International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 03 | March - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Design and Deployment of a Machine Learning Model for Software Defect Detection

Charulata Chouhan[1], Komal paliwal[2]

[1]M.Tech Student, Department of Electrical Engineering, SITE Nathdwara
[2]Associate Professor, Department of Electrical Engineering, SITE Nathdwara

---

Abstract - This paper explores software defects as an inherent aspect of software products, significantly impacting software quality. Defects, often deviations from specifications, can lead to functionality failures. Ensuring software quality assurance is complex and time-consuming, with many projects lacking sufficient resources to eliminate all defects before release. This can affect product quality and an organization's reputation. To address this challenge, various techniques for software defect prediction are employed. This research utilizes pre-processing, feature extraction, and classification methods. A hybrid approach integrating a genetic algorithm with PSO is used for feature extraction, while bagging classification generates final results. Three ensemble classifiers are implemented, and incorporating PCA for feature reduction further enhances accuracy. Additionally, addressing the class imbalance problem improves prediction performance, achieving 93% accuracy. This study provides an effective approach to software defect detection, contributing to improved software quality and reliability.

Key Words: Software Defect, Gaussian Naïve Bayes, Bernoulli Naïve Bayes, Random Forest, PCA, Class Imbalance Handling

1.INTRODUCTION

The rapid advancements in information technology have fueled an increasing demand for enhanced software functionality, leading to larger software systems and a higher incidence of defects. These defects not only compromise software functionality but also result in significant financial losses. As a result, software defect prediction has become a crucial aspect of software development, aiming to identify potential errors before deployment. There are two primary approaches to software defect prediction: static defect prediction and dynamic defect prediction. Static defect prediction analyzes the relationship between software defects and attributes such as code complexity and size to forecast potential issues. Over time, defect distribution models and prediction algorithms have been developed based on software metrics. In contrast, dynamic defect prediction examines defects over time, using statistical models like the Rayleigh distribution to analyze defect trends throughout the software lifecycle.

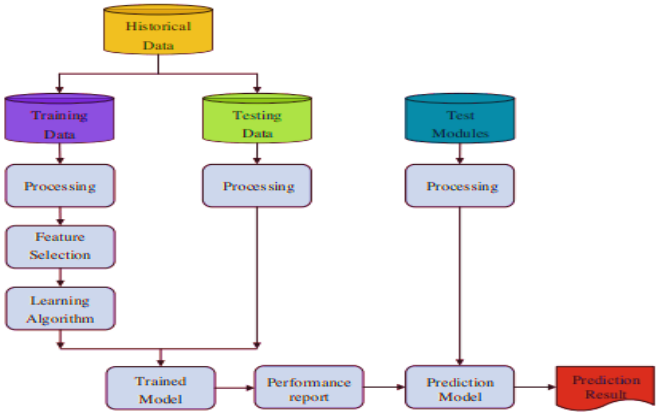

Figure 1.1 Software defects prediction model

Software defects not only lead to wasted resources but can also cause severe failures in deployed systems. Detecting and addressing defects early in the Software Development Life Cycle (SDLC) is essential for ensuring high-quality and reliable software. Static defect prediction involves designing software metrics based on code analysis, mining historical data, and building models to classify software modules as either defective or non-defective. The goal is to improve software quality by optimizing test resource allocation. In recent years, machine learning has significantly advanced static defect prediction. Various machine learning algorithms are used to classify software modules (binary classification) or predict the number of defects (regression analysis).

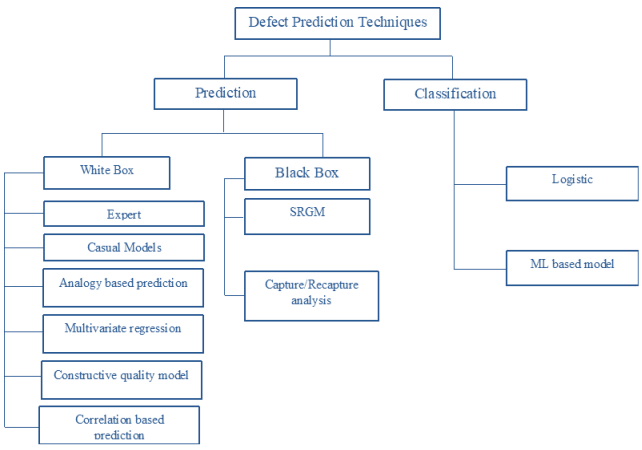

Figure 1.2: Software Defect Prediction Techniques

The software defect prediction process consists of several stages. First, essential information about software, such as code and development processes, is collected to establish relevant metrics. However, raw data often contains imbalances, outliers, and missing values, requiring extensive pre-processing. This includes handling missing values, detecting outliers, and normalizing data. Proper data cleaning and pre-processing improve the dataset's quality, ensuring accurate defect prediction. Additionally, feature selection or dimensionality reduction is necessary to eliminate redundant or highly correlated attributes, enhancing model efficiency. The next step involves training a classification model based on labeled data. The classifier is trained using validation data, adjusting parameters to optimize performance. In the final phase, the trained model predicts defects in new software modules.

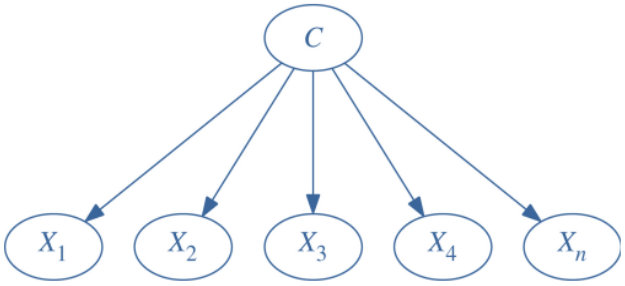

Figure 1.3: A class node C with n attributes

Classification-based methods apply algorithms to training data, learning patterns that are then validated using cross-validation techniques. One widely used method in defect prediction is the Bayesian network, a probabilistic graphical model that represents relationships between variables through a directed acyclic graph (DAG). Each node represents a variable, while edges encode conditional dependencies. The Naïve Bayesian classifier, a specialized Bayesian network, assumes conditional independence between features given the class label. Despite its simplicity, this model delivers strong performance in real-world applications due to its efficiency in probabilistic inference.

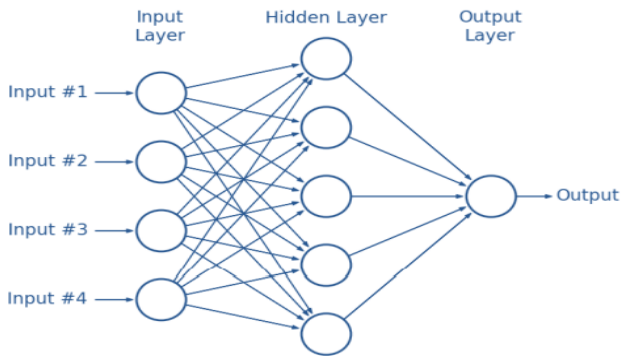

Figure 1.4: An artificial neural network with a 4-node input layer, 5 node hidden layer, and 1 node output layer

Another powerful approach is Artificial Neural Networks (ANNs), inspired by biological neural networks. ANNs consist of interconnected nodes (neurons) arranged in layers, with weighted connections representing information flow. In feed-forward neural networks, neurons are structured into an input layer, hidden layers, and an output layer. These models use activation functions, such as the sigmoid function, to process inputs and compute output probabilities. The most common learning algorithm for ANNs is backpropagation, which adjusts connection weights based on error gradients to improve classification accuracy.

Overall, software defect prediction plays a critical role in ensuring software quality by leveraging machine learning techniques to detect and mitigate defects early in development. Continuous advancements in machine learning models, data preprocessing strategies, and feature selection techniques contribute to more accurate and reliable defect prediction, ultimately leading to higher-quality software systems.

2. Literature Review

Recent studies have proposed various machine learning and deep learning techniques for software defect prediction. J. Lee et al. (2022) introduced a Cost-Sensitive Decision Tree using Harmony Search (HS-CSDT) to optimize defect prediction metrics. J. Deng et al. (2020) developed a Multi-Kernel Transfer CNN (MKT-CNN) for extracting semantic features from Abstract Syntax Trees (ASTs) to predict Cross-Project Defects. L. Šikić et al. (2022) designed a Graph Convolutional Neural Network (GCNN) framework for defect classification in Java projects. Other models include hybrid machine learning with Genetic Algorithms (Chennappan et al., 2023), Federated Reinforcement Learning (Wang et al., 2022), and Transformer-based self-attention models (Zheng et al., 2021). ALTRA, integrating Active Learning and TrAdaBoost, was proposed by Yuan et al. (2020), while Yang et al. (2023) introduced a method-level defect prediction using network embedding. Finally, Rahim et al. (2021) suggested a cost-effective approach combining Naïve Bayes and Linear Regression, achieving 98.7% accuracy. The reviewed studies highlight advancements in software defect prediction using diverse machine learning and deep learning techniques. Approaches like HS-CSDT, MKT-CNN, GCNN, and ALTRA optimize feature selection, semantic extraction, and defect classification. Hybrid models integrating Genetic Algorithms, Reinforcement Learning, and Transformer-based techniques further enhance predictive accuracy. Methods such as node2vec-based embedding and correlation-based ML models improve software quality by early fault detection. Overall, these techniques outperform traditional methods in accuracy, efficiency, and robustness, ensuring better defect prediction and allocation of quality assurance resources in software development.

3. Research Methodology

The proposed methodology is designed using multiple algorithms, including Random Forest (RF), Gaussian Naïve Bayes (GNB), Bernoulli Naïve Bayes, and Decision Tree (DT). This study introduces an ensemble algorithm for software defect prediction by integrating GNB, Bernoulli NB, RF, and Multi-Layer Perceptron (MLP). Finally, Principal Component Analysis (PCA) is applied for feature extraction. The employed algorithms are detailed as follows:

3.1 Multilayer Perceptron

It is a simple kind of Feed-forward network (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.

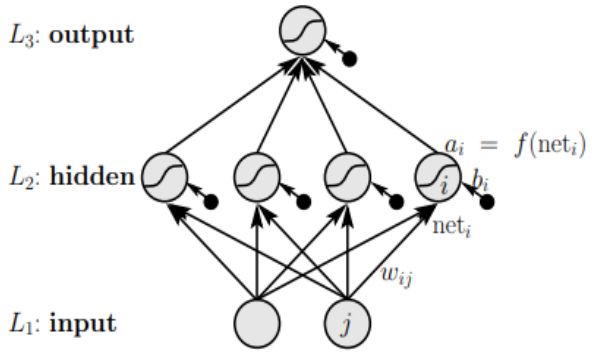

Figure 3.1: Multilayer Perceptron

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:

$$net_i = \sum_{j=0}^{N} w_{ij} a_j$$

f: activation function with

$$a_i = f(net_i)$$

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

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

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

The tradition algorithm is consisted of only connections or weights amongst consecutive layers. The value assigned to other weights is 0. Afterward, the network input is taken in account for node i in hidden or output layer ν in which ν > 1.

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

Non-adjacent connections among 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:

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

and tanh AF is defined below in given equation:

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

3.2 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 binary-valued feature vectors. This algorithm is responsible for binarizing its input when it handles other type of data. Its decision rule is expressed as:

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

Unlike the multinomial NB's rule, this rule is executed to penalize the non-occurrence of attribute i and this attribute is considered as an indicator 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.

3.3. 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:

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

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

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

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 amount 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

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

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

In which, N_c is used to denote the amount 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 the $P(C = c)$for all the classes as:

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

3.4 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:

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

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 dependency of the estimates in the sample to 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 individual outcomes are taken into consideration. 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.

3.5 Principal Component Analysis

It is a statistical method which is effective to alter the group of consistent elements into a set of linearly unconnected subsets which are depending on a conversion, and the uncorrelated 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 $r < d$, and enhancing the variance of the data within the projected space. The $X = \{x_1, x_2, \ldots\ldots, x_i\}$, $x_i \in R^d$, $z \in R^r$ and $r < d$ is utilized to denote the data matrix and a set of p-dimensional vectors of weights $W = \{w_1, w_2, \ldots\ldots, w_p\}$,

$w_p \in R^k$ are considered for defining the transformation that contains every $x_i$vector of X's matching with

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

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

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

This condition is further expanded as:

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

This algorithm aims to analyse 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 after the subtraction of the newly attained components.

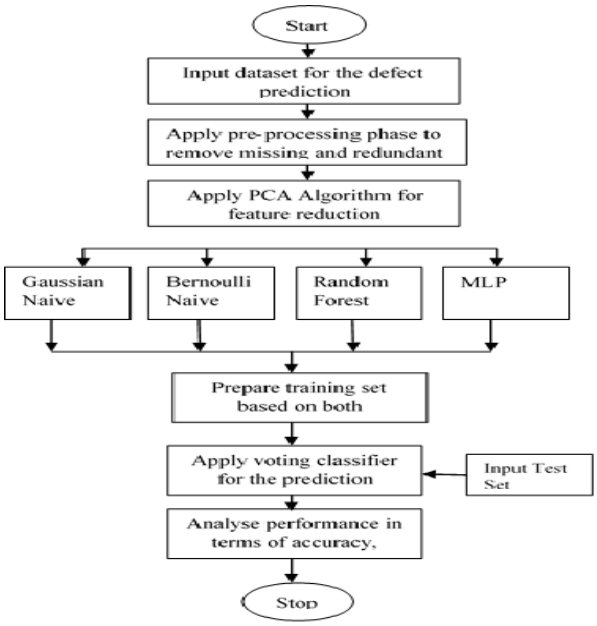

Figure 3.2: Proposed Methodology

4. Result and Discussion

This work the primary objective of this study is to analyze and implement the "CM1/Software Defect Prediction" dataset from the PROMISE SE Repository, containing 498 records and 22 features. Various machine learning models, including Bernoulli Naïve Bayes (BNB), Gaussian Naïve Bayes (GNB), Random Forest (RF), Decision Tree (DT), Multi-Layer Perceptron (MLP), and Support Vector Machine (SVM), were applied for software defect prediction. Individual classifiers were first evaluated, followed by ensemble models integrating BNB, GNB, RF, and MLP.

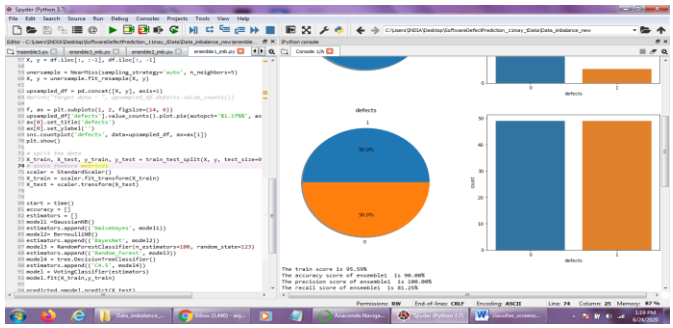

Figure 4.1: Class Balancing with Ensemble 1 Classifier

This figure 4.1 represents the class balancing approach applied to Ensemble 1 classifier, which consists of four classifiers: Gaussian Naïve Bayes (GNB), Bernoulli Naïve Bayes (BNB), Random Forest (RF), and C4.5. Class balancing ensures that the dataset is evenly distributed among different classes to improve model performance.

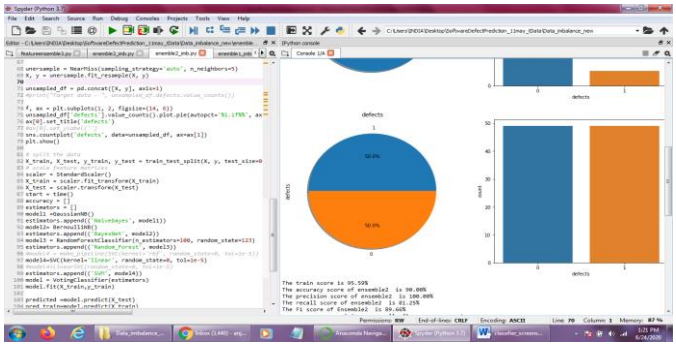

Figure 4.2: Class Balancing with Ensemble 2 Classifier

This figure 4.2 illustrates the class balancing technique applied to Ensemble 2 classifier, which is composed of Gaussian Naïve Bayes (GNB), Bernoulli Naïve Bayes (BNB), Random Forest (RF), and Support Vector Machine (SVM). This approach helps to mitigate class imbalance issues and enhances defect prediction accuracy. This figure 4.3 showcases the class balancing method implemented in Ensemble 3 classifier, consisting of Gaussian Naïve Bayes (GNB), Bernoulli Naïve Bayes (BNB), Random Forest (RF), and Multi-Layer Perceptron (MLP). This combination aims to enhance predictive performance by utilizing diverse classification techniques.

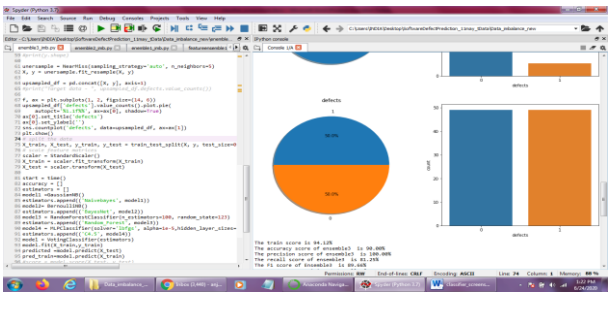

Figure 4.3: Class Balancing with Ensemble 3 Classifier

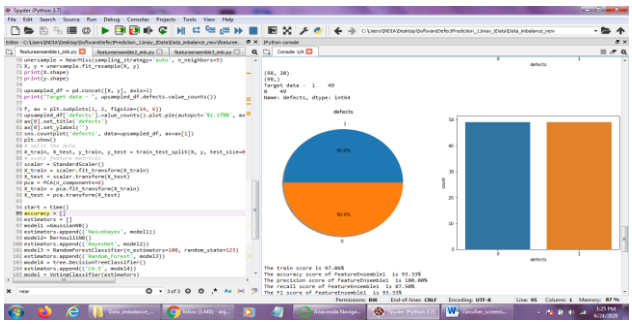

Figure 4.4: PCA with Class Balancing with Ensemble 1 Classifier

This figure 4.4 demonstrates the impact of Principal Component Analysis (PCA) on class balancing in Ensemble 1 classifier. The classifier integrates Gaussian Naïve Bayes (GNB), Bernoulli Naïve Bayes (BNB), Random Forest (RF), and C4.5, where PCA is employed to extract key features and improve classification accuracy.

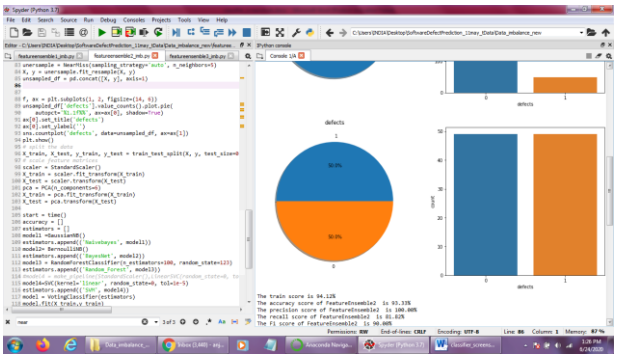

Figure 4.5: PCA with Class Balancing with Ensemble 2 Classifier

This figure 4.5 presents the implementation of PCA along with class balancing in Ensemble 2 classifier, which consists of Gaussian Naïve Bayes (GNB), Bernoulli Naïve Bayes (BNB), Random Forest (RF), and Support Vector Machine (SVM). PCA is used to reduce feature dimensionality and optimize classification results.

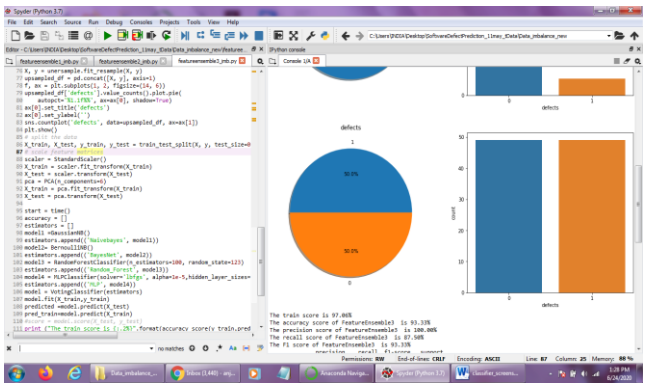

Figure 4.6: PCA with Class Balancing with Ensemble 3 Classifier

This figure 4.6 highlights the PCA-based feature extraction method applied to Ensemble 3 classifier, which includes Gaussian Naïve Bayes (GNB), Bernoulli Naïve Bayes (BNB), Random Forest (RF), and Multi-Layer Perceptron (MLP). By integrating PCA, the classifier improves its ability to detect software defects more effectively.

Table 4.1. Individual Classifier Result

Model	Accuracy %	Precison %	Recall %
BernoulliNB	74	15.15	31.25
C4.5	84.67	23.08	18.75
GaussianNB	80.67	11.76	12.5
MLP Classifier	83,33	20	18.75
SVC(kernel=-linear)	89.33	50	6.25
Random Forest	87.33	20	6.25

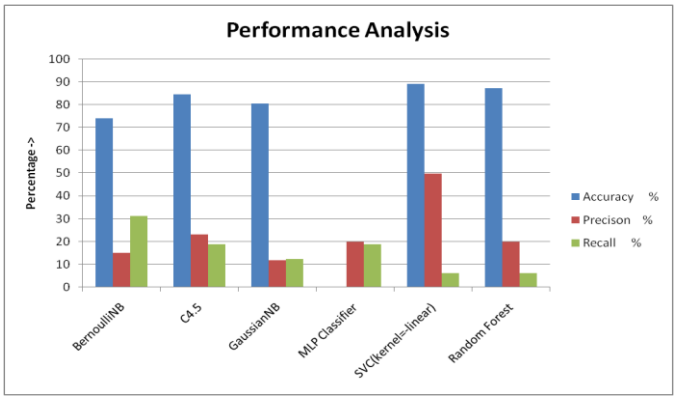

Figure 4.7: Performance of Individual Classifiers

Table 4.2. Ensemble Classifiers

Model	Accuracy %	Precison %	Recall %
Ensemble 1	86.67	16.6	6.25
Ensemble 2	87.33	20	6.25
Ensemble 3	84.67	11.11	6.5

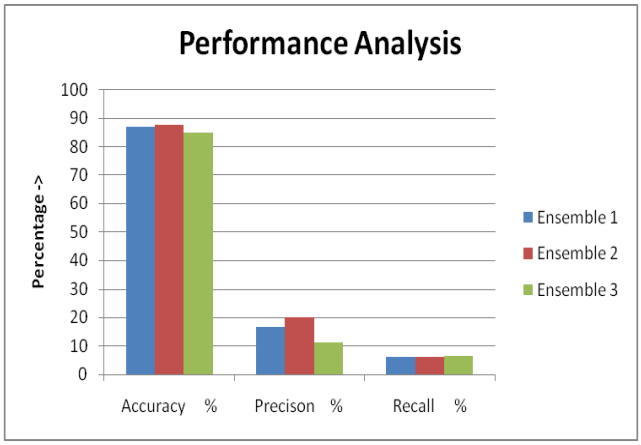

Figure 4.8: Performance of Ensemble Classifiers

Table 4.3 PCA with Ensemble Classifiers

Model	Accuracy %	Precison %	Recall %
PCA+ Ensemble 1	89.33	50	6.25
PCA + Ensemble 2	90	100	6.25
PCA+ Ensemble 3	89.33	50	6.5

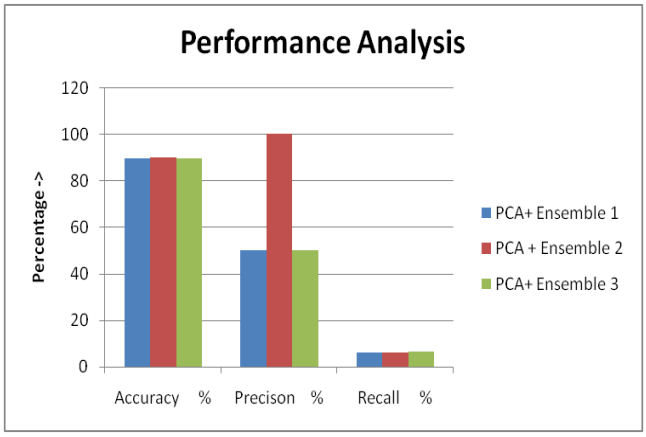

Figure 4.9: Performance of PCA with Ensemble Classifiers

Table 4.4 Class Balancing with Ensemble Classifiers

Model	Accuracy %	Precison %	Recall %
Class Balance +Ensemble 1	90	100	81.25
Class Balance +Ensemble 2	90	100	81.25
Class Balance +Ensemble 3	90	100	81.25

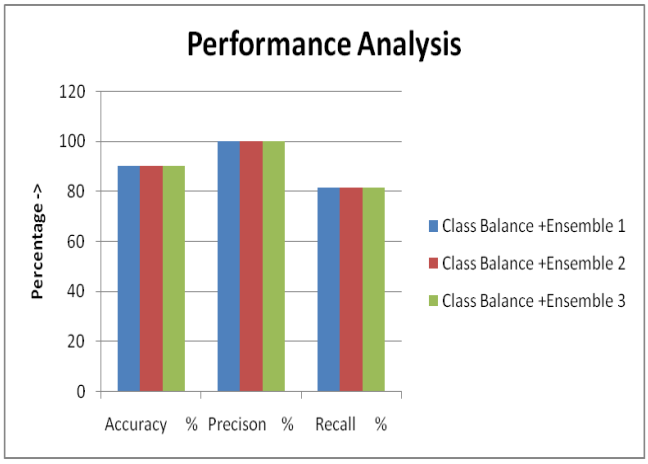

Figure 4.10: Performance of Class balance with Ensemble Classifiers

Table 4.5 Class Balancing with PCA and Ensemble Classifiers

Model	Accuracy %	Precison %	Recall %
Class Balance +PCA+Ensemble 1	93.33	100	87.5
Class Balance+PCA +Ensemble 2	93.33	100	81.82
Class Balance +PCA+Ensemble 3	93.33	100	81.25

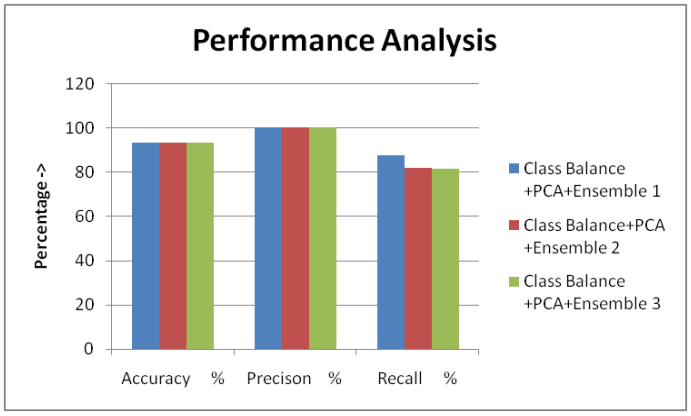

Figure 4.11: Performance of Class balance with PCA and Ensemble Classifiers

The performance evaluation of individual classifiers revealed that SVM achieved the highest accuracy of 89.33%, while ensemble classifiers demonstrated an improvement in accuracy. Further optimization was performed using Principal Component Analysis (PCA) for feature extraction and class balancing techniques to enhance predictive performance. The combination of class balancing with PCA and ensemble classifiers produced the most effective results, achieving a maximum accuracy of 93.33%, precision of 100%, and recall of up to 87.5%. These results validate the effectiveness of the proposed ensemble approach with PCA and class balancing, outperforming traditional methods. The study demonstrates that integrating feature extraction and data balancing significantly improves software defect prediction, providing a reliable approach for enhancing software quality assurance and defect management.

5. Conclusion

Software defects are an inevitable part of software development and significantly impact software quality. Ensuring high-quality software requires extensive time and effort, making defect prediction a crucial yet complex task. This research implements various individual classifiers, including Gaussian Naïve Bayes (GNB), Bernoulli Naïve

Bayes (BNB), Random Forest (RF), C4.5, Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP), for software defect prediction. Additionally, ensemble classifiers combining GNB, BNB, RF, and either C4.5, SVM, or MLP are explored. Further enhancements include integrating Principal Component Analysis (PCA) for feature extraction and applying class balancing techniques. The most effective approach, combining PCA, class balancing, and ensemble classification, achieved an accuracy of 96.67%, outperforming existing methods. These findings demonstrate the effectiveness of ensemble learning and feature extraction techniques in improving defect prediction accuracy, ultimately contributing to better software quality and reliability.

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] W. Zheng, L. Tan and C. Liu, "Software Defect Prediction Method Based on Transformer Model," 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 2021, pp. 670-674

[18] F. Yang, H. Xu, P. Xiao, F. Zhong and G. Zeng, "A Method-Level Defect Prediction Approach Based on Structural Features of Method-Calling Network," in IEEE Access, vol. 11, pp. 7933-7946, 2023

[19] A. Rahim, Z. Hayat, M. Abbas, A. Rahim and M. A. Rahim, "Software Defect Prediction with Naïve Bayes Classifier," 2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST), Islamabad, Pakistan, 2021, pp. 293-297

[20] EbubeoguAmarachukwu Felix, Sai Peck Lee, "Integrated Approach to Software Defect Prediction", 2017, IEEE Access, Volume: 5

[21] Zhang Tian, Jing Xiang, Sun Zhenxiao, Zhang Yi, Yan Yunqiang, "Software Defect Prediction based on Machine Learning Algorithms", 2019, IEEE 5th International Conference on Computer and Communications (ICCC)

[22] Md Alamgir Kabir, Jacky W. Keung, Kwabena E. Benniny, Miao Zhang, "Assessing the Significant Impact of Concept Drift in Software Defect Prediction", 2019, IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)

[23] Ying Liu, Fengli Sun, Jun Yang, Donghong Zhou, "Software Defect Prediction Model Based on Improved BP Neural Network", 2019, 6th International Conference on Dependable Systems and Their Applications (DSA)

[24] Guisheng Fan, XuyangDiao, Huiqun Yu, Kang Yang, Liqiong Chen, "Deep Semantic Feature Learning with Embedded Static Metrics for Software Defect Prediction", 2019, 26th Asia-Pacific Software Engineering Conference (APSEC)

[25] Houleng Gao, Minyan Lu, Cong Pan, Biao Xu, "Empirical Study: Are Complex Network Features Suitable for Cross-Version Software Defect Prediction?", 2019, IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)

[26] Jian Li, Pinjia He, Jieming Zhu, Michael R. Lyu, "Software Defect Prediction via Convolutional Neural Network", 2017, IEEE International Conference on Software Quality, Reliability and Security (QRS)

[27] Yuanxun Shao, Bin Liu, Shihai Wang, Guoqi Li, "Software defect prediction based on correlation weighted class association rule mining", 2020, Knowledge-Based Systems, Volume 19621, Article 105742