JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

A PCA BASED SOFTWARE FAULT PREDICTION MODEL USING ADRF

[1]Madhusmita Das, [2]Debasish Pradhan, [3]Suryakant Mahapatra
Asst. Professor, Asst. Professor, Asst. Professor
Department of Computer Science
NIIS Institute of Business Administration, Bhubaneswar, India

Abstract

The process of software testing is crucial in the development of software. Usually, errors committed by developers are corrected during the latter phases of the software development procedure., resulting in a greater impact from the defects. To avoid this, it is essential to anticipate defects early on during the software development phase. This proactive approach allows for the efficient allocation of testing resources. The process of defect prediction entails categorizing software modules into those likely to have defects and those not likely to have defects. The main goal of this study is to reduce the negative effects caused by two major issues faced in defect prediction, namely, the unequal distribution of data and the extensive number of factors in defect datasets. This research paper involves assessing multiple software metrics using feature selection methods like PCA, along with several machine learning classifiers including Adaboost, MLP, NB, J48, and Random Forest. The objective is to classify software modules as either prone to defects or not prone to defects. The suggested model utilizes a blend of Adaboost and random forest classifiers, incorporating PCA for dimension reduction. The experimental analysis relies on the publicly available NASA dataset. The findings indicate that the combination of Adaboost and random forest algorithms, coupled with PCA for the MC1 dataset, produced the most favourable outcomes compared to other datasets. The defect prediction accuracy reached an impressive 98.52% in contrast to the algorithms employed in the study.

Keyword: Software fault Prediction(SFP), AdaBoost, Random Forest, ADRF, Principal Component Analysis(PCA)

1. Introduction

Machine learning is altering how we deal with our day-to-day tasks. It permits us to generate self-functioning types of equipment, controls the intellectual assistants that are used in our day-to-day life, and assists to make sure the cloud runs perfectly. Artificial intelligence cannot function without machine learning. Machine learning is also used in the field of real applications in software testing. Now a days machine learning provides us with a superior way to do software testing. Software testing is the process of evaluating and verifying that what the software is supposed to do. Therefore, software testing is an essential part of the SDLC (Software Development Life Cycle). Originally, software testing was a manual process. After the development of AI & automation testing, it becomes more efficient, accurate, and faster. Now, Artificial intelligence (AI) is variously making over software testing. Due to these reasons, it simplifies test creation, reduces the need for test maintenance, and makes new ways to consider the result. The main advantages of testing include preventing bugs, minimizing development costs, and improving the rate of performance. Machine learning is a specific domain within the broader field of artificial intelligence, which encompasses the ability of machines to mimic human behaviour. Artificial intelligence encompasses systems that can execute intricate tasks, much like humans do when they tackle problems. ML in software testing is the best gadget for quality assurance (QA) since test automation is developed & implemented. Testing is a crucial aspect of the software development life cycle (SDLC). In the past, testing used to be a manual process that was monotonous, tiring, repetitive, and consumed a significant amount of time. It also caused delays in releasing and implementing software modifications. However, the advent of automation in testing has revolutionized this practice. Automation in testing involves the utilization of tools powered by machine learning to carry out testing tasks, manage test data, and produce reports for future analysis. Presently, numerous organizations employ artificial intelligence and machine learning in test automation to expedite their SDLC procedures.

2. Literature Review

Balaram et al. proposed a model that combines ensemble random forest with an adaptive synthetic sampling algorithm, achieving an accuracy of 0.85 [1]. Pravin Chandra et al. provide an overview of machine learning techniques for software fault prediction, including conventional methods. Their aim is to describe the issue of fault proneness [2]. Deepak Sharma et al. suggested a model that analyzes various machine learning techniques, highlighting important characteristics and presenting a vertical chevron list of decision tree techniques. They assess the efficiency of each technique [3]. Osama et al. proposed a model that employs ensemble random forests and deep learning for software fault prediction. They develop an algorithm that outperforms the CNN algorithm [4]. C. Laksmi Prabha et al. suggest that neural networks have the lowest failure rate compared to random forests in their study. However, dimensional classification achieves the highest detection rate. In cases of equal accuracy predictions, the failure rate parameter can be utilized to determine the correct outcome [5]. M. Farida Begam et al. utilize historical data to predict future software faults. They employ ensemble classifiers, specifically random forest, and

validate their models using K-Fold cross-validation technique. The results indicate that their models effectively work in all scenarios [6].

Awni Hammouri et al. suggested the evaluation process has been showed that ML algorithms like Naïve Bayes (NB), Artificial Neural Network (ANN) and Decision Tree (DT) can be used effectively with high accuracy rate[7]. Aimen Khalid et al. used K-means clustering techniques for software defect prediction and used the SVM and optimized SVM models which performed with the highest achieved accuracy 95% and 95.80% respectively [8]. Praman Deep Singh et al. has suggested a model in which PC1 data set with Linear Classifier algorithm produced an accuracy of 93.59% [9]. M. Farida Begam et.al. proposed a model by using SPSS tools with 10-fold cross validation extreme learning machine produced an accuracy of 87.5% [10].

3. Purposed Methodology

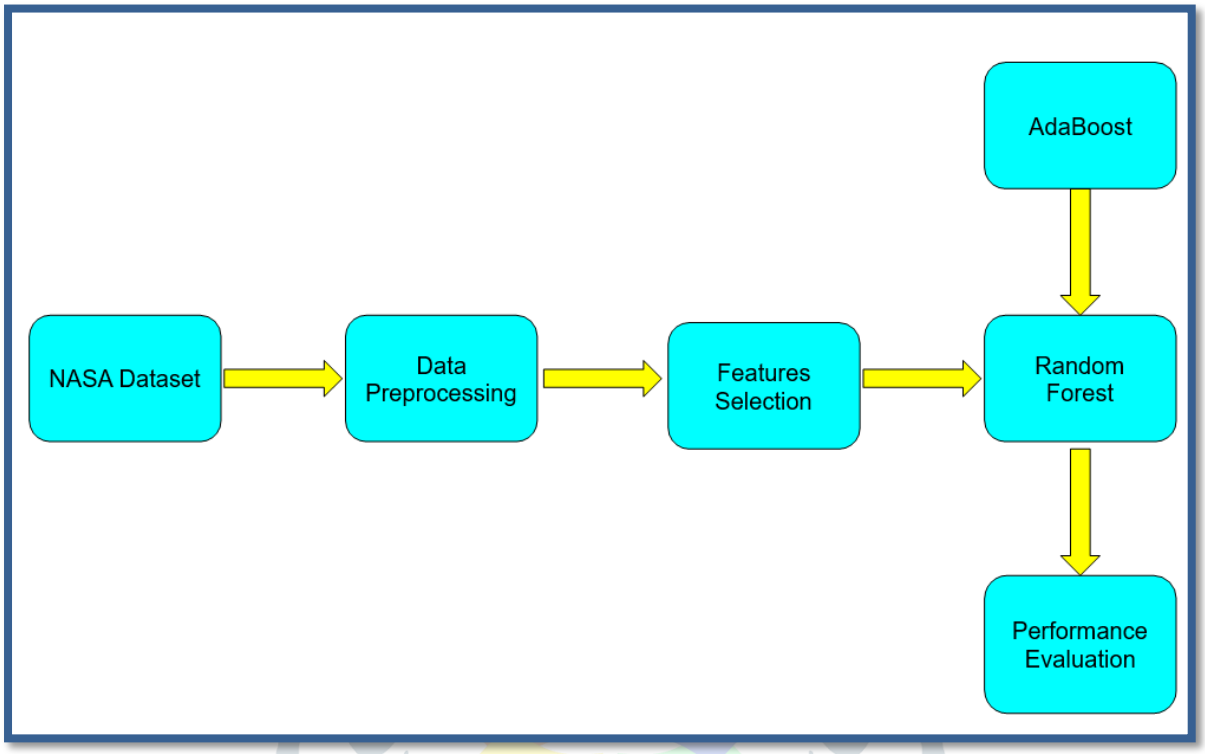

Fig.1. block diagram for planned system

3.1 Machine learning classifier

The goal of a machine learning classifier is to build a predictive model that can generalize from the training data to make accurate predictions or classifications on new, unseen data. The classifier learns from the features or attributes of the training data and their corresponding class labels. It then uses this knowledge to classify new instances based on their feature values.

3.1.1. Random forest

A potent machine learning technique known as Random Forest falls under the category of ensemble learning. By leveraging multiple decision trees, it achieves precise and resilient predictions. Its widespread adoption stems from its capacity to handle intricate datasets and address overfitting concerns. Essentially, Random Forest constructs a group of decision trees and amalgamates their predictions to yield the outcome, making it an ensemble learning approach. By aggregating the predictions of multiple trees, it can achieve better generalization and reduce the impact of individual decision trees' errors. To make predictions with Random Forest, each decision tree in the ensemble independently predicts the class label (for classification) or target value (for regression) of a new instance. The ultimate forecast is derived by combining the individual tree predictions using either majority voting (for classification) or averaging (for regression). Advantages and Limitations: It offers several advantages, including its robustness against over fitting, ability to handle high-dimensional data, and capability to capture complex relationships in the data. It is less sensitive to noise and outliers compared to individual decision trees. However, 's main limitations include its potential for increased computational complexity due to the construction and combination of multiple decision trees, as well as its reduced interpretability compared to a single decision tree. It is a versatile and widely used algorithm in various domains such as finance, healthcare, and natural language processing. It continues to be an active area of research, with on going developments to enhance its performance and address specific challenges in different applications.

3.1.2 AdaBoost

AdaBoost, also known as Adaptive Boosting, is an ensemble learning technique that merges the predictions of several weak classifiers to create a robust classifier. It is widely utilized for classification tasks and is recognized for its capability to enhance the performance of feeble learners. The algorithm prioritizes instances that were misclassified by previous classifiers, enabling subsequent classifiers to rectify their errors. By combining the predictions of multiple models, AdaBoost generates a final prediction. In this case, the models employed are weak classifiers or weak learners, which are modest models that exhibit slightly better performance than random guessing. Each instance in the training data is assigned weights by AdaBoost, with all instances initially having equal weights. Following each iteration, the weights of misclassified instances are increased, while the weights of correctly classified instances are decreased. This way, It focuses on the instances that are harder to classify, allowing subsequent weak learners to pay more attention to them. It offers several advantages, including its ability to handle complex datasets, improve weak learners' performance, and adapt to the characteristics of the data. It can be applied to various types of weak classifiers and is less prone to over fitting. However, It can be sensitive to noisy data and outliers, and it may be computationally expensive compared to individual weak classifiers. It remains a widely used and influential algorithm in the field of machine learning. Researchers continue to explore variations and enhancements to AdaBoost, as well as its applications in different domains, such as computer vision, natural language processing, and bioinformatics.

3.1.3 Naïve Bayes

A commonly used machine learning algorithm for classification tasks is Naive Bayes (NB). It operates on the principles of Bayes' theorem and assumes that features are independent of each other when conditioned on the class label. Despite its simple and "naive" assumption, NB has demonstrated effectiveness in a variety of real-world applications including text classification, spam filtering, sentiment analysis, and recommendation systems. Bayes' theorem forms the foundation of NB, which calculates the probability of an event based on prior knowledge or evidence. The theorem states that the probability of a hypothesis (class label) given the observed evidence (feature values) is proportionate to the probability of the evidence given the hypothesis, multiplied by the prior probability of the hypothesis. In mathematical terms:
P(hypothesis | evidence) = (P(evidence | hypothesis) * P(hypothesis)) / P(evidence)

Advantages and Limitations: It has several advantages, including its simplicity, scalability, and efficiency in both training and prediction. It can handle large feature spaces and is less prone to over fitting, making it suitable for high-dimensional datasets. However, its naive assumption

3.1.4 Multilayer Perception

The Multilayer Perceptron (MLP) is a fundamental type of artificial neural network (ANN) that consists of multiple layers of interconnected artificial neurons, or nodes. Each node in an MLP is typically a mathematical function that takes inputs, performs a calculation, and produces an output. MLPs are known as feed forward neural networks, meaning that information flows through the network in one direction, from the input layer to the output layer, without any loops or feedback connections. It's primary aim is to learn and model complex patterns or relationships in data. It has been widely used in various domains, including image and speech recognition, natural language processing, and financial analysis. MLPs are particularly effective in tasks where the input-output mapping is nonlinear and where there is a large amount of data available for training. MLPs have been the basis for many advanced neural network architectures, such as convolutional neural networks (CNNs) for image analysis and recurrent neural networks (RNNs) for sequential data processing. These architectures build upon the basic principles of MLPs and extend them to handle specific data types and tasks.

3.1.5 J48

It is a popular decision tree algorithm used for classification tasks in machine learning. It is an extension of the earlier ID3 algorithm and was developed by Ross Quinlan. It builds decision trees by recursively partitioning the training data based on the values of input features to make predictions. It is a decision tree algorithm, which represents decisions and their possible consequences as a tree-like structure. Each internal node in the tree corresponds to a test on a particular feature, and each leaf node represents a class label or a decision. The path from the root to a leaf node captures the sequence of decisions that lead to the final classification. Advantages and Limitations: It offers several advantages, including its interpretability, simplicity, and ability to handle both categorical and numerical attributes. It can handle noisy data and provides insights into the important features for classification. However, decision trees generated by J48 tend to be biased towards features with more levels or attributes with many values. Additionally, decision trees can be sensitive to small changes in the training data, leading to different tree structures. It remains a widely used and influential decision tree algorithm, providing a foundation for more advanced tree-based algorithms and ensemble methods. It is important to note that this introduction is based on the information available up until my knowledge cut off in September 2021. If there have been any significant developments or advancements in J48 since then, I may not be aware of them.

3.1.6 ADRF

ADRF (Adaptive Boosting with Random Forest) is a technique for ensemble learning that combines the advantages of two algorithms to enhance the accuracy of classification. It involves using the Random Forest algorithm as the weak learner within the AdaBoost framework. During each iteration of AdaBoost, the weak learner (Random Forest) is trained on a modified version of the training data, where the modifications are based on weights associated with each data point. These weights are updated by considering the errors in classification. The final classification result is obtained by aggregating the predictions of all weak learners through weighted voting, with the weights being determined by the accuracy of each weak learner. By combining the Random Forest and AdaBoost algorithms, this approach can offer improved accuracy in classification as well as robustness compared to using either algorithm independently. The Random Forest component of ADRF is effective in handling

complex and high-dimensional data, while AdaBoost adaptively concentrates on challenging instances to enhance overall performance. Through the integration of these techniques, ADRF maximizes the strengths of both algorithms, resulting in a more powerful ensemble classifier. Top of Form

3.2 Feature Selection

Feature selection is a crucial process in machine learning, which aims to select a subset of important features from a larger pool of available features. This selection is done to enhance the performance of the model, mitigate overfitting issues, and improve the interpretability of the results. There exist several techniques for feature selection, and I will provide an overview of a few commonly employed methods.

3.2.1 Principal Component Analysis (PCA):
PCA, which stands for Principal Component Analysis, is a widely used technique for reducing the dimensionality of a dataset. Its goal is to transform a dataset with many variables into a lower-dimensional space while retaining the most crucial information. To achieve this, PCA identifies the main directions, known as principal components, that capture the highest amount of variation in the data. The process typically begins by standardizing the dataset, where each feature is adjusted to have a mean of zero and a variance of one. After standardization, PCA calculates the covariance matrix of the standardized data. This matrix describes the relationships between different features and measures how they vary together. The subsequent step involves performing an Eigen decomposition of the covariance matrix. This decomposition breaks down the matrix into its eigenvectors and eigenvalues. The eigenvectors represent the principal components, and the corresponding eigenvalues indicate the amount of variation explained by each principal component. It is important to consider that the selection of a feature extraction method should depend on the specific characteristics of the dataset, the problem being addressed, and the requirements of the model. Experimentation with different techniques and assessing their impact on the model's performance is necessary to identify the most suitable approach.

4.**Experimental discussion and result analysis.**

The WEKA tool was used to evaluate the PCA+ADRF model on a computer system that had a core-i7, 3.4GHz processor, 16GB of RAM, and was running the Windows 11 operating system. To perform software fault prediction, NASA datasets such as JM1, PC5, PC4, MC1 and KC1 were employed. Further details regarding the parameters can be found in the subsequent description.

Sensitivity also known as the True Positive Rate, refers to the ratio of correctly predicted positive instances to the total number of actual positive instances. It can be calculated using the formula Sensitivity = True Positives (TP) divided by the sum of True Positives and False Negatives (TP + FN).

Specificity also known as the true negative rate, refers to the likelihood that a true negative result will be obtained from a negative test. Mathematically, it can be calculated by dividing the number of true negatives (TN) by the sum of true negatives and false positives (TN + FP).

Accuracy (ACC) is a metric that measures the proportion of correct predictions compared to the total number of predictions made. It is calculated by dividing the sum of true positives (TP) and true negatives (TN) by the sum of true positives, true negatives, false positives (FP), and false negatives (FN).

Precision (PPV) is a measure that indicates how accurate a model is in correctly classifying positive samples. It is calculated by dividing the number of true positive samples by the total number of positive samples classified, regardless of whether they were classified correctly or incorrectly. The formula for Precision is: PPV = True Positives (TP) divided by the sum of True Positives and False Positives (TP + FP).

F1 Score (F1) is a metric used to assess the performance of a particular model. It measures the model's effectiveness by considering both precision and recall through a harmonic average. Essentially, the F1 score provides a statistical evaluation of how well the model performs. It is calculated by considering true positives (TP), false positives (FP), and false negatives (FN) according to the formula: F1 = 2TP / (2TP + FP + FN). The resulting F1 score ranges from 0 (lowest) to 1 (highest), representing the average performance of a person based on precision and recall.

Matthews Correlation Coefficient (MCC) is a metric used in machine learning to assess the accuracy of binary classifications involving two classes. It considers true positives, true negatives, false positives, and false negatives, making it a well-balanced measure suitable for cases where there is a significant difference in class sizes. Essentially, the MCC acts as a correlation coefficient between the predicted and observed binary classifications, providing a value that ranges from -1 to +1. A coefficient of +1 indicates a perfect prediction, 0 suggests an average random prediction, and -1 signifies an inverse prediction.

MCC= TP*TN-FP*FN/sqrt((TP+FP) *(TP+FN) *(TN+FP) *(TN+FN))

Were

TP (True Positive)- Correctly classified positive cases

TN (True Negative)-Correctly classified negative cases

FN (False Negative)-Incorrectly classified positive cases

FP (False Positive)-Incorrectly classified negative cases.

Fig.1 represents the Block diagram for planned System. In Table.1 the performance evaluation is carried out without using any feature selection methods. In this table we have used 6 different machine learning classifiers (AB, RF, MLP, J48, NB, ADRF) on different data sets (JM1,PC4,PC5,KC1,MC1). The accuracy out perform in case of ADRF classifier on MC1 with a value of 6 cross validations which is 98.14%. The same operation with PCA is carried out on table.2, PCA+ADRF produced a highest accuracy of 98.52% on MC1 dataset with 9 fold cross validation. A comparative study has been done in graphical form between classifier without feature selection and accuracy which is represented in fig.2 where as the same operation is performed with PCA in fig.3 for JM1 dataset. Similarly the above operation is performed on MC1 dataset (fig.4,fig.5), PC5 dataset(fig.6,fig.7), KC1 dataset (fig.7, fig.8) and PC4 dataset (fig.9,fig.10) respectively. In Table.3 a comparison has been

made between some existing methodology and purposed system which conclude that the proposed system PCA+ADRF outperformed in accuracy.

Table 1. Performance Evaluation without Feature Selection

Data Set	Classify	CV	Sensitivity	Specificity	Precision	F1	MCC	Accuracy
JM1	AB	5	78.51	0	1	87.96	0	78.51
JM1	RF	5	81.14	56.43	96.12	88	23.33	79.41
JM1	NB	8	80.97	48.91	94.65	87.28	19.99	78.33
JM1	J48	7	81.50	43.49	91.41	86.17	19.72	76.96
JM1	MLP	6	79.78	57.36	98.15	88.02	16.4	79.02
JM1	ADRF	7	66.78	80.45	40.98	50.79	39.62	79.50
MC1	AB	6	97.74	1	1	98.85	14.58	97.74
MC1	RF	9	97.08	72.73	99.85	98.95	34.94	97.94
MC1	NB	9	98.11	63.35	90.9	94.37	8.7	89.40
MC1	J48	5	97.93	41.67	99.64	98.77	20.4	97.59
MC1	MLP	8	97.82	27.27	99.59	98.7	12.38	97.43
MC1	**ADRF**	**6**	**97.74**	**1**	**1**	**98.85**	**14.58**	**98.14**
PC5	AB	6	52.7	77.74	33.12	40.68	25.78	73.41
PC5	RF	7	66.78	80.45	40.98	50.79	39.62	78.14
PC5	NB	9	43.78	83.27	63.48	51.82	29.65	67.50
PC5	J48	9	56.1	82.7	53.72	54.88	38.27	75.69
PC5	MLP	8	52.55	78.52	37.15	43.53	27.54	73.47
PC5	ADRF	7	66.78	80.45	40.98	50.79	39.62	78.62
KC1	AB	9	61.84	75.88	14.97	24.1	20.95	74.98
KC1	RF	8	62.96	79.24	32.48	42.86	32.85	77.01
KC1	NB	7	49.75	78.15	31.53	38.6	23.63	73.37
KC1	J48	10	53.37	79.18	35.35	42.53	28.06	74.64
KC1	MLP	10	61.21	77.23	22.61	33.02	25.88	75.66
KC1	ADRF	8	62.96	79.24	32.48	42.86	32.85	77.30
PC4	AB	10	89.47	69.01	98.02	93.55	38.77	88.34
PC4	RF	9	90.45	74.7	98.11	94.12	46.46	89.43
PC4	NB	10	90.28	50.26	95.41	92.77	37.57	87.18
PC4	J48	10	92.48	58.6	94.14	93.3	48.54	88.34
PC4	MLP	10	92.75	60.9	94.5	93.62	50.84	88.89
PC4	ADRF	9	90.45	74.17	98.12	94.14	46.48	89.50

Table 2. Performance Evaluation with PCA

Data Set	Classify	No of Features	CV	Sensitivity	Specificity	Precision	F1	MCC	Accuracy
JM1	PCA+AB	16	5	78.51	0	100	87.96	0	78.53
JM1	PCA+RF	16	5	81.14	56.43	96.12	88.00	23.33	79.43
JM1	PCA+NB	16	8	80.97	48.91	94.65	87.28	19.99	78.35
JM1	PCA+J48	16	8	80.95	40.18	91.23	85.78	16.42	85.78
JM1	PCA+MLP	16	6	79.78	57.36	98.51	88.02	16.40	79.05
JM1	PCA+ADRF	16	5	81.15	56.48	96.14	88.12	23.34	86.11
MC1	PCA+AB	9	5	97.69	0.00	1.00	98.83	0.00	97.81
MC1	PCA+RF	9	9	98.08	72.73	99.85	98.95	34.94	97.98
MC1	PCA+NB	9	9	98.11	6.35	90.89	94.36	8.70	89.45
MC1	PCA+J48	9	5	97.93	41.67	99.64	98.77	20.40	97.63

MC1	PCA+MLP	9	8	97.82	27.27	99.59	98.70	12.38	97.48
MC1	**PCA+ADRF**	**9**	**9**	**98.55**	**72.81**	**99.95**	**98.96**	**34.97**	**98.52**
PC5	PCA+AB	16	7	57.60	77.62	30.57	39.94	27.85	74.69
PC5	PCA+RF	16	7	66.78	80.45	40.98	50.79	39.62	78.21
PC5	PCA+NB	16	9	60.37	75.95	21.02	31.18	23.94	74.46
PC5	PCA+J48	16	9	56.10	82.70	53.72	54.88	38.27	75.72
PC5	PCA+MLP	16	10	55.89	78.43	35.24	43.23	29.10	74.52
PC5	PCA+ADRF	16	7	66.81	80.46	40.98	50.79	39.61	78.99
KC1	PCA+AB	16	9	61.84	75.88	14.97	24.10	20.95	75.14
KC1	PCA+RF	16	8	62.96	79.24	32.48	42.86	32.85	77.01
KC1	PCA+NB	16	7	49.75	78.15	31.53	38.60	23.63	73.37
KC1	PCA+J48	16	10	53.37	79.18	35.35	42.53	28.06	75.11
KC1	PCA+MLP	16	8	61.61	77.12	21.97	32.39	25.68	75.79
KC1	PCA+ADRF	16	8	62.96	79.35	32.51	42.86	32.87	77.81
PC4	PCA+AB	10	10	89.47	69.01	98.02	93.55	38.77	88.41
PC4	PCA+RF	10	8	91.11	75.53	97.93	94.40	50.35	89.46
PC4	PCA+NB	10	10	90.28	55.26	95.41	92.77	37.57	87.25
PC4	PCA+J48	10	10	92.48	58.60	94.14	93.30	48.54	88.41
PC4	PCA+MLP	10	10	92.75	60.90	94.50	93.62	50.84	88.95
PC4	PCA+ADRF	10	8	91.21	75.55	97.93	94.40	50.45	89.59

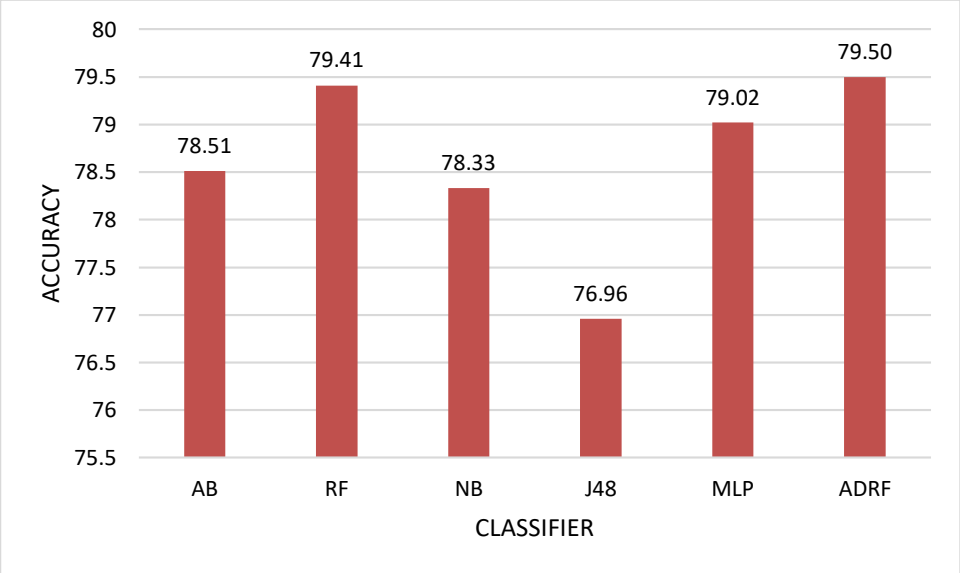

Fig.2: Accuracy for JM1 dataset without Feature Selection

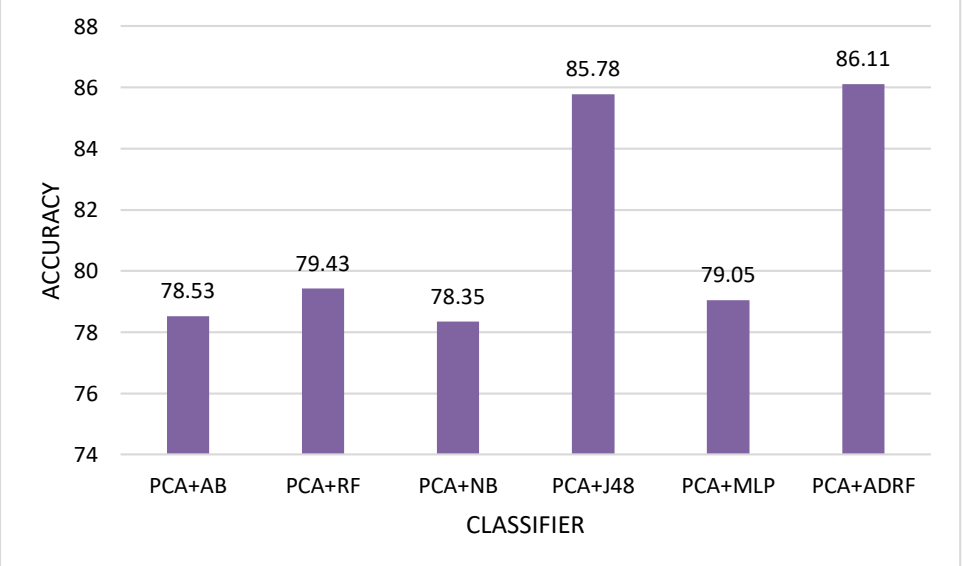

Fig.3: Accuracy for JM1dataset with feature selection

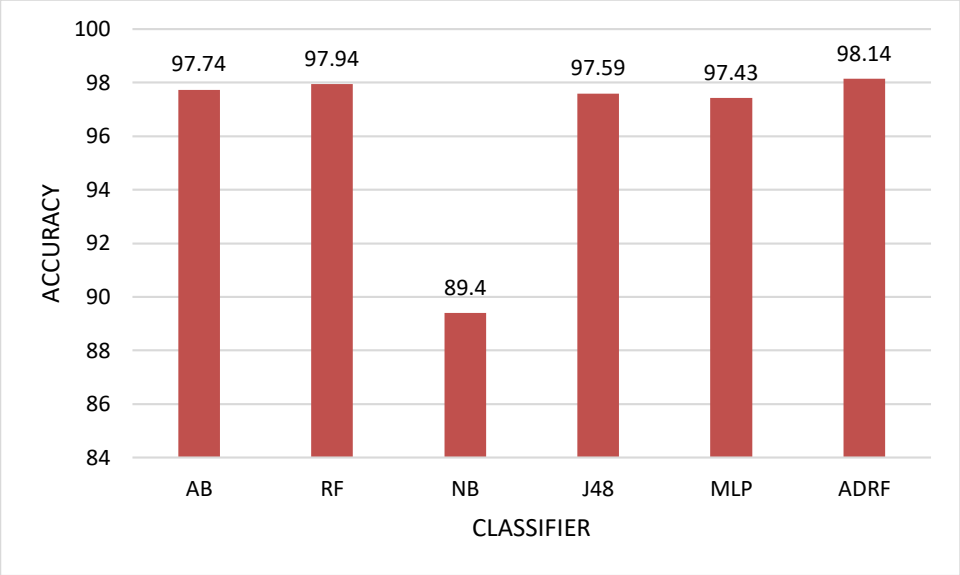

Fig.4: Accuracy for MC1 dataset without Feature Selection

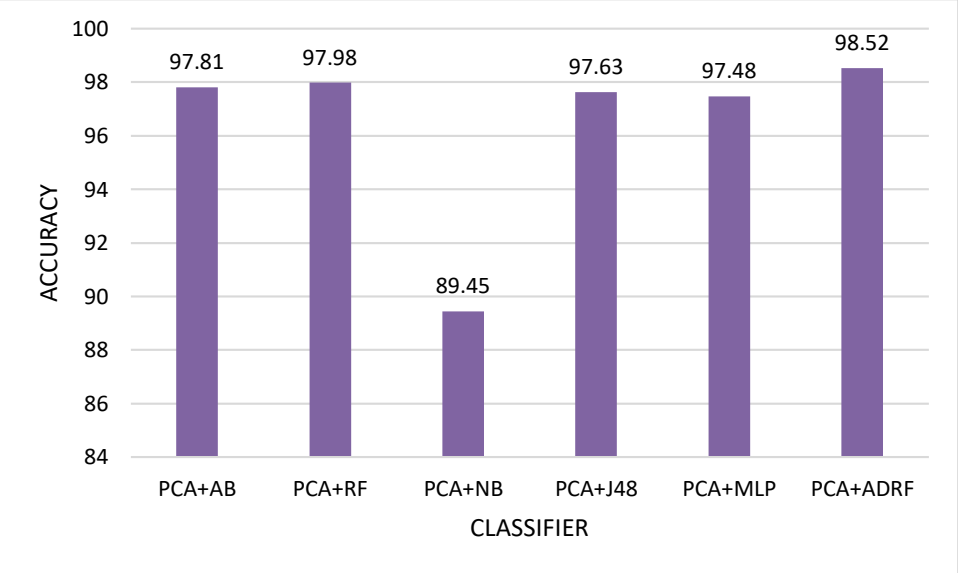

Fig.5 : Accuracy for MC1 dataset with feature selection

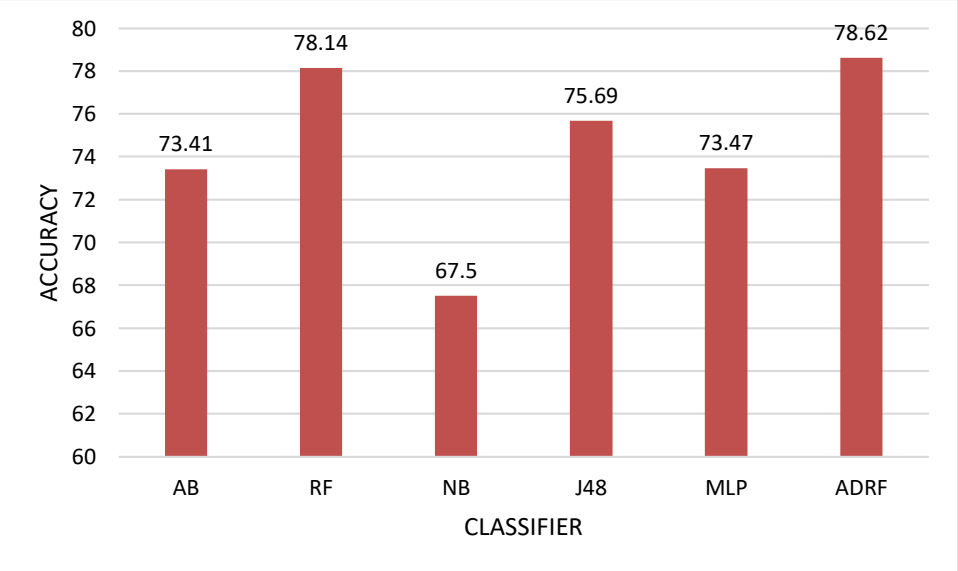

Fig.4: Accuracy for PC5 dataset without Feature Selection

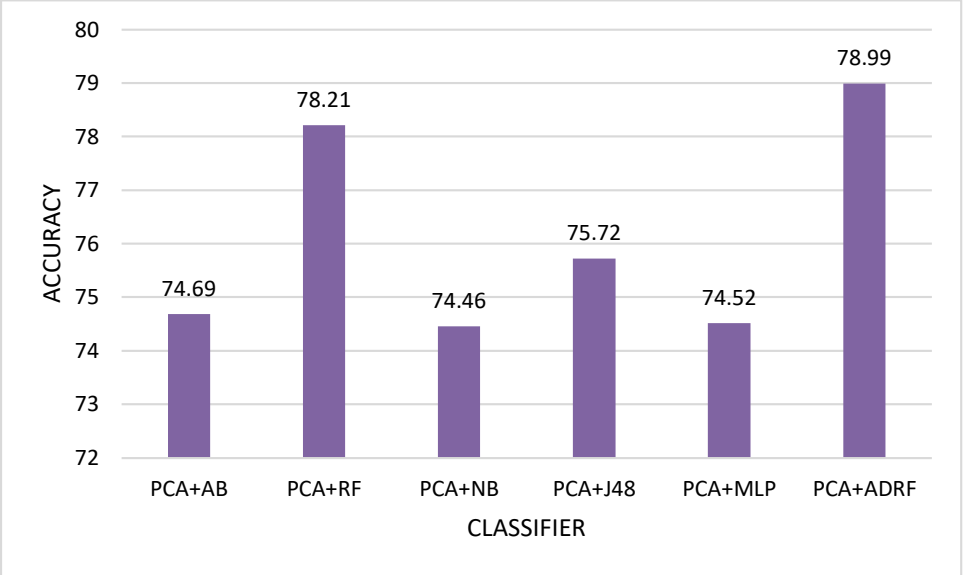

Fig.5: Accuracy for PC5 dataset with feature selection

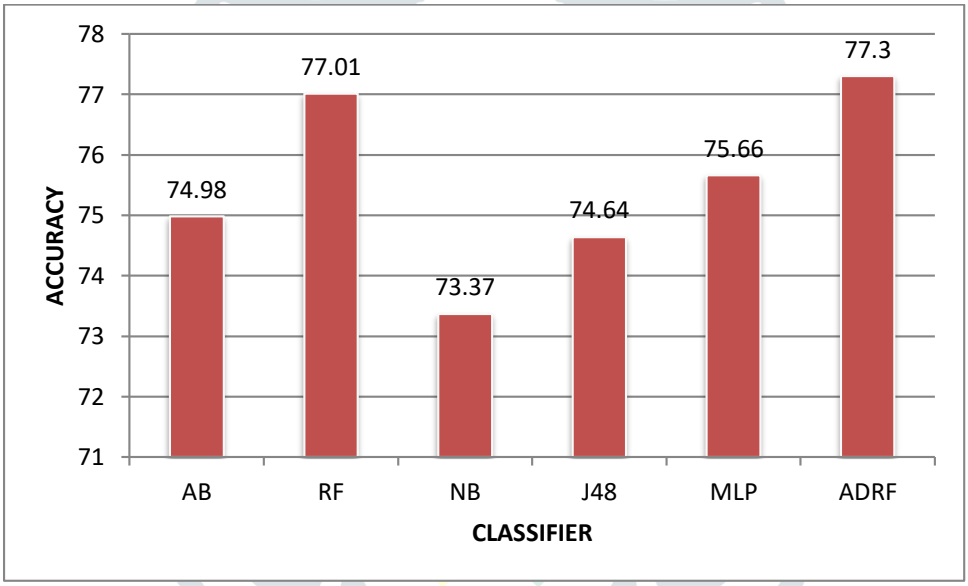

Fig.6: Accuracy for KC1 dataset without Feature Selection

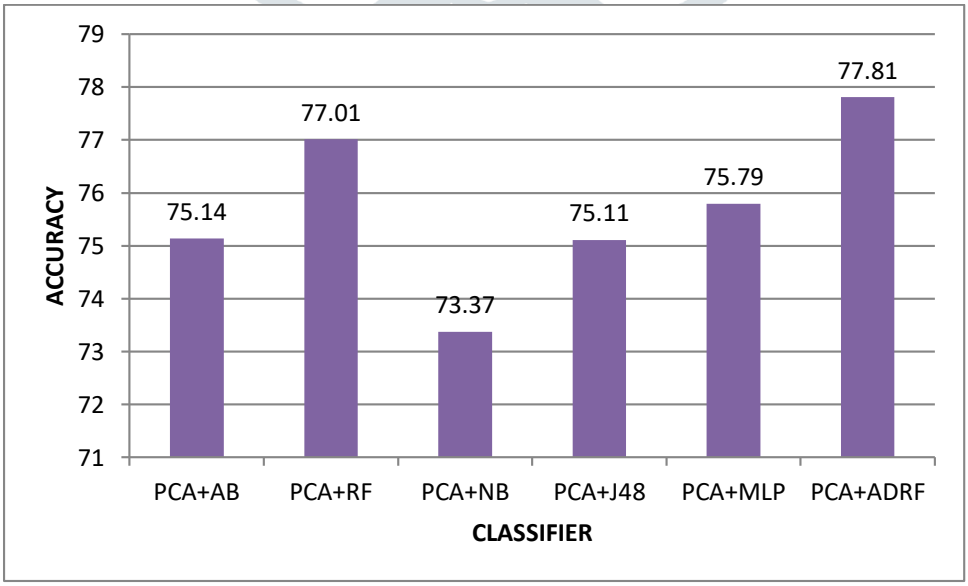

Fig.7: Accuracy for KC1 dataset with feature selection

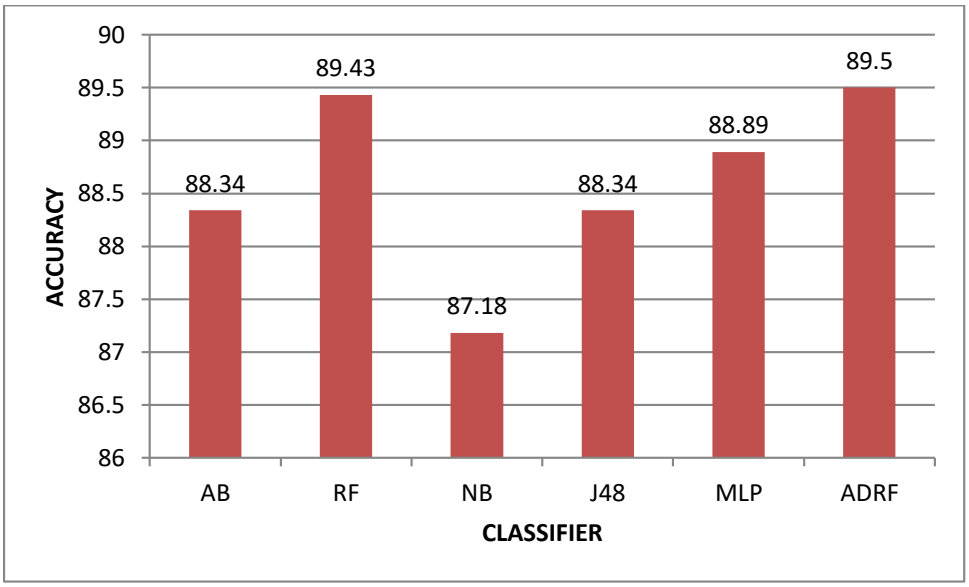

Fig.10: Accuracy for PC4 dataset without Feature Selection

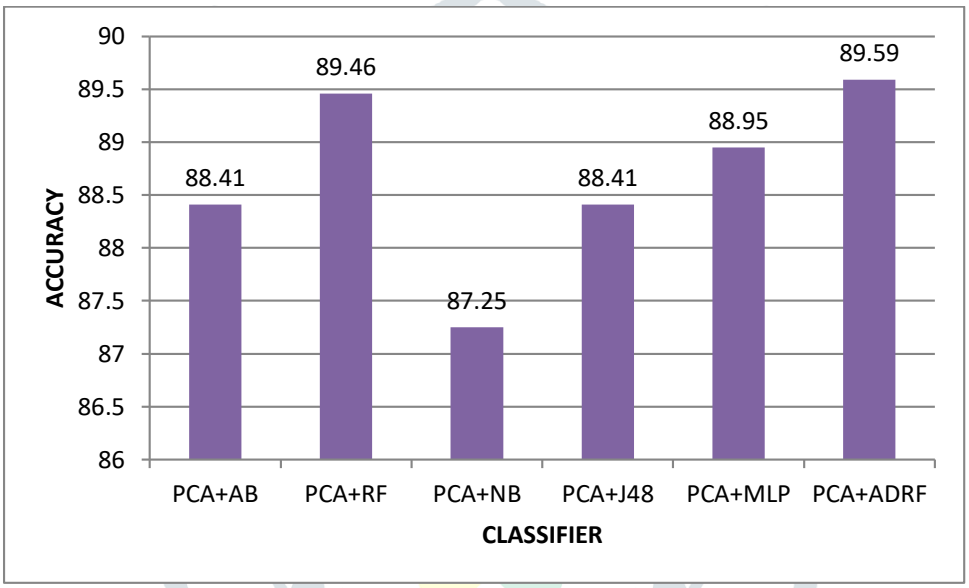

Fig.11: Accuracy for PC4 dataset with feature selection

Table.3: Performance Comparison of Existing Methodology

References	**Methodologies**	**Accuracy(%)**
Balaram et al. [1]	Random Forest+ ASSA	85.00
Aimen Khalid et al. [8]	KNN +SVM	95.80
Praman Deep Singh et al.[9]	Linear Regression	93.59
M. Farida Begam et.al.[10]	ELM	87.5
R. Jayanthi et al. [11]	PCA+ANN	93.64
Huihua Lu et al. [20]	PCA+RF	95.18
Proposed Methodology	**PCA+ADRF**	**98.52**

5. Conclusion

In our recent study, we have created a precise approach called PCA+ADRF to detect software defects within individual modules. This method involves using PCA to reduce the complexity of the features. Then, we employ a combination of two algorithms, AD and RF, known as ADRF, to develop a highly accurate and automated classifier. Through empirical findings, we have demonstrated that the PCA+ADRF method outperforms all other existing techniques across six different datasets. Specifically, when applied to the MC1 dataset, our approach achieves an outstanding accuracy of 98.52%. Researchers are still interested in developing defect prediction techniques for software systems that are more precise and efficient. Our study contributes to early problem detection, which saves time and reduces the overall cost of software projects. Further research can focus on developing various hybrid models that can predict software system flaws with improved accuracy and fewer errors.

References

[1] Balaram, A., Vasundra, S. Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm. Autom Software Eng 29, 6 (2022). https://doi.org/10.1007/s10515-021-00311-z

[2] Sharma, Deepak & Chandra, Pravin. (2018). Software Fault Prediction Using Machine-Learning Techniques. 10.1007/978-981-10-5547-8_56.

[3] Alqasem, Osama & Akour, Mohammed. (2019). Software Fault Prediction Using Deep Learning Algorithms. International Journal of Open Source Software and Processes. 10. 1-19. 10.4018/IJOSSP.2019100101.

[5] C. L. Prabha and N. Shivakumar, "Software Defect Prediction Using Machine Learning Techniques," *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, Tirunelveli, India, 2020, pp. 728-733, doi: 10.1109/ICOEI48184.2020.9142909.

[6] Immaculate, S. & Begam, M. & Floramary, M.. (2019). Software Bug Prediction Using Supervised Machine Learning Algorithms. 1-7. 10.1109/IconDSC.2019.8816965.

[7] Hammouri, Awni & Hammad, Mustafa & Alnabhan, Mohammad & Alsarayrah, Fatima. (2018). Software Bug Prediction using Machine Learning Approach. International Journal of Advanced Computer Science and Applications. 9. 10.14569/IJACSA.2018.090212.

[8] Khalid, Aimen & Badshah, Gran & Ayub, Nasir & Shiraz, Muhammad & Ghouse, Mohamed. (2023). Software Defect Prediction Analysis Using Machine Learning Techniques. Sustainability. 15. 5517. 10.3390/su15065517.

[9] Singh, Praman & Chug, Anuradha. (2017). Software defect prediction analysis using machine learning algorithms. 775-781. 10.1109/CONFLUENCE.2017.7943255.

[10] Farida Begam, M. & Floramary, M.. (2019). Software Bug Prediction Using Supervised Machine Learning Algorithms. 1-7. 10.1109/IconDSC.2019.8816965.

[11] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifiers", *Cluster Computing*, vol. 22, no. 1, pp. 77-88, 2019.

[12] E.A. Felix and S.P. Lee, "Integrated approach to software defect prediction", IEEE Access, vol. 5, pp. 21524-21547, 2017.

[13] T. Wang, Z. Zhang, X. Jing and L. Zhang, "Multiple kernel ensemble learning for software defect prediction", *Autom. Software. Eng.*, vol. 23, pp. 569-590, 2015.

[14] Z. Xu, J. Xuan, J. Liu and X. Cui, "MICHAC: defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering", *2016 IEEE 23rd International Conference on Software Analysis Evolution and Reengineering (SANER)*, pp. 370-381, 2016.

[15] D. Ryu and J. Baik, "Effective multi-objective naïve Bayes learning for cross-project defect prediction", Appl. Soft Comput., vol. 49, no. 1062, 2016.

[16] C. Shan, B. Chen, C. Hu, J. Xue and N. Li, "Software defect prediction model based on LLE and SVM", *Proceedings of the Communications Security Conference (CSC '14)*, pp. 1-5, 2014.

[17] K. Han, J.-H. Cao, S.-H. Chen and W.-W. Liu, "A software reliability prediction method based on software development process", *Quality Reliability Risk Maintenance and Safety Engineering (QR2MSE) 2013 International Conference on*, pp. 280-283, 2013.

[18] S. Parthipan, S. Senthil Velan and C. Babu, "Design level metrics to measure the complexity across versions of ao software", *Advanced Communication Control and Computing Technologies (ICACCCT) 2014 International Conference on*, pp. 1708-1714, 2014.

[19] A.M. Bautista, T.S. Feliu, J. Mejia, M. Munoz, A. Rocha and J. Calvo-Manzano, "Defect prediction in software repositories with artificial neural networks" in Trends and Applications in Software Engineering. Advances in Intelligent Systems and Computing, Cham:Springer, vol. 405, 2016.

[20] H. Lu, B. Cukic and M. Culp, "Software defect prediction using semisupervised learning with dimension reduction", *Automated Software Engineering (ASE) 2012 Proceedings of the 27th IEEE/ACM International Conference on*, pp. 314-317, 2012.

[21] M. Alenezi, S. Banitaan and Q. Obeidat, "Fault-proneness of open source systems: An empirical analysis", *Synapse*, vol. 1, pp. 256, 2014.

[22] Ahmed Iqbal et al., "Performance Analysis of Machine Learning Techniques on Software Defect Prediction using NASA Datasets", *Int. J. Adv. Comput. Sci. Appl*, vol. 10, no. 5, 2019.

[23] Malkit Singh and Dalwinder Singh Salaria, "Software defect prediction tool based on neural network", *International Journal of Computer Applications*, vol. 70, no. 22, 2013.

[24] I. A and A. Saha, "Software Defect Prediction: A Comparison Between Artificial Neural Network and Support Vector Machine", *Adv. Comput. Commun. Technol*, pp. 51-61, 2017.

[25] Abdullah Alsaeedi and Mohammad Zubair Khan, "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study", *Journal of Software Engineering and Applications*, vol. 12, no. 5, pp. 85-100, 2019.

[26] Shuo Wang and Xin Yao, "Using class imbalance learning for software defect prediction", *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434-443, 2013.

[27] Chen, Lq., Wang, C. & Song, Sl. Software defect prediction based on nested-stacking and heterogeneous feature selection. Complex Intell. Syst. 8, 3333–3348 (2022). https://doi.org/10.1007/s40747-022-00676-y

[28] Bowes D, Hall T, Petrić J (2018) Software defect prediction: do different classifiers find the same defects? Software Qual J 26(2):525–552

[29] Khuat T T, Le M H (2020) Evaluation of sampling-based ensembles of classifiers on imbalanced data for software defect prediction problems.SN Computer Science 1(2):1-16

[30] Bejjanki KK, Gyani J, Gugulothu N (2020) Class imbalance reduction (CIR): a novel approach to software defect prediction in the presence of class imbalance. Symmetry 2(3):407