A Hybrid Approach to Software Defect Prediction with Stacking Classifier

[1]K.Jahnavi Saravanavalli, [2] Dr. R. Mahendran

Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Green Fields, Vaddeswaram, Gundur District, Andhra Pradesh 522 302

[1]jahnavikorla7@gmail.com, [2]mahemtechbdu@gmail.com

Abstract: The prediction of software defects is essential for improving software quality and reducing the cost of testing by determining problematic modules for focused testing. This paper Examines Sophisticated Methodologies, Including the "Synthetic Minority Over-Sampling Technique (Smote) and Particle Swarm Optimization (PSO), Tackle Difficulties Related to Class Impacts and Feature Selection, Utilizing Benchmark Datasets Such As CM1, JM1, JM1 MC2, MW1, PC1, PC3, and PC4". A stacking classifier that combines a tree of decision -making tree, random forests and lightGBM was used, and works quite well on all data files. The proposed method focuses on the use of a file learning to improve the accuracy of prediction by merging the best parts of several classifiers. Comprehensive evaluation shows that the proposed model is strong and reliable and finds that it is better to find problematic software modules than other models. The aim of this effort is to improve the prediction of defects in the software by offering cost -effective and efficient ways to find defects in time, which will lead to better quality of software and better use of resources during testing.

"Index Terms - *Machine learning, software defect prediction, ensemble classification, heterogeneous classifiers, random forest, support vector machine, naïve Bayes".*

1. INTRODUCTION

The software industry is extremely important for development in all areas of today's interconnected world life. With globalization, Globe quickly became a "global village." Software applications are now everywhere and are an important part of everyday life, business, and most important infrastructure [1], [2]. The Covid 19 pandemic was even stronger. More and more people and businesses are using online platforms for communication, shopping, working from home and other important tasks [3,], [5]. The software industry has ensured that software continues to influence every part of modern life, and its quality is of paramount concern. The lifecycle "Software Development Life Cycle (SDLC), Software Development and Quality Assurance (QA)" is closely related to working together. The development team proceeds with coding the code that is provided by the QA team to put it under rigorous tests. This step will make you locate and report a problem or errors in the software. Feedback is given to the development team and they attempt to deliver the problem. This cyclic method keeps on proceeding to acquire quality software products without error [6] [7]. Figure 1 demonstrates the way in which this workflow moves through development to QA and SDLC. The primary objective of this process is that the software undergoes the quality,

functionality and reliability testing to make sure that the software ready to be attributed to the user. However, the is not easy to create error-free software. Time, money, and trained workers - three important things have a major impact on how software is guaranteed. As more people want to deliver software faster, the requirements for fast and inexpensive testing methods are becoming increasingly important. Companies are under pressure to produce high-quality software quickly. At the same time, it maintains low cost and uses resources optimally [8]. As a result, many people were interested in automated options for predicting defects to optimize the QA process.

"Software Default Prediction (SDP)" prediction was a good way to solve these problems. SDP uses ML to view and infer previous software data as there is likely to be an error in future software modules. The SDP model helps teams to find possible flaws before They are subjected to different software measures like code size, code complexity as well as history defects [9]. Such proactive strategy enables the firms to make their testing efforts proactive, minimize the risk of issues, enhance software quality and exploit their resources at the same time [10]. As a result, SDP is becoming increasingly popular as an important device for modern software development. This helps businesses increase the need for stable, high quality software around the world. This is increasingly complex and interrelated.

2. LITERATURE SURVEY

The prediction of "Software Defect Prediction (SDP)" is now a very important topic of research, as there is a growing need for high -quality software and a strong need to make the most of time, money and people in software development. The purpose of the SDP is to find defective software modules at the beginning of the development process to be able to test and debug on these modules. Over the past few decades, different approaches to machine learning and file methods have been tested to make the software defects more accurate and reliable. This literature overview focuses on the most important ideas and methods in this discipline focusing on the selection of functions, problems with class imbalance and the method of learning the file.

Neuron networks have shown amazing skills to find complicated patterns in software measurement, and are therefore often used in models that predict software defects. "M. S. Alkhasawneh [11] has performed" extensive analysis of neural network application integrated with the methodology of choosing elements to predict software errors. The study emphasizes the need to find the most important elements in the data file to make the predictions more accurate. Choosing functions helps reduce the number of dimensions in the data, which accelerates training and reduces excess. This ultimately the models of defects. The Alkhasawheh study finds that neuron networks, if used with good methods of function selection, may be more accurate than typical ML classifiers when creating predictions.

Research conducted by T. F. Husin and M. R. Pribadim [12] examined the application of vector machines of the smallest squares for classification of defects. They used a selection of functions to find the most important functions in the data file that had the greatest impact on the

prediction process. The authors said that this hybrid method was more accurate predictions, showing that the selection of the right features is very important to improve ML models in predicting defects.

The problem with class imbalance is one of the biggest problems in SDP. This is when defective software modules are generally a small group compared to non -defictive modules. This imbalance can cause models to be biased towards the dominant class, making predictions less useful. To solve this problem, scientists dealt with other methods of data sampling, such as underline, underlining and techniques of synthetic data production, such as Smote (excessive synthetic minority sampling technique).

B. J. Odejide et al. [1] They conducted empirical investigations of several data scanning techniques to reduce the problem of classification weights in SDP. In contrast to traditional sample approaches such as random resampling and containers, the authors had more sophisticated techniques such as Small. Their study shows that the Small Base Sample sample approach works better as they created a minority class of synthetic samples that contributed to compensating data files and making the model more accurate. The study also shows that using data scanning methods in ML models such as "Random Forest and SVM significantly improves the ability to find defective software modules".

A. O. Balogun et al. [18] They made another important addition to determine class imbalances by examining that small-based learning techniques and methods are based on predictions of software defects. Your research shows that a uniform ensemble approach using several copies of the same classifier can make predictions more accurately when data is unbalanced. The authors constructed a voting file combining predictions from different classifiers to create a more accurate and balanced predictive model for defects.

Ensemble Learning has become a powerful method in SDP, as it can use the strengths of several classifiers to perform a final prediction that is more accurate and reliable. Numerous research has examined different ensemble strategies, including voting, bagging and stacking procedures.

F. Jiang et al. [16] They proposed a random learning methodology based on a reduction to predict software defects. To make the model more stable and less variable, the authors assembled several classifiers in the file. They also used algorithms to select functions to find the most important functions in the data file that made the predictions even more accurate. The file model made better than many of the best ML techniques, which shows that the learning of the file works well in SDP.

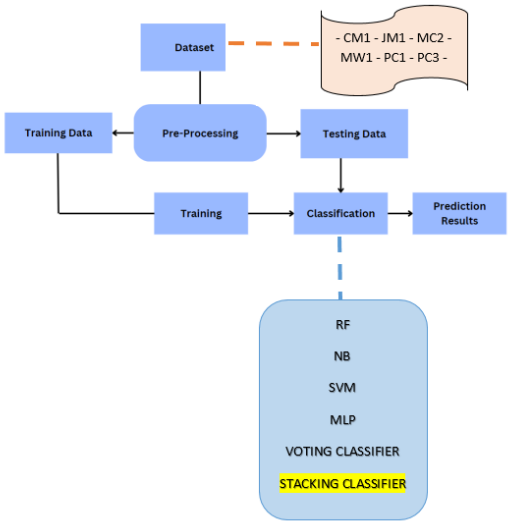
A.O. Balogun et al. [19] The notion of file integration and the assessment and testing on the basis of the Analytical Network Process (ANP). The researchers tested different file classifiers with APs in order to determine the best possible combo of classifiers to predict software defects. The authors concluded that the file techniques are on a consistent basis superior to the individual classifiers and therefore the models which produce trees, random forests, as well as support of Vector machines (SVMs) models produces optimal results.

R. J. Jacob et al. [20] Their study was carried out on the focused software error predictions on mood classification. The authors have analyzed the performance of some ensemble techniques like the study that revealed that adjustment-based ensemble techniques fit best to bring together the intensity of multiple classifiers like Randall Swald, Naive Books and "Support Vector Machines (SVMs)" to deliver better predictions. This has been confirmed during the study that the voting -based set techniques are far superior in predicting defects particularly in cases where one is dealing with a data set that does not appear to be balanced.

Stacking ensembles also gained great attention in the literature together with techniques based on the vote. Stacking uses another model of ML, usually called Meta-Learner, to aggregate forecasts of several classifiers and generating final predictions. A. Alsaeedi and M. Z. Khan [21] have a comparative analysis of various stacking file techniques to predict defects in software. Their study was examined by various combinations of "basic classifiers, including decision-making trees, Support Vector Machine and artificial neural networks that used various" meta-looters to strengthen the final predictions. The results of the study showed that the folded files were always better than individual classifiers. The biggest results came from models that included decision -making trees and neural networks.

3. METHODOLOGY

The proposed solution wants to improve the prediction of software errors using advanced ML methods and learning file technology. "The system uses various classifiers such as random forest, Support Vector Machine (SVM), Naive Bayes, and Multi-Layer Perceptron (MLP)" to get a baseline how well it can identify defects. "File approaches such as adaptive voting classifier, which combines the best parts of random forest, svm, naive bayes and MLP", are used to make things even more accurate and reliable. The stacking classifier, which combines the models of tree decision -making, random forest and lightGBM, is also used to use different learning patterns and to improve the ability to find defects. We use techniques such as excessive synthetic minority sampling techniques to repair the class imbalance. To ensure that the model is economical and efficient, we use particle swarm optimization to select the best features. This method is manufactured to accurately find faulty modules and helps reduce the cost of developing high-quality software and long testing processes.



"Fig 1 Proposed Architecture"

The image shows a common way to work for classification tasks. You will first get a data file. Then you divide it into training and test sets. Before the model is trained, the training data must be processed in advance. "Then pre -processed training data are used to train various

ML algorithms such as Random Forest (RF), Naive Bayes (NB), Support Vector Machine (SVM), and Multilayer Perceptron (MLP)". These models are used to sort test data as soon as they are trained, and their predictions are compared with real labels to see how well they are doing. Also, file approaches such as voting classifier and stacking classifier are also used to predict more than one model. This could improve overall accuracy.

i) Dataset:

The initial step in the training phase includes aggregation of previous software defects. For this investigation we chose the well -known benchmark data sets from the "NASA Metrics Data Program (MDP) repository were selected, including CM1, JM1, MC2, MW1, PC1, PC3, and PC4". These data sets are often used in the research of the prediction of software defects, which guarantees representativeness and comparability to existing studies. Data files correspond to a single piece of software and instances to a piece of software module. The software quality attributes that were followed during SDLC included; "LOC_COMMENT, LOC_TOTAL, CALL_PAIRS, HALSTEAD_LENGTH and HALSTEAD_CONSTANT which were recorded in SDLC with these modules". The dependent properties are utilized in predicting and the independent attribute that indicates a counter of whether the module functions or not [25].

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	9.0	5.0	3.0	2.0	2
1	19.0	3.0	1.0	2.0	0
2	0.0	9.0	0.0	0.0	0
3	2.0	15.0	2.0	1.0	9
4	5.0	5.0	1.0	0.0	0

"Fig.2 Dataset for CM1"

	LOC_BLANK	BRANCH_COUNT	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CYCLOMAT
0	1.0	7.0	0.0	0.0	
1	5.0	37.0	0.0	6.0	
2	2.0	1.0	0.0	0.0	
3	16.0	1.0	0.0	0.0	
4	0.0	7.0	0.0	0.0	

"Fig.3 Datasets for JM1"

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	16.0	19.0	0.0	0.0	22
1	9.0	13.0	2.0	0.0	14
2	2.0	3.0	0.0	0.0	0
3	35.0	97.0	3.0	0.0	51
4	1.0	3.0	1.0	1.0	0

"Fig.4 Datasets for MC2"

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	5.0	7.0	13.0	0.0	7
1	5.0	21.0	3.0	1.0	2
2	2.0	5.0	0.0	0.0	4
3	3.0	5.0	3.0	0.0	15
4	3.0	5.0	2.0	0.0	5

"Fig.5 Datasets for MW1"

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	1.0	3.0	2.0	0.0	0
1	2.0	5.0	2.0	1.0	3
2	0.0	3.0	1.0	0.0	0
3	2.0	3.0	2.0	0.0	0
4	19.0	17.0	13.0	0.0	0

"Fig.6 Datasets for PC1"

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	16.0	13.0	1.0	6.0	11
1	2.0	7.0	0.0	0.0	7
2	1.0	13.0	5.0	0.0	0
3	8.0	3.0	1.0	0.0	1
4	1.0	5.0	2.0	1.0	1

"Fig.7 Datasets for PC3"

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	8.0	1.0	0.0	0.0	6
1	1.0	7.0	3.0	0.0	0
2	33.0	29.0	7.0	9.0	20
3	1.0	19.0	0.0	17.0	0
4	7.0	13.0	5.0	9.0	9

"Fig.8 Datasets for PC4"

ii) Pre-Processing:

a) Data Processing: Data processing is important phases that ensure that the data file is

good enough for predictive modeling. The first step is to find and remove duplicate data items to reduce redundancy and make sure everything is consistent. Cleaning gets rid of unnecessary or irrelevant features to make the data file easier. Label coding is used to change variable categories to numbers so that ML algorithms can cooperate with them. These processes are more complete and set the phase for modeling training that is accurate and fast.

b) "Feature Selection": The choice of functions is mainly about finding variables that are most useful for predictions. The input functions (x) and target variables are carefully selected from the data file and discarded. This reduces noise and processing time by receiving only the data needed for training and testing. Function selection improves model performance and prediction accuracy. In this case, the data file is limited to only the most important functions and corresponding output specifications.

c) Oversampling with SMOTE: Overgrowth synthetic minority technology corrects the level of the class in the data file by creating incorrect examples of minority classes. This approach creates new data points by filling in gaps between existing samples that maintain a balanced class. Smote eliminates distortion in machine learning models by visible minority class. This makes it easier to find defective modules correctly, especially in data sets, where classes are quite uneven.

d) Feature Selection using PSO: To find the most useful subgroup of functions, "Particle Swarm Optimization (PSO)" is used to select functions. PSO focuses on different combinations of characteristics over and over again to find the best for the model, based on how the particle swarms behave in social situations. By maintaining only the most important features and deprivation of those that are not useful or unnecessary, PSO makes the more efficient and accurate model of defect prediction and ensures that the data file is not too complicated or too simple.

iii) Training & Testing:

If you are training and testing the model, split the processed data file into two parts. One is for model training and the other is for testing. Models are created and improved using training subgroups so that they can be trained from the data. After the model is trained, it is tested in test subgroups, but in subgroups that we have never seen before, we evaluate how well it can generalize. This method ensures that the model can accurately predict fresh data that has not seen, which is a good way to test, how well it works.

iv) Algorithms:

Random Forest: A random forest is a way to learn, which builds several decision -making trees during training and combines their results to make them more accurate and less likely to be transformed. In this project, it is used to guess whether a piece of software is interrupted or not a combination of several trees findings to make a more reliable and accurate estimate.

"SVM (Support Vector Machine)": SVM is a subordinate model of learning that identifies the best hyperplane to divide the classes in a lot of dimensions. This project uses SVM to sort software modules according to quality parameters, making it easier to tell the difference

between defective and non -defictive modules with high levels of accuracy.

Naive Bayes: Naive Bayes is a probability classifier that uses Bayes' sentence and assumes that functions are independent. This project uses it to predict software errors by finding out how likely a piece of software is Buggy based on past data. It works well with huge data sets and categorical data.

"MLP (Multi-Layer Perceptron)": MLP is a form of artificial brain network of a large number of the layers of the neurons. It is used in classification. This study predicts software errors by using MLP to find complex expressions in the data and modeling how properties and target variables do not use linearity.

"Adaptive Voting Classifier (RF + SVM + NB + MLP)": Using a weighted voting system, it integrates "the adaptive voting classifier of the random forest, SVM, Naive Bayes and MLP, so that the predictions are more accurate overall". This file method uses the best parts of each model to make a strong prediction of software errors.

"Stacking Classifier (DT + RF with LightGBM): The stacking classifier combines decision tree (DT), random forest (RF)", and LightGBM to create a metamodel that trains from the cost of the base model to improve predictions. She uses this study to improve the accuracy of predicting defects by mixing several models and using their strengths.

4. EXPERIMENTAL RESULTS

Accuracy: The test accuracy is how well the difference between sick and healthy people can recognize. To find out how accurate the test is, we need to find out how many cases we looked at was real positives and how many of them were real negatives. In mathematics it can be said as:

$$"Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}} (1)"$$

Precision: The accuracy focuses on the percentage of correctly classified cases or samples of all those that have been marked as positive. The formula to determine the accuracy is therefore:

$$"Precision = \frac{\text{True Positive}}{\text{True Positive } + \text{False Positive}} (2)"$$

Recall: In ML, the appeal is a gauge that quantifies the capacity of a model to identify all of the applicable instances of a specific type. It represents the number of correctly classified positive calls out of the usual large actual positive cases which are measures of how a model depicts all the instances of an individual type.

$$"Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} (3)"$$

F1-Score: The F1 score is a way to quantify, how accurate the ML model is. It assembles the accuracy score and download the model. Accuracy statistics will tell you how many times the model has created a valid forecast on the entire data file.

$$"F1\ Score = 2 * \frac{Recall\ \mathrm{X}\ Precision}{Recall + Precision} * 100(1)"$$

Sensitivity: Sensitivity is a way to find out how well the test or tool can find a problem in humans. It comes to see how many people test a positive state compared to how many people actually have it.

$$Specificity = \frac{TP}{(TP + FN)} (5)$$

Specificity: Lets say you want to know divide the number of people who have been found not to have it status by the total number of people who do not have whatever it is, so combining the people who tested negative and the people who tested positive and did not stifle the thing.

$$Specificity = \frac{TN}{(TN + FP)} (6)$$

AUC-ROC Curve: AUC-ROC curve is a way to test, how well the classification problem works at different thresholds. ROC shows the real positive speed and false positive speed on the chart. AUC measures how well the model can recognize the difference between classes. Higher AUC means that the model works better.

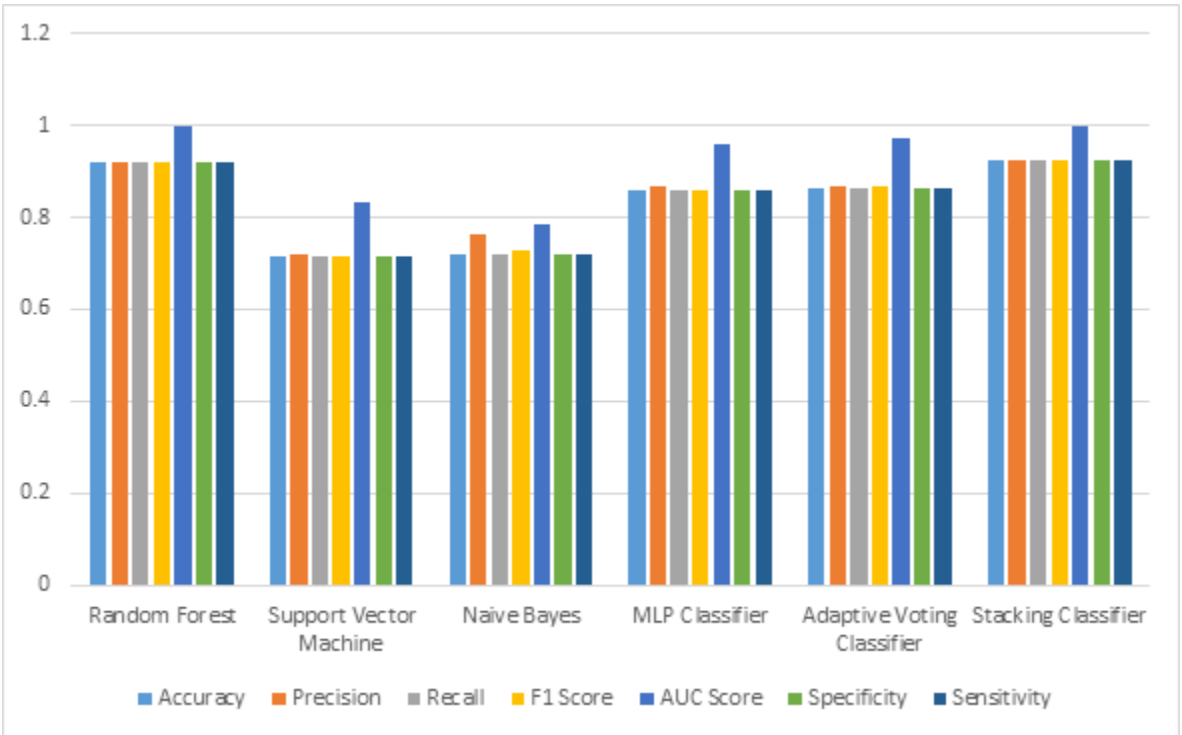
$$AUC = \sum_{i=1}^{n-1} (FPR_{i+1} - FPR_i) \cdot \frac{TPR_{i+1} + TPR_i}{2} (7)$$

"Tables (1 to 7) Check accuracy, accuracy, appeal, score F1, AUC score, specificity and sensitivity of each method". The stacking classifier regularly beats all other methods on all data sets. The tables also allow you to compare statistics for different algorithms.

"Table.1 Performance Evaluation Metrics for CM1"

ML Model	Accuracy	Precision	Recall	F1 Score	AUC Score	Specificity	Sensitivity
Random Forest	0.918	0.921	0.918	0.918	1.000	0.918	0.918
Support Vector Machine	0.713	0.719	0.713	0.714	0.833	0.714	0.713
Naive Bayes	0.719	0.765	0.719	0.726	0.783	0.721	0.719
MLP Classifier	0.860	0.867	0.860	0.860	0.961	0.859	0.860
Adaptive Voting Classifier	0.865	0.866	0.865	0.866	0.971	0.865	0.865
Stacking Classifier	**0.924**	**0.924**	**0.924**	**0.924**	**1.000**	**0.924**	**0.924**

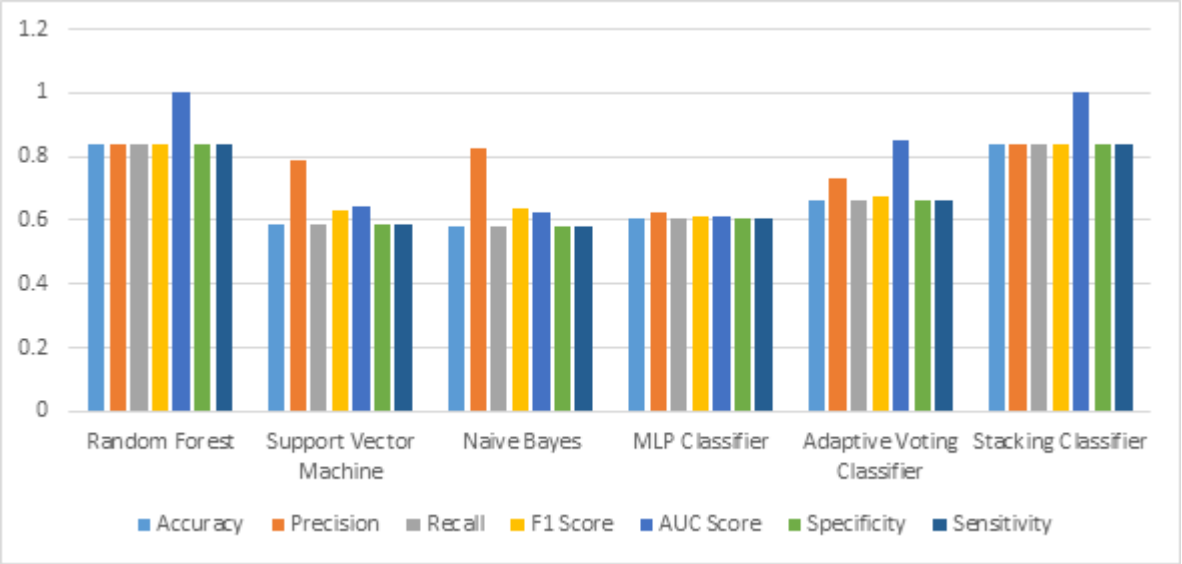
"Graph.1 Comparison Graphs for CM1"



"Table.2 Performance Evaluation Metrics for JM1"

ML Model	Accuracy	Precision	Recall	F1 Score	AUC Score	Specificity	Sensitivity
Random Forest	0.840	0.840	0.840	0.840	1.000	0.840	0.840
Support Vector Machine	0.588	0.786	0.588	0.633	0.643	0.588	0.588
Naive Bayes	0.582	0.825	0.582	0.639	0.624	0.582	0.582
MLP Classifier	0.608	0.622	0.608	0.611	0.613	0.608	0.608
Adaptive Voting Classifier	0.661	0.735	0.661	0.674	0.851	0.661	0.661
Stacking Classifier	**0.836**	**0.837**	**0.836**	**0.836**	**1.000**	**0.836**	**0.836**

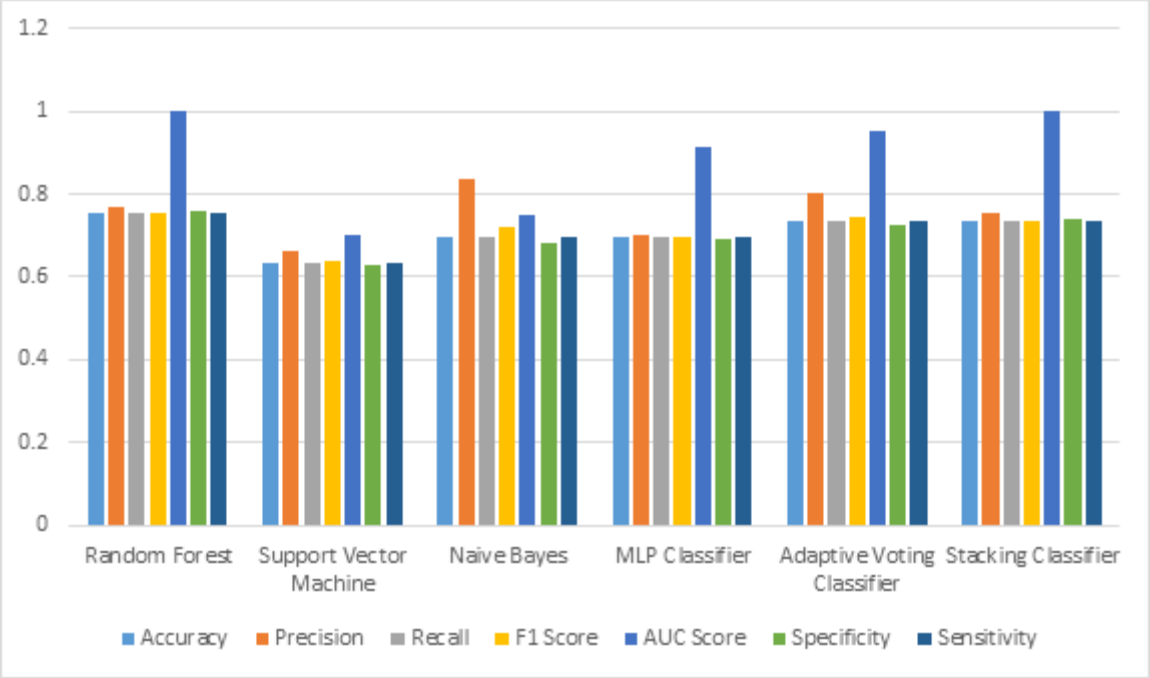
"Graph.2 Comparison Graphs for JM1"



"Table.3 Performance Evaluation Metrics for MC2"

ML Model	Accuracy	Precision	Recall	F1 Score	AUC Score	Specificity	Sensitivity
Random Forest	0.755	0.768	0.755	0.756	1.000	0.758	0.755
Support Vector Machine	0.633	0.664	0.633	0.639	0.702	0.627	0.633
Naive Bayes	0.694	0.838	0.694	0.719	0.750	0.683	0.694
MLP Classifier	0.694	0.702	0.694	0.695	0.913	0.691	0.694
Adaptive Voting Classifier	0.735	0.804	0.735	0.745	0.954	0.727	0.735
Stacking Classifier	**0.735**	**0.754**	**0.735**	**0.737**	**1.000**	**0.739**	**0.735**

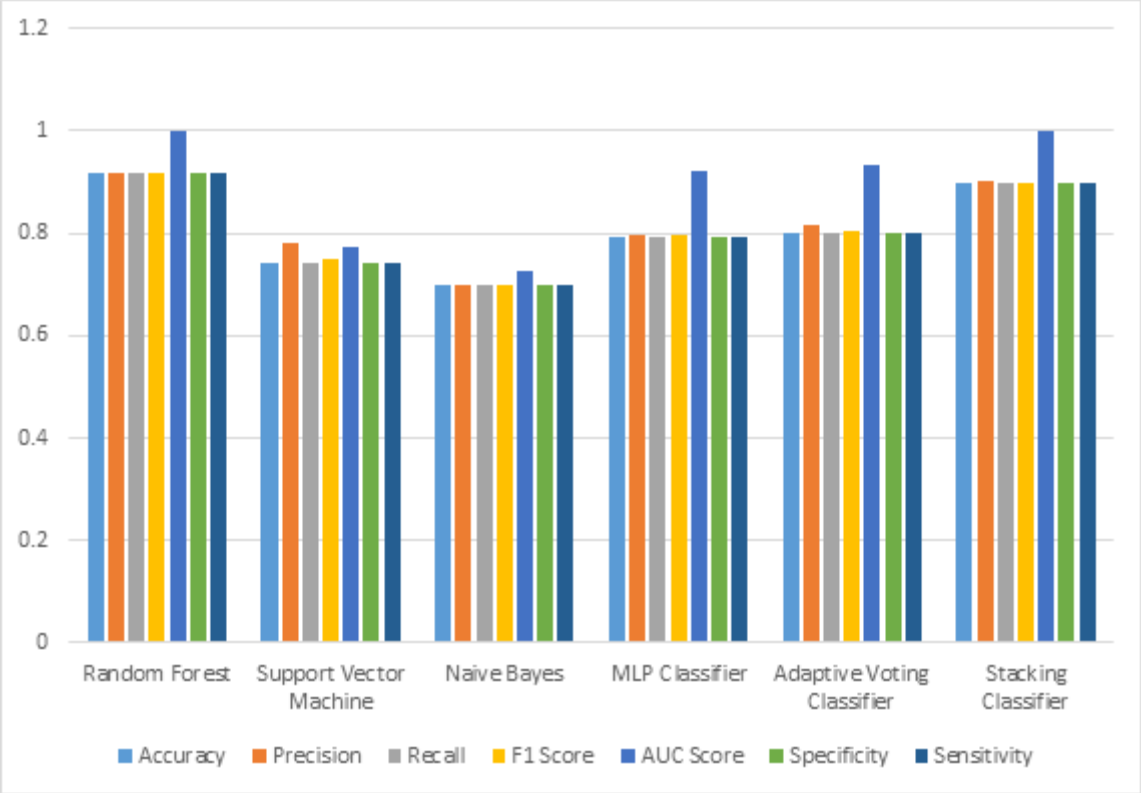
"Graph.3 Comparison Graphs for MC2"



"Table.4 Performance Evaluation Metrics for MW1"

ML Model	Accuracy	Precision	Recall	F1 Score	AUC Score	Specificity	Sensitivity
Random Forest	0.919	0.919	0.919	0.919	1.000	0.919	0.919
Support Vector Machine	0.743	0.782	0.743	0.748	0.772	0.743	0.743
Naive Bayes	0.699	0.700	0.699	0.699	0.726	0.699	0.699
MLP Classifier	0.794	0.798	0.794	0.795	0.923	0.794	0.794
Adaptive Voting Classifier	0.801	0.815	0.801	0.803	0.933	0.801	0.801
Stacking Classifier	**0.897**	**0.901**	**0.897**	**0.897**	**1.000**	**0.897**	**0.897**

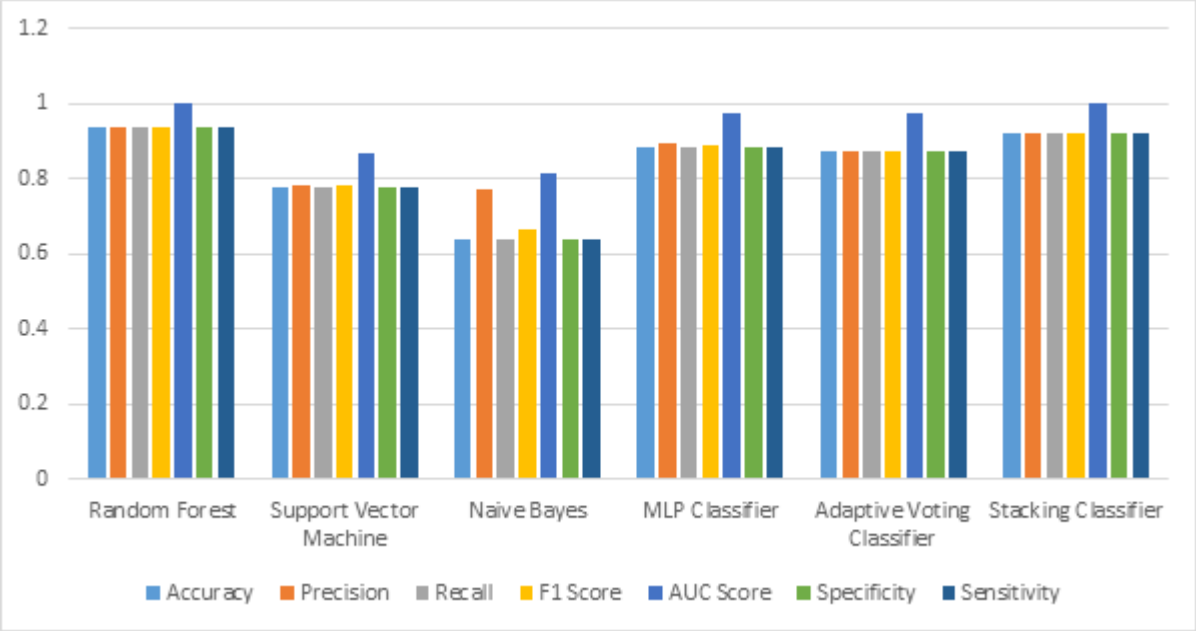
"Graph.4 Comparison Graphs for MW1"



"Table.5 Performance Evaluation Metrics for PC1"

ML Model	Accuracy	Precision	Recall	F1 Score	AUC Score	Specificity	Sensitivity
Random Forest	0.938	0.938	0.938	0.938	1.000	0.938	0.938
Support Vector Machine	0.780	0.782	0.780	0.781	0.867	0.780	0.780
Naive Bayes	0.638	0.771	0.638	0.664	0.813	0.640	0.638
MLP Classifier	0.886	0.893	0.886	0.887	0.975	0.886	0.886
Adaptive Voting Classifier	0.871	0.871	0.871	0.871	0.974	0.871	0.871
Stacking Classifier	**0.922**	**0.923**	**0.922**	**0.922**	**1.000**	**0.923**	**0.922**

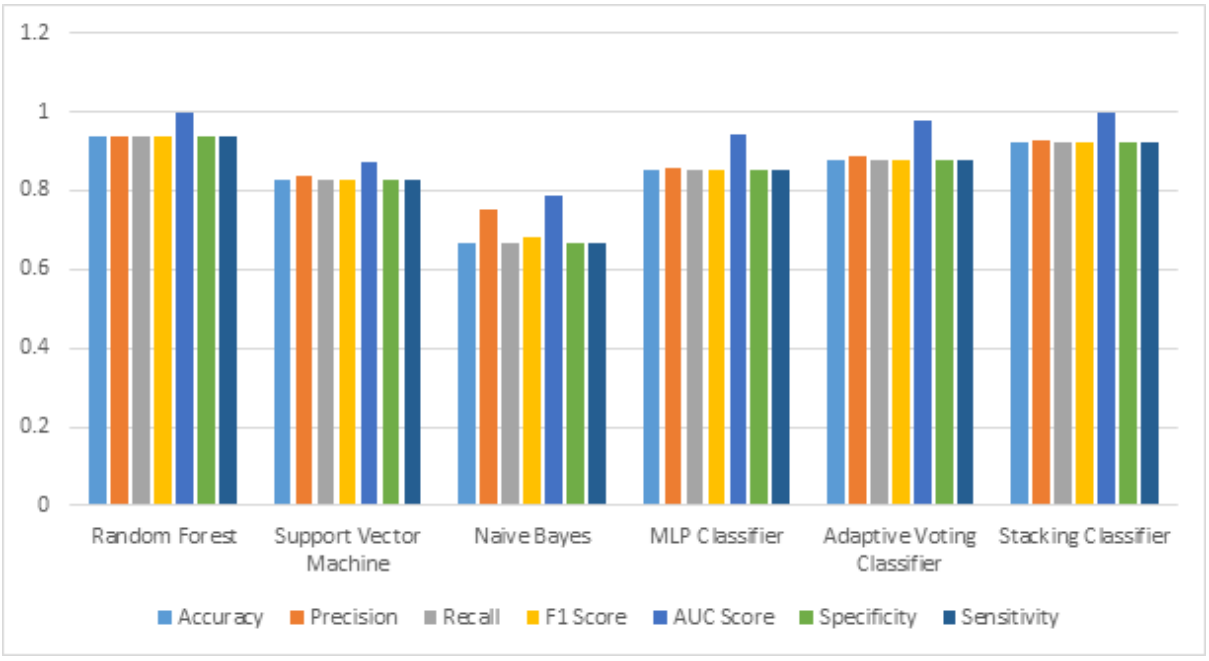
"Graph.5 Comparison Graphs for PC1"



"Table.6 Performance Evaluation Metrics for PC3"

ML Model	Accuracy	Precision	Recall	F1 Score	AUC Score	Specificity	Sensitivity
Random Forest	0.936	0.937	0.936	0.936	1.000	0.936	0.936
Support Vector Machine	0.829	0.835	0.829	0.829	0.870	0.829	0.829
Naive Bayes	0.668	0.752	0.668	0.682	0.788	0.668	0.668
MLP Classifier	0.850	0.855	0.850	0.850	0.941	0.850	0.850
Adaptive Voting Classifier	0.875	0.885	0.875	0.875	0.978	0.875	0.875
Stacking Classifier	**0.924**	**0.927**	**0.924**	**0.924**	**1.000**	**0.924**	**0.924**

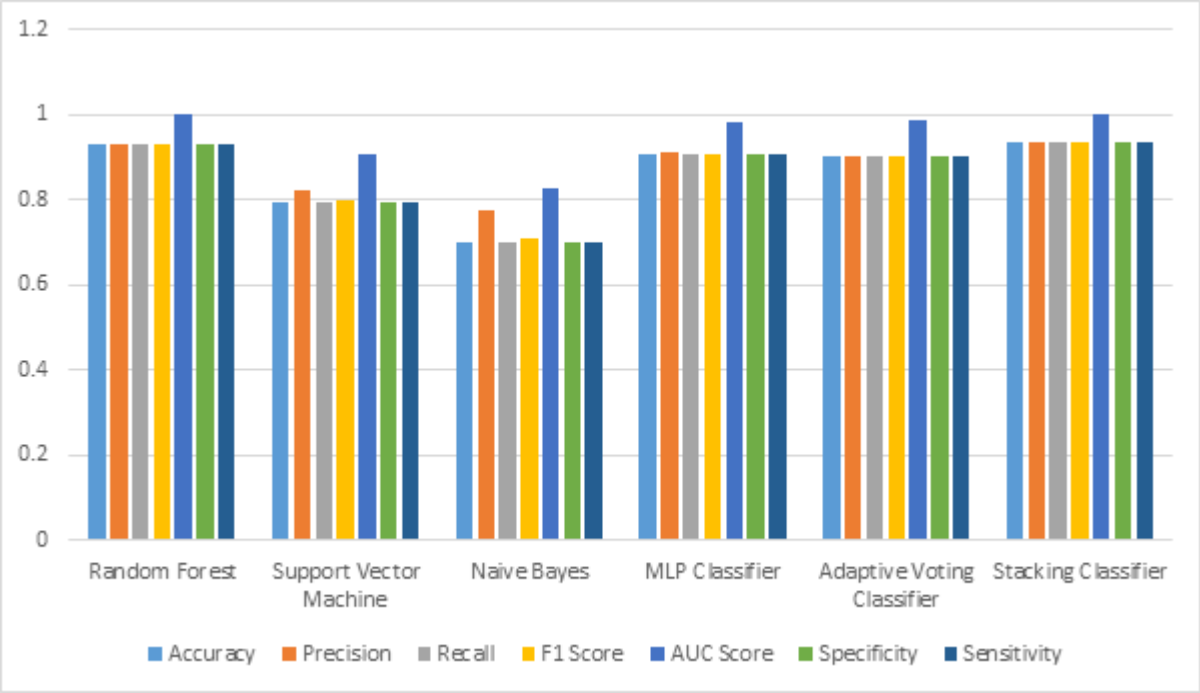
"Graph.6 Comparison Graphs for PC3"



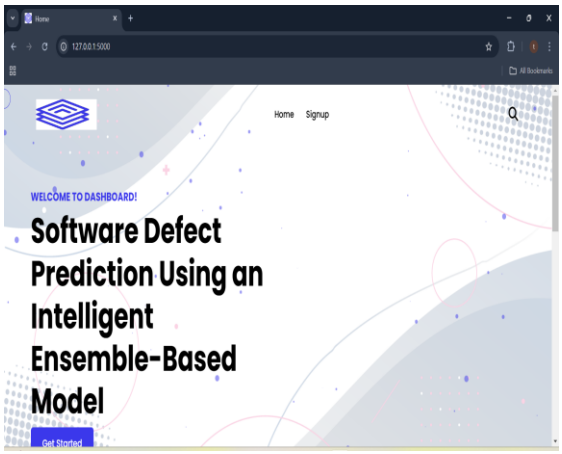
"Table.7 Performance Evaluation Metrics for PC4"

ML Model	Accuracy	Precision	Recall	F1 Score	AUC Score	Specificity	Sensitivity
Random Forest	0.929	0.930	0.929	0.929	1.000	0.929	0.929
Support Vector Machine	0.796	0.820	0.796	0.798	0.905	0.796	0.796
Naive Bayes	0.700	0.774	0.700	0.711	0.826	0.700	0.700
MLP Classifier	0.905	0.910	0.905	0.906	0.982	0.905	0.905
Adaptive Voting Classifier	0.904	0.904	0.904	0.904	0.987	0.904	0.904
Stacking Classifier	**0.935**	**0.936**	**0.935**	**0.935**	**1.000**	**0.935**	**0.935**

"Graph.7 Comparison Graphs for PC4"

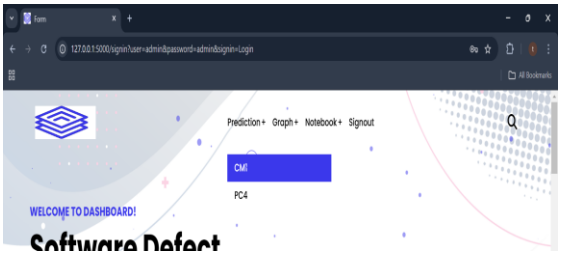


"Graphs (1 to 7) show accuracy in light blue, orange accuracy, memory in gray, f1-score in light yellow, AUC in blue, specificity in green and sensitivity in dark blue". The stacking classifier overcomes all other models on all data sets and reaches the highest values. The above graphs show these results in a visual way.



"Fig.9 Home Page"

This is the control panel of the user interface in the picture above. It has a welcome message for the navigation page.

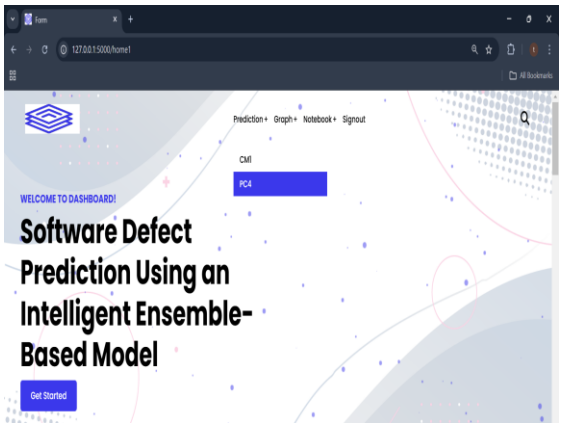


"Fig.10 User input Page"

This is the user's input page in the picture above. The user can upload data from the DataSetCM1 here.

"Fig.11 Test result for CM1"

This is the result screen in Figure 11 above. The user will see the output here.



"Fig.12 User input Page"

Figure 12 above shows the user's input page where the user can upload data from the PC4 data file.

"Fig.13 Test result for PC4"

This is the result screen in Figure 13 above where the user sees the output.

5. CONCLUSION

The aim of the defect defect is to find broken modules before the test phase so that testing can focus on modules that most likely have problems. The effective model of "defect prediction significantly reduces software development" expenditures by ensuring the best use of resources during work. "The system uses data sets such as CM1, JM1, MC2, MW1, PC1, PC3 and PC4, and advanced methods such as Smote to select Class and Particle Swarm Optimization (PSO)". This ensures that input data for training is balanced and useful. The stacking classifier was the most accurate of all tested methods on all data files. To improve his ability to create predictions. This combination of approaches and high -performance file learning shows how important it is to find defective modules, facilitate testing and help produce high -quality software in time and budget.

6. FUTURE SCOPE

Future research can examine the merger of DL models, including CNN and LSTM, to clarify complex formulas within the software failure

data. Using explained AI methods could make it easier to understand predictions, which would help engineers find out what caused defects. The use of this method on larger data sets in the real world and dynamic software systems would be more scalable and usable for a wider range of situations. Also, the use of advanced sampling methods with algorithms of optimization can even improve the prediction of defects, "which would help with cost -effective and high quality software development in a wide range of fields".

REFERENCES

[1] Z. M. Zain, S. Sakri, and N. H. A. Ismail, ''Application of deep learning in software defect prediction: Systematic literature review and meta analysis,'' Inf. Softw. Technol., vol. 158, Jun. 2023, Art. no. 107175, doi: 10.1016/j.infsof.2023.107175.

[2] M. Unterkalmsteiner et al., ''Software startups—A research agenda,'' 2023, arXiv:2308.12816.

[3] S. Aftab, S. Abbas, T. M. Ghazal, M. Ahmad, H. A. Hamadi, C. Y. Yeun, and M. A. Khan, ''A cloud-based software defect prediction system using data and decision-level machine learning fusion,'' Mathematics, vol. 11, no. 3, p. 632, Jan. 2023, doi: 10.3390/math11030632.

[4] S. Goyal, ''Heterogeneous stacked ensemble classifier for software defect prediction,'' in Proc. 6th Int. Conf. Parallel, Distrib. Grid Comput. (PDGC), Waknaghat, India, Nov. 2020, pp. 126–130, doi: 10.1109/PDGC50313.2020.9315754.

[5] S. Mehta and K. S. Patnaik, ''Stacking based ensemble learning for improved software defect prediction,'' in Proc. 5th Int. Conf. Microelectron., Comput. Commun. Syst., vol. 748, 2021, pp. 167–178.

[6] M. Shafiq, F. H. Alghamedy, N. Jamal, T. Kamal, Y. I. Daradkeh, and M. Shabaz, ''Retracted: Scientific programming using optimized machine learning techniques for software fault prediction to improve software quality,'' IET Softw., vol. 17, no. 4, pp. 694–704, Jan. 2023, doi: 10.1049/sfw2.12091.

[7] Y. Tang, Q. Dai, M. Yang, T. Du, and L. Chen, ''Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm,'' Int. J. Mach. Learn. Cybern., vol. 14, no. 6, pp. 1967–1987, Jan. 2023, doi: 10.1007/s13042-022-01740-2.

[8] S. Goyal, ''3PcGE: 3-parent child-based genetic evolution for software defect prediction,'' Innov. Syst. Softw. Eng., vol. 19, no. 2, pp. 197–216, Jun. 2023, doi: 10.1007/s11334-021-00427-1.

[9] J. Liu, J. Ai, M. Lu, J. Wang, and H. Shi, ''Semantic feature learning for software defect prediction from source code and external knowledge,'' J. Syst. Softw., vol. 204, Oct. 2023, Art. no. 111753, doi: 10.1016/j.jss.2023.111753.

[10] A. K. Gangwar and S. Kumar, ''Concept drift in software defect prediction: A method for detecting and handling the drift,'' ACM Trans. Internet Technol., vol. 23, no. 2, pp. 1–28, May 2023, doi: 10.1145/3589342.

[11] M. S. Alkhasawneh, ''Software defect prediction through neural network and feature selections,'' Appl. Comput. Intell. Soft Comput., vol. 2022, pp. 1–16, Sep. 2022, doi: 10.1155/2022/2581832.

[12] T. F. Husin and M. R. Pribadi, ‘‘Implementation of LSSVM in classification of software defect prediction data with feature selection,’’ in Proc. 9th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI), Jakarta, Indonesia, Oct. 2022, pp. 126–131, doi: 10.23919/EECSI56542.2022. 9946611.

[13] J. A. Richards, ‘‘Supervised classification techniques,’’ in Remote Sensing Digital Image Analysis. Cham, Switzerland: Springer, 2022, pp. 263–367.

[14] B. J. Odejide, A. O. Bajeh, A. O. Balogun, Z. O. Alanamu, K. S. Adewole, A. G. Akintola, and S. A. Salihu, ‘‘An empirical study on data sampling methods in addressing class imbalance problem in software defect prediction,’’ in Proc. Comput. Sci. Online Conf. Cham, Switzerland: Springer, Apr. 2022, pp. 594–610.

[15] X. Wu and J. Wang, ‘‘Application of bagging, boosting and stacking ensemble and EasyEnsemble methods for landslide susceptibility mapping in the three Gorges reservoir area of China,’’ Int. J. Environ. Res. Public Health, vol. 20, no. 6, p. 4977, Mar. 2023, doi: 10.3390/ijerph20064977.

[16] F. Jiang, X. Yu, D. Gong, and J. Du, ‘‘A random approximate reduct based ensemble learning approach and its application in software defect prediction,’’ Inf. Sci., vol. 609, pp. 1147–1168, Sep. 2022, doi: 10.1016/j.ins.2022.07.130.

[17] H. Chen, X.-Y. Jing, Y. Zhou, B. Li, and B. Xu, ‘‘Aligned metric representation based balanced multiset ensemble learning for heterogeneous defect prediction,’’ Inf. Softw. Technol., vol. 147, Jul. 2022, Art. no. 106892, doi: 10.1016/j.infsof.2022.106892.

[18] A. O. Balogun, A. O. Bajeh, V. A. Orie, and A. W. Yusuf-Asaju, ‘‘Software defect prediction using ensemble learning: An ANP based evaluation method,’’ FUOYE J. Eng. Technol., vol. 3, no. 2, pp. 50–55, Sep. 2018, doi: 10.46792/fuoyejet.v3i2.200.

[19] A. O. Balogun, F. B. Lafenwa-Balogun, H. A. Mojeed, V. E. Adeyemo, O. N. Akande, A. G. Akintola, A. O. Bajeh, and F. E. Usman-Hamza, ‘‘SMOTE-based homogeneous ensemble methods for software defect prediction,’’ in Computational Science and Its Applications—ICCSA 2020, vol. 12254, O. Gervasi, B. Murgante, S. Misra, C. Garau, I. B. D. Taniar, B. O. Apduhan, A. M. A. C. Rocha, E. Tarantino, C. M. Torre, and Y. Karaca, Eds. Cham, Switzerland: Springer, 2020, pp. 615–631.

[20] R. J. Jacob, R. J. Kamat, N. M. Sahithya, S. S. John, and S. P. Shankar, ‘‘Voting based ensemble classification for software defect prediction,’’ in Proc. IEEE Mysore Sub Sect. Int. Conf. (MysuruCon), Hassan, India, Oct. 2021, pp. 358–365, doi: 10.1109/MysuruCon52639.2021. 9641713.

[21] A. Alsaeedi and M. Z. Khan, ‘‘Software defect prediction using supervised machine learning and ensemble techniques: A comparative study,’’ J. Softw. Eng. Appl., vol. 12, no. 5, pp. 85–100, 2019, doi: 10.4236/jsea.2019.125007.

[22] A. Iqbal and S. Aftab, ‘‘A classification framework for software defect prediction using multi-filter feature selection technique and MLP,’’ Int. J. Mod. Educ. Comput. Sci., vol. 12, no. 1, pp. 18–25, Feb. 2020, doi: 10.5815/ijmecs.2020.01.03.

[23] M. Cetiner and O. K. Sahingoz, ‘‘A comparative analysis for machine learning based

software defect prediction systems,'' in Proc. 11th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT), Kharagpur, India, Jul. 2020, pp. 1–7, doi: 10.1109/ICCCNT49239.2020.9225352.

[24] K. Wang, L. Liu, C. Yuan, and Z. Wang, "Software defect prediction model based on LASSO–SVM,'' Neural Comput. Appl., vol. 33, no. 14, pp. 8249–8259, Jul. 2021, doi: 10.1007/s00521-020-04960-1.

[25] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets,'' IEEE Trans. Softw. Eng., vol. 39, no. 9, pp. 1208–1215, Sep. 2013, doi: 10.1109/TSE.2013.11.