estimation (MLE)-based fitness function (FF) of metrics was presented and employed to initialize metrics. Five sets of actual datasets were employed to evaluate the investigated approach against an individual technique. The experimental results revealed the stability of investigated approach over others with respect to higher convergence speed and accuracy. Besides, the novel FF was assisted in tackling the issues related to slow convergence speed and lower accuracy of solution. The investigated approach was proved more applicable to estimate and predict defects occurred in software.

S. Kwon, et.al (2023) projected a function-level just-in-time (JIT)-software defect prediction (SDP) technique depending upon a pre-trained method for dealing with the drawback of predicting software

defects [25]. For this, the limited testing resources were prioritized for the defect-prone functions [25]. The transformer-based deep learning (TDL) model was utilized and its training was done a large corpus of code snippets, which further provided a defect proneness for the altered functions at a commit level. The CodeBERT, GraphCodeBERT and UniXCoder methods were computed on edge-cloud systems. The primary emphasis was on analyzing the efficacy of these methods for edge-cloud systems. The results exhibited that the last method was performed better as compared to others in the WPDP environment. moreover, the projected technique was proved stable to predict software defects.

A. Wang, et.al (2023) developed a federal prototype learning based on prototype averaging (FPLPA) method in which federated learning (FL) was integrated with prototype learning (PL) to predict heterogeneous defect in software [26]. The one-sided selection (OSS) algorithm was implemented for eliminating noise from local training data, and the Chi-Squares Test algorithm was employed for selecting the optimal subset of features. Thereafter, the convolution prototype network (CPN) model was put forward for creating their own local prototypes. This model had offered more robustness against heterogeneous data in contrast to convolutional neural networks (CNN), to avoid the deviation impact of class imbalances in software data. The prototype was taken in account as the communication subject amid clients and server. The local prototype was developed irreversibly for protecting privacy in the communication procedure. The last task was to update the presented model using the loss of local prototype and global prototype as regularization. The AEEEM, NASA and Relink datasets were applied for quantifying the developed method. The simulation outcomes depicted the superiority of developed method over traditional methods for predicting defect.

3. Research Methodology

The presented methodology is conducted on various models, called random forest, Gaussian Naïve Bayes, BNB and Decision Tree. This work projects an ensemble algorithm for predicting the defect in software. This algorithm is an integration of Gaussian Naïve Bayes, BNB, Random Forest and Multilayer Perceptron. In the end, PCA algorithm is implemented to extract the features. The utilized algorithms are discussed as:

A. Multilayer Perceptron

It is a kind of FFN. More than one perceptron is involved in this algorithm. The outcome generated from one perceptron is fed into the next one as input. Furthermore, the state of a neuron is evaluated using a nonlinear function. Figure 3 represents a general framework of Multilayer Perceptron algorithm.

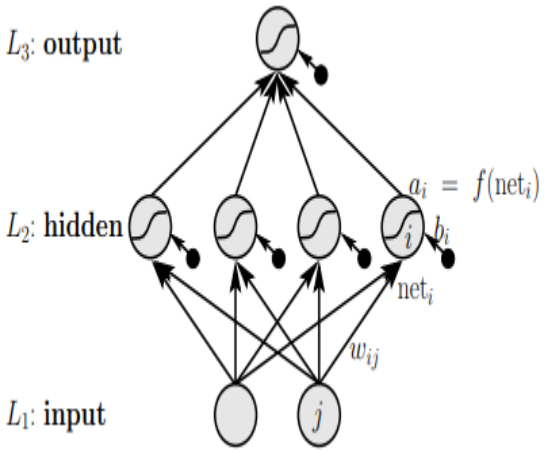

Figure 2: Multilayer Perceptron

In the given figure:

a_i= activity of the ith unit.

a_o=1: activity of 1 of the bias unit

w_{ij}= weight from unit j to unit i

w_{io}= = b_i : bias weight of unit i

W: number of weights

N: number of units

I: number of inputs units $(1 \leq i \leq I)$ placed in the first layer called the input layer

O: number of output units $(N - O + 1 \leq i \leq N)$ available in the last layer called the output layer

M: number of hidden units $(I < i \leq N - O)$ present in the hidden layers.

L: number of layers, at which L_v illustrates the index set of the vth layer; $L_1 = \{1, \ldots, I\}$ and $L_1 = \{N - O + 1, \ldots, N\}$

net_i : network input to the ith unit (I <i) calculated as:

$$net_i = \sum_{j=0}^{N} w_{ij} a_j \quad (1)$$

f: activation function with

$$a_i = f(net_i) \quad (2)$$

A number of activation functions (AFs) f_i are defined for distinct units. AF is also called the transfer function.

An FF-MLP has only links which are taken from units in lower layers to units of superior ones:

$$i \in L_v \; and \; j \in L_{v'} \; and \; v' \leq v \Rightarrow \; w_{ij} = 0 \quad (3)$$

The traditional algorithm only uses associations or weights between successive layers. Other weights have been given the value of zero. After that, the input is considered for the node in the output layer or hidden layer v, where v > 1.

$$\forall_{i \epsilon L_v} : net_i = \sum_{j:j \epsilon L_{V=1}}^{N} \quad w_{ij} \, a_j \quad (4)$$

Non-adjacent associates amongst units, present in layers are known as shortcut connections.

Activation Functions

Sigmoid function is a major kind of AFs. The logistic function is expressed as:

$$f(a) = \frac{1}{1+exp\,(-a)} \quad (5)$$

and *tanh* AF is defined below in given equation:

$$f(a) = tanh\; tanh\;(a) \; = \frac{expexp\,(a) - exp\,(-a)}{expexp\,(a) + exp\,(-a)} \quad (6)$$

B. Bernoulli Naive Bayes

This algorithm aims to train Naïve Bayes (NB) and classification models are included in this algorithm for distributes data with regard to multivariate Bernoulli distributions. It implies the availability of a variety of attributes. In addition, each attribute is employed as a binary-valued variable. Hence, there is necessity of samples for the class which are utilized as feature vectors of binary value. This algorithm is responsible for binarizing its input when it handles other type of data. Its decision rule is expressed as:

$$P(x_i|y) = P(i|y)x_i + \big(1 - P(i|y)\big)\,(1 - x_i) \quad (7)$$

Unlike the multinomial NB's rule, this rule is executed to penalize the non-occurrence of attribute i and this attribute is considered as a factor for class y. The multinomial variant doesn't consider this attribute. Bernoulli Naive Bayes is simulated and trained on the basis of word occurrence vectors to classify the text. On some data sets of smaller documents, this algorithm offers efficiency. The major task is of computing the frameworks concerning time.

C. Gaussian Naive Bayes

The Gaussian distributions is implemented in the Naive Bayes (NB) algorithm for handling the continuous features in order to illustrate the likelihoods of the features related to the classes. Therefore, a Gaussian PDF assists in defining every feature as:

$$X_i \sim N(\mu, \sigma^2) \quad (8)$$

The shape of Gaussian probability density function is similar to a bell and it can be defined mathematically as:

$$N\;(\mu, \sigma^2)(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e - \frac{(x-\mu)^2}{2\sigma^2} \quad (9)$$

In which, μ is used to signify the mean and σ^2 defines the variance. The parameters employed in NB model must be available as $O(n, k)$ in which n is total features and the number of classes is illustrated with k. In particular, every continuous feature has a normal distribution $P(X_i \setminus C) \backsim N(\mu, \sigma^2)$. The metrics of these normal distributions are expressed as

$$\mu_{X_i|C=c} = \frac{1}{N_c}\sum_{i=1}^{N_c} \quad x_i \quad (10)$$

$$\sigma^2{}_{X_i|C=c} = \frac{1}{N_c}\sum_{i=1}^{N_c} \quad {x_i}^2 - \mu^2 \quad (11)$$

In which, N_c is used to denote the number of instances in which C is equal to c and N denotes the total instances available to train the data. The relative frequencies are assisted in computing $P(C = c)$ for all the classes as:

$$P(C = c) = \frac{N_c}{N} \quad (12)$$

D. Random Forest

It is considered as an ensemble system. The notion, related to develop a tiny decision tree (DT) on the basis of some features, is considered. This algorithm consumes least cost. The trees are combined after creating various small and weak DTs in parallel so that a single and strong learner is developed subsequent to achieve the majority votes. This algorithm is presented as an effective learning method of superior accuracy in the training stage.

Particularly, RF is a predictive tool in which diverse randomized base regression trees are implemented as $\{r_n(x, \ominus_m, D_n), m \geq 1\}$, here, $\ominus_1, \ominus_2$ .... have not any association among one another. This algorithm employs Regression Trees for creating the aggregated regression estimate as:

$$\underline{r}_n(X, D_n) = E_{\ominus}[r_n(X, \ominus, D_n)], \quad (13)$$

This equation contains $E_{\ominus}$ to represent the expectation with random metric, that is conditioned on X and the data set D_n. This algorithm aims to exclude the dependence of the estimates in the sample which can alleviate a notation, and to write it for defining $\underline{r}_n(X)$ rather than $\underline{r}_n(X, D_n)$. In particular, the above expression is quantified on the basis of Monte Carlo. For this, the random trees are extracted and the average of the discrete results are utilized. The efficiency of consecutive cuts is computed with respect to randomizing variable $\ominus$. Random Forest algorithm emphasizes on creating

the trees individually for selecting the coordinate so that the split and its position are comprised. The variable $\ominus$ is used to define an independent variable X and D_n is the training sample.

E. Principal Component Analysis

It is a robust method that is effective to alter the group of consistent elements into a set of linearly unconnected subsets which are depending on a conversion, and the distinct variables are generated using this method. This method is also called as an orthogonal linear transformation (LT) and its implementation is done to project the primary dataset with another projection system. The projection of the 1st coordinate is considered in the largest variance, and a projection of the 2nd one is kept in the 2nd largest variance. This algorithm helps in locating the LT as $z = W_k^T \; in \; which \; x \in R^d$, and $r < d$, enhancing the variance of the data in the projected space. The $X = \{x_1, x_2, \dots\dots\dots, x_i\}$, $x_i \in R^d$, $z \in R^r$ and $r < d$ is utilized to denote the data matrix and a set of p-dimensional vectors of weights $W = \{w_1, w_2, \dots\dots\dots, w_p\}$, $w_p \in R^k$ are considered for defining the transformation that contains every $x_i$vector of X's matching with

$$t_{k(i)} = W_{|(i)} T_{x_i} \qquad (14)$$

For maximizing the variance, an initial weight W_1 must have to satisfy a condition:

$$W_i = arg \; arg \; max \; _{|w|} = \{\sum_i \quad (x_i \cdot W)^2\} \qquad (15)$$

This condition is further expanded as:

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

This algorithm aims to analyze a symmetric grid $X^T X$ successfully after attaining the chief Eigen value of the matrix as W. Subsequent to generateW_1, this algorithm focuses on projecting the primary data matrix X projected onto the W_1 in the space for assuming the preliminary PC in the conversion. This results in attaining the additional segments along these lines when the attained elements are subtracted.

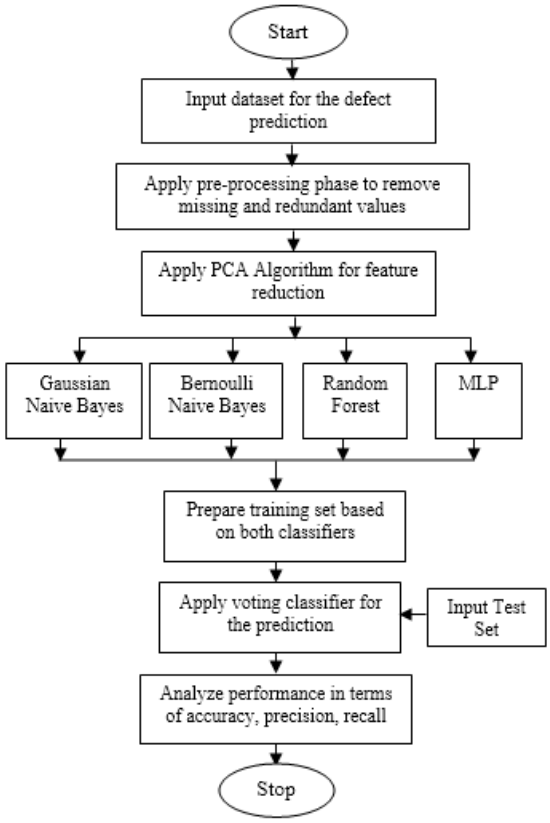

Figure 3 Proposed Methodology

4. Results and Discussion

This work aims to analyse and implement the "CM1/Software Defect Prediction" whose extraction is done from PROMISE SE Repository. This dataset has498 records and twenty-two features. Moreover, it also contains five diverse lines of code measure, 3 McCabe parameters, 4 base Halstead, 8 derived Halstead, a branch count, and 1 goal field. This work chooses this sample data as it is generated through the authentic source and it is present publicly.

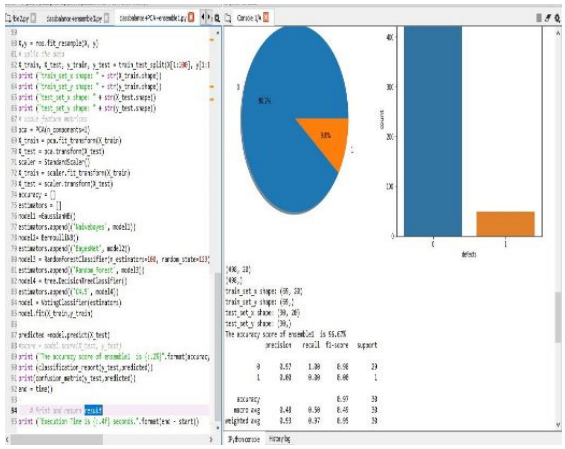

Figure 4 Proposed Methodology

Figure 4 demonstrates the implementation of class balancing with PCA ensemble 1 algorithm that put 4 algorithms, called Gaussian Naïve Bayes, Bernoulli NB, Random Forest and C4.5 together.

TABLE I. 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

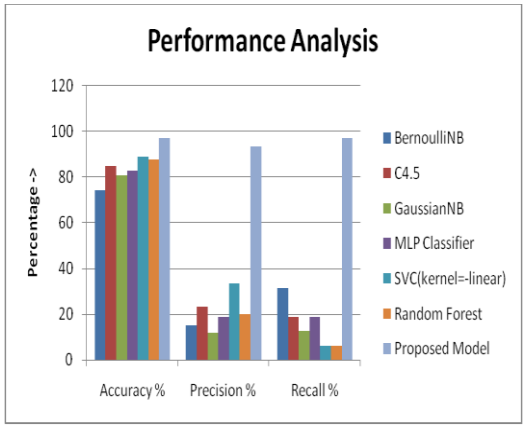

Figure 5: Performance of Models

The implementation of different algorithms, namely BNB, GNB, RF, DT, MLP and SVM is done for predicting the defect occurred in software. Figure 5 illustrates that for deploying an individual algorithm, no technique of extracting features including PCA or class balancing is considered while predicting the software fault. The suggested technique is an ensemble of BNB, GNB, RF and MLP. PCA with class balancing is adopted to extract the features. The suggested technique has generated more optimal results in comparison with the existing methods for predicting the faults occurred in software.

Conclusion

Software defect and a basic component of software product are the major elements of software quality. The defects occurred on software are unavoidable components. Furthermore, there is not any surety of quality of software, and more time is required to compute it. Different ways are present to define the defects based on quality. However, the complex task is to predict the defects in software. This research work presents a number of techniques such as, GNB, BNB, RF, C4.5, SVM and MLP for predicting the software defect. Moreover, it suggests an ensemble approach of GNB, BNB, Random Forest and Multilayer Perceptron to predict the software defects. Moreover, PCA along with the ensemble 1 class balance is adopted to predict the software defect. The accuracy attained from the suggested technique is calculated 96.67% as compared to the existing methods.

References

[1] H. Chen, X. -Y. Jing and B. Xu, "Heterogeneous Defect Prediction through Joint Metric Selection and Matching," 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), Hainan, China, 2021, pp. 367-377

[2] P Lakshmi, T. LathaMaheswari, “An effective rank approach to software defect prediction using software metrics”, 10th International Conference on Intelligent Systems and Control (ISCO), vol. 3, issue 21, pp. 679-684, 2019

[3] M. Kakkar, S. Jain, A. Bansal, P.S. Grover, “Evaluating Missing Values for Software Defect Prediction”, International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), vol. 37, issue 14, pp. 543-554, 2019

[4] S. Agarwal, S. Gupta, R. Aggarwal, S. Maheshwari, L. Goel, S. Gupta, “Substantiation of Software Defect Prediction using Statistical Learning: An Empirical Study”, 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU), vol. 5, issue 11, pp. 109-115, 2019

[5] J. Huang, X. Guan and S. Li, "Software Defect Prediction Model Based on Attention Mechanism," 2021 International Conference on Computer Engineering and Application (ICCEA), Kunming, China, 2021, pp. 338-345

[6] M. M. Ahmed, B. S. Kiran, P. H. Sai and M. Bisi, "Software Fault-Prone Module Classification Using Learning Automata based Deep Neural Network Model," 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2021, pp. 1-6

[7] A. Joon, R. Kumar Tyagi and K. Kumar, "Noise Filtering and Imbalance Class Distribution Removal for Optimizing Software Fault Prediction using Best Software Metrics Suite," 2020 5th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2020, pp. 1381-1389

[8] S. Kassaymeh, S. Abdullah and M. Alweshah, "Salp swarm optimizer for modeling the software fault prediction problem", Journal of King Saud University - Computer and Information Sciences, vol. 4, no. 5, pp. 1402-1406, 11 February 2021

[9] R. Chennappan and Vidyaathulasiraman, "An automated software failure prediction technique using hybrid machine learning algorithms", Journal of Engineering Research, vol. 11, no. 1, pp. 1-8 20 January 2023

[10] S. Moudache and M. Badri, "Software Fault Prediction Based on Fault Probability and Impact," 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), Boca Raton, FL, USA, 2019, pp. 1178-1185

[11] J. Lee, J. Choi, D. Ryu and S. Kim, "Holistic Parameter Optimization for Software Defect Prediction," in IEEE Access, vol. 10, pp. 106781-106797, 2022

[12] J. Deng, L. Lu, S. Qiu and Y. Ou, "A Suitable AST Node Granularity and Multi-Kernel Transfer Convolutional Neural Network for Cross-Project Defect Prediction," in IEEE Access, vol. 8, pp. 66647-66661, 2020

[13] L. Šikić, A. S. Kurdija, K. Vladimir and M. Šilić, "Graph Neural Network for Source Code Defect Prediction," in IEEE Access, vol. 10, pp. 10402-10415, 2022

[14] R. Chennappan and Vidyaathulasiraman, "An automated software failure prediction technique using hybrid machine learning algorithms", Journal of Engineering Research, vol. 7, no. 4, pp. 127–131, 20 January 2023

[15] A. Wang, Y. Zhao, G. Li, J. Zhang, H. Wu and Y. Iwahori, "Heterogeneous Defect Prediction Based on Federated Reinforcement Learning via Gradient Clustering," in IEEE Access, vol. 10, pp. 87832-87843, 2022

[16] Z. Yuan, X. Chen, Z. Cui and Y. Mu, "ALTRA: Cross-Project Software Defect Prediction via Active Learning and Tradaboost," in IEEE Access, vol. 8, pp. 30037-30049, 2020

[17] M. Ali et al., "Software Defect Prediction Using an Intelligent Ensemble-Based Model," in IEEE Access, vol. 13, no. 4, pp. 127-134, 2024, doi: 10.1109/ACCESS.2024.3358201.

[18] Y. Al-Smadi, M. Eshtay and A. A. Abd El-Aziz, "Reliable prediction of software defects using Shapley interpretable machine learning models", Egyptian Informatics Journal, vol. 24, no. 3, pp. 386-394, 31 July 2023, doi: 10.1016/j.eij.2023.05.011.

[19] C. Yu, Z. Ding and X. Chen, "HOPE: Software Defect Prediction Model Construction Method via Homomorphic Encryption," in IEEE Access, vol. 9, pp. 69405-69417, 2021, doi: 10.1109/ACCESS.2021.3078265.

[20] R. Haque, A. Ali, S. McClean, I. Cleland and J. Noppen, "Heterogeneous Cross-Project Defect Prediction Using Encoder Networks and Transfer Learning," in IEEE Access, vol. 12, pp. 409-419, 2024, doi: 10.1109/ACCESS.2023.3343329.

[21] W. Wen et al., "Cross-Project Software Defect Prediction Based on Class Code Similarity," in IEEE Access, vol. 10, pp. 105485-105495, 2022, doi: 10.1109/ACCESS.2022.3211401.

[22] S. Kassaymeh, S. Abdullah and M. Alweshah, "Salp swarm optimizer for modeling the software fault prediction problem", Journal of King Saud University - Computer and Information Sciences, 11 February 2021, vol. 34, no. 6, pp. 3365-3378, June 2022, doi: 10.1016/j.jksuci.2021.01.015.

[23] W. Wen et al., "A Cross-Project Defect Prediction Model Based on Deep Learning With Self-Attention," in IEEE Access, vol. 10, pp. 110385-110401, 2022, doi: 10.1109/ACCESS.2022.3214536.

[24] L. Yang, Z. Li, D. Wang, H. Miao and Z. Wang, "Software Defects Prediction Based on Hybrid Particle Swarm Optimization and Sparrow Search Algorithm," in IEEE Access, vol. 9, pp. 60865-60879, 2021, doi: 10.1109/ACCESS.2021.3072993.

[25] S. Kwon, S. Lee, D. Ryu and J. Baik, "Pre-Trained Model-Based Software Defect Prediction for Edge-Cloud Systems," in Journal of Web Engineering, vol. 22, no. 2, pp. 255-278, March 2023, doi: 10.13052/jwe1540-9589.2223.

[26] A. Wang, L. Yang, H. Wu and Y. Iwahori, "Heterogeneous Defect Prediction Based on Federated Prototype Learning," in IEEE Access, vol. 11, pp. 98618-98632, 2023, doi: 10.1109/ACCESS.2023.3313001.