International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

Improving model performance for software defect detection and prediction using ensemble method and cross validation techniques

Emmanuel U. Oyo-Ita [1,*], Emmanuel A. Edim [2], Anthony Otiko [3] and Darlington E. Izuki [4]

[1] *Department of Computer Science, University of Cross River State, Nigeria.*
[2] *Department of Computer Science, University of Calabar, Nigeria.*
[3] *Department of Computer Science, University of Cross River State, Nigeria.*
[4] *Directorate of Information & Communication Technology, University of Cross River State Nigeria.*

International Journal of Science and Research Archive, 2024, 12(02), 2363–2373

Publication history: Received on 07 July 2024; revised on 16 August 2024; accepted on 19 August 2024

Article DOI: https://doi.org/10.30574/ijsra.2024.12.2.1518

Abstract

Software defects and quality assurance are crucial aspects of software development that should be considered during the software development cycle. To ensure high-quality software, it is essential to have a robust quality assurance process in place. System reliability and quality are very key components that must be considered during software development, and this can only be achieved when software undergoes a thorough test process for errors, anomalies, defects, omissions, and bugs. Early software defect prediction and detection play an essential role in ensuring the reliability and quality of software systems, ensuring that software companies discover errors or defects early enough and allocate more resources to defect-prone modules. This study proposes the development of an enhanced classifier model for software defect prediction and detection. The aim is to harness the collective intelligence of selected base classifiers like Support Vector Machine, Logistic regression, Decision Trees, Random Forest, AdaBoost, Gradient Boosting, K-Nearest Neighbor, GaussianNB, and Multi-Layer Perception to improve accuracy, robustness, and generalization in identifying potential defects using a soft voting ensemble technique. The ensemble model leveraged the confidence probability of the soft voting technique and the generalization advantage of cross-validation leading to a more robust and dynamic model. The performance of the model with existing classifiers was evaluated using accuracy, F1 score, Precision, and area under the ROC curve (ROC- AUC) as the evaluation metrics. The results of the experiment revealed that the Proposed Classifier produced an overall Accuracy rate of 93%, and ROC AUC of 98%. The results demonstrate the effectiveness of our enhanced ensemble classifier in software defect detection and prediction. By harnessing the strengths of diverse base classifiers, our approach provides a robust and adaptive solution to the challenges of early detection and mitigating defects in software systems. This research contributes to the advancement of reliable software development practices and lays the foundation for future enhancements in ensemble-based defect detection methodologies.

Keywords: Base Classifier; Cross-Validation; Ensemble; Machine learning; Software Defect; Soft Voting

1. Introduction

Software defects and quality assurance are crucial aspects of software development that should be considered during the software development cycle as they ensure the release of reliable, efficient, and high-performing software products [1]. To ensure high-quality software, it is essential to have a robust quality assurance process in place. These processes involve code reviews, testing, and inspections to identify and eliminate defects before the software is released to the end-users [2].

* Corresponding author: Emmanuel U. Oyo-Ita

In a study by Alsawalqah et al [3], they acknowledged software defect prediction as a reliable approach towards early and timely identification of error-prone modules of software before the software undergoes the testing process and this is capable of helping software developers allocate appropriate resources to the affected module. One of the objectives of this research is to develop a software defect prediction and detection model. This study looks to achieve this through the deployment of machine learning models to help detect and predict software defects.

Software quality is an error-free product that produces predictable results and can be delivered within time and cost constraints [4]. As a result, it is very important to have appropriate approaches to develop high-quality software that can meet the increasing needs in today's business world. Past studies suggest that no single defect detection technology can solve all types of defect detection problems.

According to Saheed et al.[5] most of the research on machine learning methods for identifying software defects performed poorly in terms of prediction accuracy and other performance indicators. Many of these studies just looked at accuracy, which is insufficient to gauge how well SDP is performing. As a result, they suggested a machine-learning model for SDP that has seven ensembles.

So, this study focuses on enhancing the effectiveness and efficiency of the defect detection model's accuracy.

However, achieving high accuracy in defect detection models with less misclassified data remains a challenging task. Ensemble methods and Cross-validation Techniques when combined have shown promising results in improving a model's performance. This article explores the combination of ensemble methods and cross-validation techniques to enhance software defect detection accuracy.

Commonly used strategies for model enhancements include Ensemble methods, cross-validation, Feature engineering, Transfer learning, and more. This research seeks to combine two of these strategies to produce a more efficient classification model for software defect detection.

Regression and classification strategies are mostly suitable for defect detection. The objective of regression techniques is to predict the number of software defects [6] while the classification approach aims to decide whether a software module is faulty or not. Classification models can be trained from the defect data of the previous version of the same software. The trained models can then be used to predict further potential software defects.

1.1. Ensemble Techniques

In the realm of software defect prediction and detection, ensemble approaches have attracted a lot of interest. Numerous studies, such as that of Matloob et al. [7], have emphasized the importance of considering multiple models and ensemble approaches in bug prediction tasks. These studies also demonstrate that by combining the predictions of diverse models, an ensemble can improve bug prediction performance by leveraging their strengths and mitigating their weaknesses.

The efficacy of an ensemble classifier over single classifiers was established by Gupta et al. [8] in their research on the usefulness of ensemble classifiers over cutting-edge machine learning classifiers for predicting software errors in software modules. They used an ensemble classifier to test against cutting-edge classifiers such as J4, Random forests, Decision tables, and Naïve Bayes. Their findings showed that the ensemble classifier produced superior outcomes than solo classifiers.

One of the key advantages of ensemble learning in software defect prediction is its ability to mitigate overfitting. Overfitting occurs when a model learns the noise in the training data, resulting in poor generalization to unseen data [9]. Ensemble learning, by combining diverse models, can reduce overfitting by capturing different patterns in the data. Each base model may overfit to a certain extent, but the combination tends to produce a more generalized and robust prediction [10]. Similarly, Marçal and Garcia [11] in their experiment using ensemble techniques highlighted the potential of this technique in bug prediction as well as ultimately enhancing software reliability.

1.2. Cross Validation Technique

Cross-validation is a statistical technique used to assess and compare learning algorithms by dividing data into two segments: one for training the model and the other for validating it. In traditional cross-validation, the training and validation sets must overlap in subsequent rounds to ensure that every data point is equally verified. The most basic type of cross-validation is k-fold cross-validation. Other types of cross-validation include special cases of k-fold cross-validation or multiple rounds of k-fold cross-validation [12].

The cross-validation technique was deployed in this study as a model evaluation technique to train and assess the proposed model's performance on unseen data. This technique will improve the model's generalization abilities and further improve the model's accuracy and robustness.

2. Evaluation Measures for Software Bugs Prediction

In this section, we will discuss some measurements for software defect prediction such as

accuracy, precision, recall, and F1-score, to gain insights into the classifier's effectiveness. This will also form part of the evaluation metrics deployed in this study to evaluate the proposed classifier model. [13] in their systematic review of the ensemble learning approach for software defect prediction established that the AUC, accuracy, F-measure, Recall, and precision were mostly used to measure the prediction performance of models. This explains the adoption of the under-listed measurement parameters.

- Accuracy: The accuracy of the enhanced classifier was determined by calculating the ratio of correctly predicted instances to the total instances in the testing dataset. This metric provided an overall measure of correctness in the predictions.
- Precision: Precision-measured the ratio of correctly predicted positive instances to the total predicted positive instances. A higher precision indicated that the classifier made fewer false positive predictions. Precision is calculated by dividing the total number of instances classified as faulty (TP + FP) by the number of cases accurately identified as defective (TP) [14].
- Recall (Sensitivity): Recall calculated the ratio of correctly predicted positive instances to the actual total positive instances. A higher recall indicated that the classifier captured more of the actual positive instances.
- F1-Score: F-score metrics, which are widely used in the literature, represent the harmonic mean of accuracy and recall [15].
- ROC-AUC, by evaluating the trade-offs between TPR and FPR, determines the area under the receiver operating characteristic (ROC) curve.

True Positive (TP) represents the number of defective software instances correctly identified as defective, while True Negative (TN) is the number of clean software instances correctly identified as clean. False Positive (FP) represents the number of clean software instances incorrectly identified as defective, and False Negative (FN) is the number of defective software instances incorrectly identified as clean.

3. Experimental Methodology and Model Development

The experiment was tested on multiple publicly available datasets. These datasets include the Ar1, Ar4, Jm1, Kc1, Pc1, Pc4 and Mozilla4 datasets. Details about the dataset are provided in Table 1 below including the number of samples in each dataset, the Language of the original datasets, and the number of features computed from each of the dataset.

Table 1 Dataset Summary

Dataset	Number of Modules	Number of Defective Modules	Number of Attributes (Software Metrics)
JM1	7782	1672	21
AR1	194	36	39
AR4	1988	46	38
MOZILA	125	44	39
PC1	705	61	37
PC2	745	16	36
PC4	1287	177	37

In this section, we discuss the critical phase of the classifier model development. The experiment focuses on the design and implementation of an enhanced ensemble classifier and the steps deployed toward the construction and implementation of a robust classifier model.

The foundation of any predictive model lies in the quality and suitability of the underlying data. The dataset used for this research was sourced from an online repository which includes the PROMISE repository web and NASA repository. Comprising Ar1, Ar4, Jm1, Kc1, Pc1, Pc4 and Mozilla4. Upon completion of all the Data preprocessing and cleaning of the Datasets, the dataset was cleaned and ready for training and testing.

To construct the proposed enhanced model, a range of machine learning algorithms, each serving as a base classifier was used for the experiment. These included Decision Trees, Random Forests, Support Vector Machines (SVM), and Gradient Boosting. Decision Trees provide interpretability, while Random Forest and Gradient Boosting excel in capturing complex relationships. SVM was chosen for its ability to handle high-dimensional data and nonlinear decision boundaries.

The proposed classifier combined the predictions of these base classifiers using a soft voting strategy. One of the methods an ensemble classifier uses to forecast is the voting strategy. A voting classifier is a kind of machine learning model that anticipates an output (class) based on the probability that the chosen class will be the outcome after learning from a large ensemble of models. To predict the output class based on the most votes, it merely averages the results of each classifier that was entered into the voting classifier. Instead of building individual specialized models and evaluating their accuracy, the objective is to create a single model that learns from multiple models and makes predictions based on the total number of votes for each output class.

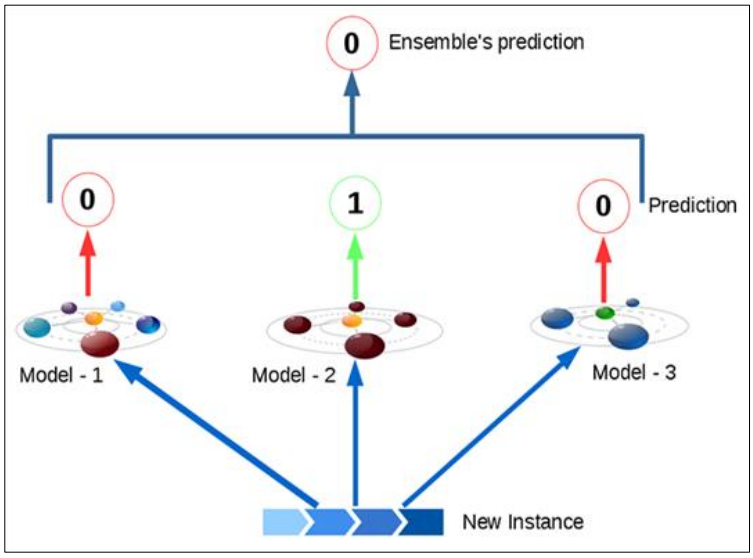

Figure 1 Voting technique Architecture Datajango (2019)

Fig. 1 shows the voting technique architecture with three base models and two out of the three predicted 0. A vote is carried out with the majority being 0 predicted as output.

There are two voting techniques known as Hard and Soft Voting.

In a hard voting classifier, the outcome is determined by which classifiers received the most votes overall.

On the other hand, a soft voting classifier is based on the average predicted class probabilities of the several classifiers. Soft voting frequently results in improved performance, especially when the classifiers produce predictions with varying degrees of uncertainty.

It is on the strength of the soft voting classifier that this study built upon in modifying and improving it to derive an even better prediction accuracy.

The Soft Voting classifier/model is represented as

$$Psoft(class\ 0) = \frac{1}{N}\sum_{i=1}^{N} w_i\ .\ p_i, 0 \quad \ldots\ldots\ldots (1)$$

Where:
$Psoft$ is the soft voting probability for 0
n is the number of classifiers,
wj is the weight associated with the j-th base classifier,
Pi (X=j) is the probability assigned by the i-th base classifier to the j-th class.

In soft voting, the class labels are predicted based on the weighted average of the predicted probabilities from individual classifiers. This strategy tends to perform well when the base classifiers have varying strengths. Soft voting uses the accuracy metrics for evaluation by combining the predicted probabilities from multiple models and selecting the class with the highest average. The disadvantages of this are that equal weight is given to all the models regardless of their performance and biases can still influence the final prediction. The use of only one single metric (accuracy) for evaluation may not provide an accurate estimate of the model's true performance. The soft voting technique does not also guarantee model adaptability or generalization to new data sets that may be given to the model.

To solve the above shortcomings, the cross-validation score for each base model is obtained and normalized. Cross-validation is a technique used to assess the model's performance by splitting the data set into multiple subsets or folds. Each fold is used as a validation set while the remaining folds are used for training. The cross-validation score is the average score performance of the model across all folds.

This technique offers a more robust evaluation metric when compared to just the model's accuracy. Cross-validation will enable the proposed classifier to mitigate the impact of data variability and provide a more representative measure of how well the model generalizes to unseen data, thereby making the classifier flexible.

The purpose of normalizing cross-validation scores is to ensure the derived scores from each training cycle are scaled to a common range. This also ensures that each base model contributes to the outcome of the model since all the base models are trained on the data set and the scores derived. This goes a long way in improving and enhancing a model's predictive capabilities. A detailed algorithmic description of the Ensemble Classifier is presented, elucidating the inner workings of this implementation.

4. Ensemble Classifier Algorithm

- Select n base classifiers (base_clf_1, base_clf_2, ..., base_clf_n)
- Send all the trained data for cross validation (CV) for each of the task models, and then find CV scores for all of them.
    - $Train\ Data = \begin{cases} Base\ model\ 1 \ \rightarrow cross\ validation\ score\ 1 \\ Base\ model\ 2 \ \rightarrow cross\ validation\ score\ 2 \\ \vdots \\ Base\ model\ N \ \rightarrow cross\ validation\ score\ N \end{cases}$ Summation of all CV scores:
- Sum of all CV score=CV score 1+CV score 2+ ⋯+CV score N
- Obtain the square root of sum of CV score:
    - $A = \sqrt{Sum\ of\ all\ CV\ scores} = \sqrt{CV\ score\ 1 + CV\ score\ 2 + \ \cdots + CV\ score\ N}$
- Normalization scores: Score n = Cv Score n/sqrt (Sum (CV Scores))
- Obtain the probabilities of success of all the base models.

From the Algorithm above, it shows the steps in building up the ensemble classifier which is represented with the equation below.

$$s_n = \{\frac{CV_{Score_n}}{\sqrt{\sum_{n=1}^{N} CV_{Score}}}, for\ i = 1, 2, \dots, N \dots\dots\dots\dots\dots\dots (2)$$

Where, i is the number of base classifiers.
CV_{Score_n} is the cross-validation score of the ith base classifier.
s_n is the normalized score for i th base classifier.

$$norm_{prob} = \sum_{n=1}^{N} S_n * P_n \dots\dots\dots\dots\dots\dots\dots\dots (3)$$

$$Ec = \begin{cases} 1, if \sum_{n=1}^{N} S_n * P_n > 0.5 \\ 0, otherwise \end{cases} \quad \ldots\ldots\ldots\ldots\ldots (4)$$

The above equation is the resulting Ensemble classifier derived during this study.

Where:
Ec is the ensemble classifier.

P_n is the success probability of the ith base classifier.

$norm_{prob}$ is the normalized probability which we find by using the above formula?

Equation 4 shows the combination of soft voting and cross-validation where the average confidence probability of each base classifier is combined with the normalized cross-validation score obtained from the based classifier to derive a very robust and dynamic ensemble classifier.

This is achieved by first finding the cross-validation score of each base model using the training dataset. Now each model's CV score is normalized by dividing it by the sum of all scores.

Now, each record for which the prediction is required is predicted through each of the base models and the probability of success is found out. Each probability is multiplied by the percentage of its respective base model and then all the probabilities are summed up. If the value is less than 50, failure is predicted, or else a success.

5. Experiment Results and Discussion

5.1. Presentation of Findings

As earlier stated, the classifier model was deployed on a total of six data sets with evaluations based on accuracy, precision, recall, F1 score, and ROC. then derived a performance score which is derived by dividing all the metrics. Find below the outputs from the deploying model.

The outcome of the experiment is presented in Tables 2 and 3 after deploying the proposed classifier on all selected data sets obtained from the NASA website. From the output displayed, the outcome showed a very strong performance across all the metrics from our proposed classifier. Also observed was that the classifier averages almost 90% across all metrics in terms of performance making it a very strong classification model. When compared to an ensemble soft voting classifier.

Table 2 Output derived from deploying the proposed classifier across all 7 datasets using

Datasets	Accuracy	F1 Score	Precision	Recall	ROC AUC
AR1	0.985294	0.985507	0.971429	1.000000	0.999134
AR4	0.886792	0.903226	0.875000	0.933333	0.946429
PC1	0.930225	0.930536	0.932760	0.928322	0.982217
PC3	0.931116	0.933941	0.969267	0.901099	0.982461
PC4	0.931116	0.921143	0.968750	0.877996	0.92461
JM1	0.832574	0.827936	0.811472	0.845082	0.900222
MOZILLA	0.954839	0.955128	0.967532	0.943038	0.987133

Table 3 Output derived from deploying the existing soft voting classifier across all 7 datasets

Datasets	Accuracy	F1 Score	Precision	Recall	ROC AUC
AR1	0.970588	0.971429	1.000000	0.944444	0.974048
AR4	0.924528	0.923077	0.888889	0.960000	0.974359
PC1	0.928948	0.928583	0.922513	0.934733	0.981523
PC3	0.922803	0.923619	0.953883	0.895216	0.980763
PC4	0.908551	0.913386	0.973621	0.860169	0.981442
JM1	0.832194	0.826463	0.793441	0.862351	0.901818
MOZILLA	0.948387	0.947712	0.957096	0.938511	0.983670

The results of the experiments, as illustrated in Tables 2 and 3 and summarized in Fig 2 below, indicate that the proposed model attained an accuracy average of 93%, contrasting with the existing Ensemble Soft Voting Classifier which achieved an accuracy average of 92% across all seven data sets experimented on. This suggests that the proposed model performed slightly better compared to the existing Ensemble soft voting classifier. The result also showed that the proposed classifier had less mis classified data when compared to the soft voting classifier making it a better preforming model. The proposed model also showed better performance in data generalization to new datasets hence its ability to have more correctly classified data and less misclassified data.

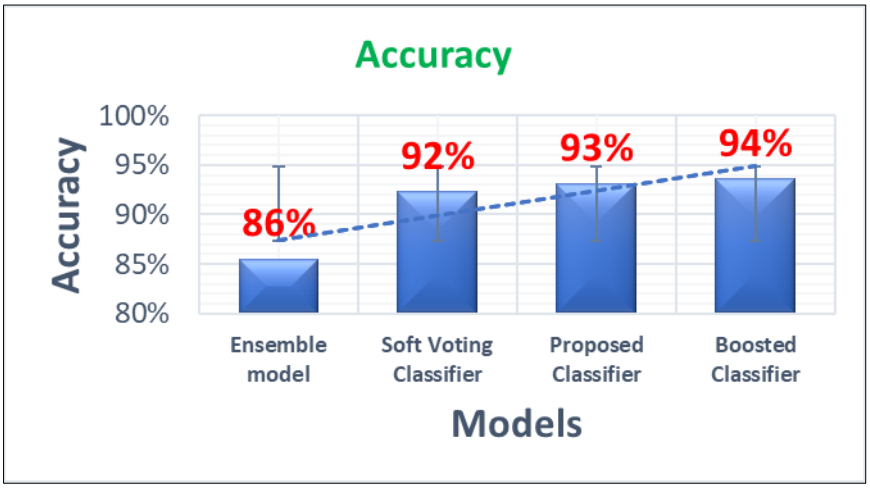

Figure 2 Average Classifiers Performance evaluation in terms of Accuracy across all datasets.

The Cross Validation technique that was applied to ensure each base model was trained on a fold of each dataset iteratively improved the learning of the model and ensure the classifiers were dynamic in nature. The Derived CV score from each training for each metric was also derived and normalized, ensuring the predictions were more robust and accurate compared to just using a single metric like accuracy; very important also is the process of normalization which can be attributed to as the main reason for the improved performance of our proposed model.

6. Related Works

Dada, et al. [16], who developed a multi-model ensemble machine learning approach for software predictions. Their research addressed the challenges of low classification accuracy and time-consuming processes in software defect detection. They presented an ensemble machine-learning model that integrated Linear Discriminant Analysis (LDA) with Random Forest as the basic learner, a Generalized Linear Model with Elastic Net Regularization (GLMNet), and k k-nearest neighbor (kNN). Using the RStudio simulation program, their experimental investigation concentrated on the CM1, JM1, KC3, and PC3 datasets from the NASA PROMISE repository. An accuracy level of 88.56% was attained on average by their ensemble technique. Alazba and Aljamaan [17] used a stacking ensemble approach built with a fine-tuned tree-based ensemble using a grid search algorithm to optimize seven selected tree-based ensembles to improve defect prediction in software. Their evaluation was on 21 publicly available data sets and they reported their results to

demonstrate the effectiveness of stacking ensemble overall fine-tuned tree-based ensemble. A study by Balogun et al. [18], noted that most studies considered model accuracy above other performance metrics and so evaluated single classifiers and ensemble techniques like stacking, bagging, and voting on 11 defect datasets. Their result ranked stacking technique and Voting as the highest and best performers in software defect prediction with a priority level of 0.0493. In addition, a review of the research done by Khalid et al. [19] on machine learning-based software defect prediction analysis was conducted. Improving model performance—more especially, accuracy and precision—was their goal. The classification of class labels was accomplished by the researchers using K-means clustering, and they applied classification models to specific features. To optimize their model, they employed partial swarm optimization. The performance evaluation included metrics such as precision, accuracy, recall, F-measure, performance error metrics, and a confusion matrix. According to their findings, all machine learning (ML) and optimized ML models achieved maximum results. Specifically, the accuracy of Optimized Random Forest (RF), and ensemble approaches reached 93.90% and 93.80%, respectively. This comparison provides a valuable benchmark for evaluating the efficacy of our proposed model in the context of software defect prediction. However, the current model, which aimed to Design an Ensemble Classifier for Software Defect Detection and Prediction achieved an overall accuracy level of 93%.

Table 4 is a tabulation and a suitable visualization to represent the accuracy percentages of the different models' studies in the context of software defect prediction.

Sanchita Pandey et al. [20] used in their experiment. To prevent over-fitting, a genetic algorithm was employed to extract pertinent features from the obtained datasets.

Using techniques from artificial neural networks, decision trees, and random forests, the collected features were sorted into non-defective and defective classes.

Moreover, f-score, recall, accuracy, and precision were used to assess the methods. Following the completion of the experiments, it was found that, with average scores of 83.40%, 53.18%, and 52.04%, respectively, the random forest method outperformed the other algorithms in terms of accuracy, precision, and f-score. Furthermore, the outcomes demonstrated that neural networks excel in recollection with an algorithmic average score of 60%. As a result, the system assisted software developers in creating high-quality software by enabling them to verify that it had few or no flaws before shipping it to clients. While this is not a poor grade, the model's average defect recall rate of 60% indicates that there is still room for improvement.

Khan [21] proposed an integration of the sampling technique and some common classification techniques to form a stronger Ensemble model for software defect prediction. The empirical results conducted showed a positive output on the benchmark datasets of software projects when compared to single classifiers. The researchers however carried out their experiment on a limited number of single classifiers.

Ten well-known software defect datasets were utilized in a comparison analysis by Alsaeedi and Khan [22]. The selection of the dataset was based on how frequently the same dataset had been used in similar studies. The base classifiers that were used were LR, DS, RF, and Linear SVC SVM. For each base classifier, the options of bagging and boosting classifiers were also considered. Python was used as the platform for conducting the research. The study employed many metrics to assess the classifiers' performance, including classification accuracy, precision, recall, F-score, and ROC-AUC score.

The datasets were divided into ten successive folds using 10-fold cross-validation, which was used to assess the classifiers' performance. The remaining folds of data are for training, and one-fold is for testing. Following that, the standard Scaler function in was used to standardize and scale the features. The remaining folds of data are for training, and one-fold is for testing. The characteristics were then scaled and standardized using the Python standard Scaler function, which operates by scaling the features into unit variance and subtracting the mean. Given that SMOTE has been extensively employed in the literature to address imbalance concerns in training data for SDP, the oversampling strategy was applied exclusively to the training data due to the extreme imbalance of the datasets. It iterated over all datasets after first supplying a list of datasets and a list of classifiers. Ten-fold cross-validation was used to divide the datasets into training and testing sets, with the data being shuffled before the split. After the dataset was divided, the Standard Scaler function was used to scale and standardize the features. The training data for each fold were re-sampled using the SMOTE approach after the features had been standardized.

Achuta et al. [23] researched software defect prediction (SDP) using classifiers such as AdaBoost, Random Forest, and Random Tree. They propose a Boosting technique that generally reduces both the variance and the bias of the classification which should lead to a considerably more accurate prediction. The evaluation parameters set out in their

study include classification accuracy, precision, and recall score. Results show that the classification accuracy of AdaBoost Performed better by 2.25% than Random Forest, and by 3 % than Random Tree. Similarly, the recall, precision, and F measure outperforms the proposed classification algorithm. Interestingly this is one of the very few models that have set their evaluation parameters to include all major measuring metrics and has a good recall score.

Kumar et al. [24] applied a stacking ensemble learning technique to improve the prediction accuracy of defects and performance based on software quality-defined code characteristics. The Stacking technique combines the base classifiers by using a meta-classifier that learns base-classifiers output. It has certain advantages, such as easy implementation and combining classifiers by investigating various inducers. Their approach was evaluated and compared with different Machine Learning (ML) classifiers on ivy2.0, tomcat, and velocity1.6 datasets available in PROMISE. Their experimental findings revealed that the proposed approach has better prediction recall, accuracy, precision, AUC-ROC, and f-measure.

A deep forest-based defect prediction model was suggested by Sanchita Pandey et al [20]. By constructing the defect prediction model (DPDF) using a deep forest model, the authors developed a sophisticated method to enhance defect prediction performance. The new cascade technique that the model employs, which converts random forest classifiers into a layer-by-layer structure, is thought to help it find more fault features. Deep learning and ensemble approaches are fully utilized in the design. 25 open-source projects from four public data sets—NASA, Promise, AEEM, and Relink—were used to assess the suggested model. When compared to the best conventional machine learning algorithm, the testing result demonstrates that the method employed raised the AUC value by 5%, demonstrating that the suggested Deep technique is useful for predicting software defects in DPDF. Comparing DPDF to six baseline classifiers—GC Forest, Deep belief network, random forest, naïve Bayes, logistical regression, and support vector machine—was another way to assess the efficacy of the algorithm. It is important to remember, nevertheless, that this model was only evaluated using Area under Curve (AUC), a prediction-based metric. This suggests that many metrics, including recall, F-score, accuracy, and precision, were overlooked.

The above-mentioned approaches differ from the proposed approach in this paper in two ways. Firstly, we combined two techniques Ensemble and Cross-Validation to improve the classifier model's performance in terms of correctly classifying data and generalization to unseen data. Several evaluation metrics such as accuracy, F1 score, AROC, precision, and Recall were used compared to the deep forest-based defect prediction model suggested by Sanchita Pandey et al which was evaluated using only one metric in Area under Curve (AUC), a prediction-based metric. This suggests that many metrics, including recall, F-score, accuracy, and precision, were overlooked.

Secondly, a very similar study to our approach presented in this paper was conducted by Dada, et al. who developed a multi-model ensemble machine learning approach for software predictions. Their research addressed the challenges of low classification accuracy and time-consuming processes in software defect detection. They presented an ensemble machine-learning model that integrated Linear Discriminant Analysis (LDA) with Random Forest as the basic learner, a Generalized Linear Model with Elastic Net

Regularization (GLMNet), and k k-nearest neighbor (kNN). Using the RStudio simulation program, their experimental investigation concentrated on the CM1, JM1, KC3, and PC3 datasets from the NASA PROMISE repository. An accuracy level of 88.56% was attained on average by their ensemble technique.

Also, Kumar et al. [24] adopted the stacking technique in their approach of combining multiple base classifiers to improve prediction accuracy. This technique uses a meta-model approach to learn how to combine predictions. Even though this approach is suitable for complex interactions, it does not offer the kind of robustness the Voting technique adopted in this study brings. Combining this robust attribute of the voting technique and the generalization attribute of cross-validation promises to produce a more accurate prediction score and a more improved model performance.

The general finding in these related works is that Ensemble techniques generally improve the performance of models compared to solo models. However, the Ensemble approach alone is not enough as there is room to further improve the model's performance and generalization to unforeseen data. Therefore, we have focused on combining two techniques capable of improving performance, reducing misclassification, increasing generalization to unseen data, and increasing the robustness of the model.

Table 4 The Accuracy Percentages of the Different Models Studies in the Context of Software Defect Prediction

S/No	AUTHORS	TITLE	Accuracy (%)
1.	Dada et al. (2021)	Ensemble Machine Learning Model for Software Defect Prediction	88.56
2	Khalid et al. (2023)	Software Defect Prediction Analysis Using ML Techniques	93
3.	Proposed (2024)	Design of an Ensemble Classifier for software Defect Detection and Prediction	93

7. Conclusion and Future Works

In conclusion, this experiment focused on designing and implementing an enhanced Ensemble classifier that seeks to be better than the existing soft classification method in predicting defects in software. The model developed used a combination of soft voting technique and Cross-Validation technique to improve upon existing methods of defect prediction. Through experiments and testing, the study looked to improve upon the existing soft classification method by modifying the model. The modification is achieved by normalizing the cross-validation score from each base model and finding the weightage average. This allows each base model to contribute some weight during the prediction process thus improving the accuracy of the model. When compared to the existing soft voting classifier, the modified classifier seemed to perform slightly better and on the same level in some cases. These results have important implications for the field of software engineering, as more accurate efficient, and flexible methods of defect prediction can help reduce the costs associated with software development and maintenance.

For further research, the modified classifier can be modified to predict the impact of a defect on the entire software module or unit of test. A collaboration with an industry partner or open-source projects to apply this classifier and evaluate its performance is also recommended.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Autumn, A., H., & Hurry, H., R. (2023). Quality assurance in Software Engineering: Best practices for success. MULTIDISCIPLINARY JOURNAL OF INSTRUCTION (MDJI), Vol. 5 No. 1 (2023). https://doi.org/10.13140/RG.2.2.35963.28965

[2] Karnavel, K., & Dillibabu, R. (2014). Development and application of new quality model for software projects. *The Scientific World JOURNAL*, *2014*, 1–11. https://doi.org/10.1155/2014/491246

[3] Alsawalqah, H., Hijazi, N., Eshtay, M., Faris, H., Al-Radaideh, A., Aljarah, I., & Alshamaileh, Y. (2020). Software defect prediction using heterogeneous ensemble classification based on segmented patterns. *Applied Sciences*, *10*(5), 1745. https://doi.org/10.3390/app10051745

[4] Hassan, F., Farhan, S., Fahiem, M. A., & Tauseef, H. (2018). A review on Machine learning techniques for software defect prediction. *Technical Journal*, *23*(02), 63–71.https://www.tj.uettaxila.edu.pk/index.php/technical-journal/article/download/405/34

[5] Saheed, Y. K., Longe, O., Baba, U. A., Rakshit, S., & Vajjhala, N. R. (2021). An ensemble learning approach for software defect prediction in developing quality software product. In *Communications in computer and information science* (pp. 317–326). https://doi.org/10.1007/978-3-030-81462-5_29

[6] J. Ge, J. Liu and W. Liu, "Comparative Study on Defect Prediction Algorithms of Supervised Learning Software Based on Imbalanced Classification Data Sets," *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Busan, Korea (South), 2018, pp. 399-406, doi: 10.1109/SNPD.2018.8441143.

[7] Matloob, F., Aftab, S., & Iqbal, A. (2019). A framework for software defect prediction using feature selection and ensemble learning techniques. *International Journal of Modern Education and Computer Science*, *11*(12), 14–20. https://doi.org/10.5815/ijmecs.2019.12.01

[8] Gupta, M., Rajnish, K., & Bhattacherjee, V. (2023). Effectiveness of ensemble classifier over State-Of-Art Machine Learning classifiers for predicting software faults in software modules. In *Lecture notes in electrical engineering* (pp. 77–88). https://doi.org/10.1007/978-981-19-5868-7_7

[9] Lear, A. M., Dada, E. G., Oyewola, D., & Baba, A. (2021). Ensemble Machine Learning Model for Software Defect Prediction. *ResearchGate*.

[10] Nguyen et al. (2013). A comparative study of ensemble methods and feature selection for software defect prediction. Information and Software Technology, 55(3), 415-429.

[11] Marçal, I., & Garcia, R. E. (2023). A comprehensible analysis of the efficacy of Ensemble Models for Bug Prediction. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2310.12133

[12] Refaeilzadeh, P., Tang, L., & Liu, H. (2009). Cross-Validation. In *Springer eBooks* (pp. 532–538). https://doi.org/10.1007/978-0-387-39940-9_565

[13] Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., Abbas, S., & Soomro, T. R. (2021). Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review. *IEEE Access*, *9*, 98754–98771. https://doi.org/10.1109/access.2021.3095559

[14] Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software/the Journal of Systems and Software*, *81*(5), 649–660. https://doi.org/10.1016/j.jss.2007.07.040

[15] Lee, T., Nam, J., Han, D., Kim, S., & In, H. P. (2011). Micro Interaction Metrics for defect prediction. *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering 5-9 September 2011, Szeged, 311-321.* https://doi.org/10.1145/2025113.2025156

[16] Dada, E., G., Oyewole, D.,Duada, A., B. (2021) Ensemble Machine Learning Model for Software Defect Prediction. (2021a). *Advances in Machine Learning & Artificial Intelligence*, *2*(1). https://doi.org/10.33140/amlai.02.01.03

[17] Alazba, A., & Aljamaan, H. (2022). Software defect prediction using stacking generalization of optimized Tree-Based ensembles. *Applied Sciences*, *12*(9), 4577. https://doi.org/10.3390/app12094577

[18] Balogun, A. O., Bajeh, A. O., Orie, V. A., & Yusuf-Asaju, A. W. (2018). Software defect prediction using Ensemble Learning: an ANP based evaluation method. *FUOYE Journal of Engineering and Technology*, *3*(2). https://doi.org/10.46792/fuoyejet.v3i2.200

[19] Khalid, A., Badshah, G., Ayub, N., Shiraz, M., & Ghouse, M. (2023). Software defect prediction analysis using machine learning techniques. *Sustainability*, *15*(6), 5517. https://doi.org/10.3390/su15065517

[20] Pandey, S., & Kumar, K. (2023). Software Fault Prediction for Imbalanced Data: A survey on recent developments. *Procedia Computer Science*, *218*, 1815–1824. https://doi.org/10.1016/j.procs.2023.01.159

[21] Khan, M. Z. (2020). Hybrid Ensemble Learning Technique for Software defect Prediction. *International Journal of Modern Education and Computer Science*, *12*(1), 1–10. https://doi.org/10.5815/ijmecs.2020.01.01

[22] Alsaeedi, A., & Khan, M. Z. (2019). Software defect prediction using Supervised machine learning and ensemble techniques: A Comparative study. *Journal of Software Engineering and Applications*, *12*(05), 85–100. https://doi.org/10.4236/jsea.2019.125007

[23] Rao, S., Kumar, R. K., & Patra, P. K. (2018). Software Defect Prediction Using Various Classifiers. *Journal of Advanced Research in Dynamical and Control Systems*, *10*, 201–207. https://www.jardcs.org/backissues/abstract.php?archiveid=4563

[24] Kumar, P. S., Behera, H. S., Nayak, J., & Naik, B. (2021). Bootstrap aggregation ensemble learning-based reliable approach for software defect prediction by using characterized code feature. *Innovations in Systems and Software Engineering*, *17*(4), 355–379. https://doi.org/10.1007/s11334-021-00399-2