Journal of **N**onlinear **A**nalysis and **O**ptimization
Vol. 16, Issue. 1: 2025
ISSN : **1906-9685**

ENHANCING SOFTWARE DEFECT PREDICTION WITH A HYBRID MACHINE LEARNING ENSAMBLE MODEL

Mrs. J. KUMARI[1], GUNJI.LALITA[2]
#1 Assistant Professor Department of Master of Computer Applications
#2 Pursuing M.C.A QIS COLLEGE OF ENGINEERING & TECHNOLOGY
Vengamukkapalem(V), Ongole, Prakasam dist., Andhra Pradesh- 523272

Abstract: Software defect prediction is essential for enhancing software quality and reducing testing costs by identifying defective modules for targeted testing. Utilizing benchmark datasets such as CM1, JM1, MC2, MW1, PC1, PC3, and PC4, this work explores advanced techniques like Synthetic Minority Over-sampling Technique (SMOTE) and Particle Swarm Optimization (PSO) for addressing class imbalance and feature selection challenges. A Stacking Classifier integrating Decision Tree, Random Forest, and LightGBM models has been employed, achieving high accuracy across all datasets. The proposed approach emphasizes leveraging ensemble learning to enhance prediction performance by combining the strengths of multiple classifiers. Comprehensive evaluations demonstrate the robustness and reliability of the proposed model, with results highlighting its superior accuracy and predictive capability in identifying defective software modules. This work aims to advance software defect prediction by providing a cost-effective and efficient solution for early defect detection, ensuring improved software quality and optimized resource allocation during testing.

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

1. INTRODUCTION

In today's interconnected global society, the software industry plays a critical role in driving progress across all sectors. Rapid globalization has effectively transformed the world into a "global village," where software applications are not only ubiquitous but also integral to daily life, business operations, and critical infrastructure [1], [2]. The COVID-19 pandemic has further accelerated this reliance on software systems, as individuals and organizations increasingly depend on online platforms for communication, commerce, remote work, and other essential activities [3], [4], [5]. As software continues to shape every aspect of the modern world, ensuring its quality has become a key priority for the industry.

In the context of the Software Development Life Cycle (SDLC), software development and quality assurance (QA) are tightly linked through a collaborative workflow. The process begins with the development team implementing the code, which is then handed over to the QA team for rigorous testing. This evaluation phase involves identifying and reporting defects or issues in the software. Feedback is provided to the development team, which works on rectifying the reported defects. This iterative process continues until a high-quality, defect-free software product is achieved [6], [7].

Figure 1 illustrates this workflow from development to QA within the SDLC. The primary objective of this process is to ensure that the software meets the required standards of quality, functionality, and reliability before it is released to end-users.

However, achieving defect-free software is not without its challenges. Three critical factors—time, financial resources, and skilled manpower—significantly impact the effectiveness of software quality assurance. As the demand for faster software delivery grows, the need for efficient and cost-effective testing strategies becomes more pressing. Companies are under increasing pressure to deliver high-quality software within tight deadlines while minimizing costs and optimizing the use of their resources [8]. This has led to the growing interest in automated approaches to defect prediction as a means to streamline the QA process.

Software Defect Prediction (SDP) has emerged as a promising solution to address these challenges. SDP involves the application of machine learning (ML) techniques to analyze historical software data and predict the likelihood of defects in future software modules. By using various software metrics—such as code complexity, size, and historical defect data—SDP models can help teams identify potential defects before the testing phase [9]. This proactive approach enables organizations to prioritize their testing efforts, reduce the likelihood of post-release defects, and ultimately improve software quality while optimizing resource usage [10]. Consequently, SDP is gaining traction as a vital tool in modern software development, helping organizations meet the growing demand for reliable, high-quality software in an increasingly complex and interconnected world.

2. LITERATURE SURVEY

Software Defect Prediction (SDP) has become a crucial research area due to the increasing demand for high-quality software and the pressing need to optimize resources, time, and manpower in software development. The goal of SDP is to identify defective software modules early in the development cycle, thus allowing for targeted testing and debugging. Over the past few decades, various machine learning techniques and ensemble methods have been explored to improve the predictive accuracy and reliability of software defect predictions. This literature survey explores the significant contributions and methodologies in this field, focusing on feature selection, class imbalance problems, and ensemble learning approaches.

Neural networks have demonstrated remarkable capabilities in detecting complex patterns in software metrics, which has made them a popular choice for software defect prediction models. M. S. Alkhasawneh [11] presented a comprehensive study on the use of neural networks combined with feature selection techniques for software defect prediction. The study emphasizes the importance of identifying the most relevant features in the dataset to improve prediction accuracy. Feature selection helps reduce the dimensionality of the data, which leads to faster training times and less overfitting, ultimately resulting in more robust defect prediction models. Alkhasawneh's research concludes that neural networks, when combined with effective feature selection methods, can outperform traditional machine learning classifiers in terms of predictive accuracy.

Similarly, the work of T. F. Husin and M. R. Pribadi [12] explored the implementation of Least-Squares Support Vector Machines (LSSVM) for defect classification. In their approach, feature selection was employed to identify the key features from the dataset that had the most significant impact on the prediction process. The authors reported that this hybrid approach improved prediction performance, demonstrating that feature selection plays a critical role in enhancing the accuracy of machine learning models used for defect prediction.

One of the major challenges in SDP is the class imbalance problem, where defective software modules are often a minority class compared to non-defective modules. This imbalance can lead to biased models that favor the majority class, thereby reducing the effectiveness of the predictions. Various data sampling methods have been explored to address this issue, including oversampling, undersampling, and synthetic data generation techniques such as SMOTE (Synthetic Minority Over-sampling Technique).

B. J. Odejide et al. [14] conducted an empirical study on different data sampling methods to address the class imbalance problem in SDP. The authors compared traditional sampling methods, such as random oversampling and undersampling, with more advanced techniques like SMOTE. Their findings indicated that SMOTE-based sampling methods provided superior performance by generating synthetic samples of the minority class, which helped balance the dataset and improve model accuracy. Additionally, the study highlights that combining data sampling techniques with machine learning models like Random Forest and SVM can significantly improve the detection of defective software modules.

Another significant contribution to addressing class imbalance was made by A. O. Balogun et al. [18], who explored the use of ensemble learning techniques combined with SMOTE-based methods for software defect prediction. Their research demonstrated that homogeneous ensemble methods, which combine multiple instances of the same classifier, can enhance predictive accuracy in imbalanced datasets. The authors proposed a voting-based ensemble method that aggregates the predictions of different classifiers, thus creating a more reliable and balanced defect prediction model.

Ensemble learning has emerged as a powerful approach in SDP due to its ability to combine the strengths of multiple classifiers to produce a more accurate and robust final prediction. Several studies have explored different ensemble techniques, including voting-based, bagging, and stacking methods.

F. Jiang et al. [16] introduced a random approximate reduct-based ensemble learning approach for software defect prediction. The authors combined multiple classifiers in an ensemble to reduce the variance and improve the robustness of the model. Their approach also incorporated feature selection methods to identify the most relevant features from the dataset, which further enhanced the accuracy of the predictions. The ensemble model outperformed several state-of-the-art machine learning algorithms, demonstrating the effectiveness of ensemble learning in SDP.

The work of A. O. Balogun et al. [19] extended the idea of ensemble learning by incorporating a ranking and evaluation framework based on the Analytic Network Process (ANP). The study evaluated different ensemble classifiers using ANP to determine the most effective combination of classifiers for software defect prediction. The authors found that ensemble methods consistently outperformed single classifiers, with the best results achieved by models that combined decision trees, random forests, and support vector machines.

A study conducted by R. J. Jacob et al. [20] focused on the application of voting-based ensemble classification for software defect prediction. The authors explored the performance of several ensemble methods, including majority voting, weighted voting, and probabilistic voting, to improve the accuracy of defect predictions. Their findings showed that voting-based methods were particularly effective in combining the strengths of different classifiers, such as Random Forest, Naive Bayes, and Support Vector Machines, to produce more reliable predictions. The study concluded that voting-based ensemble techniques offer a significant advantage in defect prediction, particularly when dealing with imbalanced datasets.

In addition to voting-based methods, stacking ensembles have also gained attention in the literature. Stacking combines the predictions of multiple classifiers by using another machine learning model, often referred to as the meta-learner, to make the final prediction. A. Alsaeedi and M. Z. Khan [21] conducted a comparative study of different stacking ensemble methods for software defect prediction. Their research explored various combinations of base classifiers, such as Decision Trees, SVM, and Artificial Neural Networks, and used different meta-learners to optimize the final predictions. The results of the study demonstrated that stacking ensembles consistently outperformed individual classifiers, with the best performance achieved by models that combined decision trees and neural networks.

3. METHODOLOGY

The proposed system aims to enhance software defect prediction by leveraging advanced machine learning techniques and ensemble learning strategies. The system incorporates individual classifiers such as Random Forest, Support Vector Machine (SVM), Naive Bayes, and Multi-Layer Perceptron (MLP) for baseline defect prediction performance. To further improve accuracy and robustness, ensemble methods like an Adaptive Voting Classifier, which combines the strengths of Random Forest, SVM, Naive Bayes, and MLP, are utilized. Additionally, a Stacking Classifier integrating Decision Tree, Random Forest, and LightGBM models is implemented to exploit diverse learning patterns and enhance defect detection capability. Techniques like Synthetic Minority Over-sampling Technique (SMOTE) are employed to address class imbalance, while Particle Swarm Optimization (PSO) aids in optimal feature selection, ensuring the model is both efficient and effective. This system is designed to identify defective modules accurately, prioritizing high-quality software development and reducing costs associated with exhaustive testing processes.

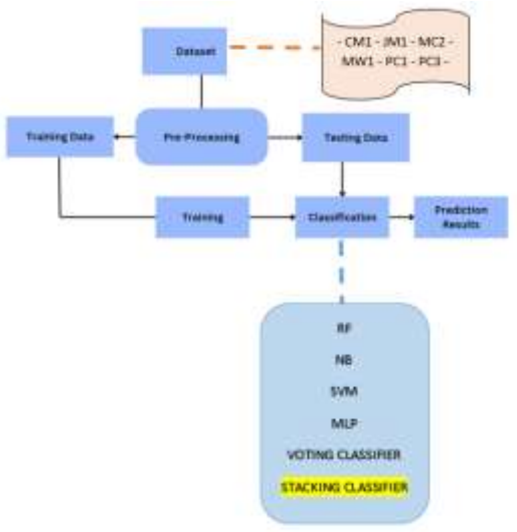
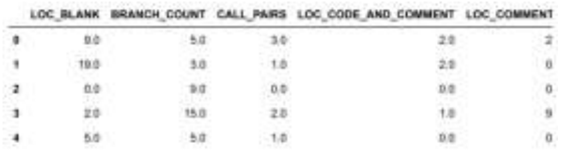


Fig 1 Proposed Architecture

The diagram illustrates a typical machine learning workflow for classification tasks. The process begins with a dataset, which is then divided into training and testing sets. The training data undergoes pre-processing to prepare it for model training. Subsequently, various machine learning algorithms, including Random Forest (RF), Naive Bayes (NB), Support Vector Machine (SVM), and Multilayer Perceptron (MLP), are trained on the pre-processed training data. Once trained, these models are used to classify the testing data, and their predictions are compared to the actual labels to evaluate their performance. Additionally, ensemble methods like Voting Classifier and Stacking Classifier are employed to combine the predictions of multiple models, potentially improving overall accuracy.

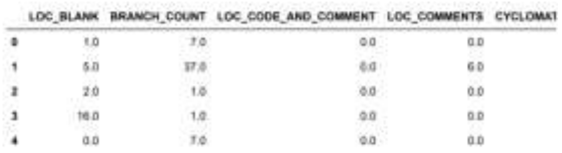
i) Dataset:

The first step in the training phase involves the collection of historical software defect datasets. For this study, well-established benchmark datasets from NASA's Metrics Data Program (MDP) repository were selected, including CM1, JM1, MC2, MW1, PC1, PC3, and PC4. These datasets are commonly used in software defect prediction research, ensuring representativeness and comparability with existing studies. Each dataset corresponds to a single software component, with instances representing software modules. These modules contain various software quality attributes, such as LOC_COMMENT, LOC_TOTAL, CALL_PAIRS, HALSTEAD_LENGTH, and HALSTEAD_CONSTANT, recorded during the SDLC. The dependent attributes are used for prediction, while the independent attribute indicates whether a module is defective or non-defective [25].



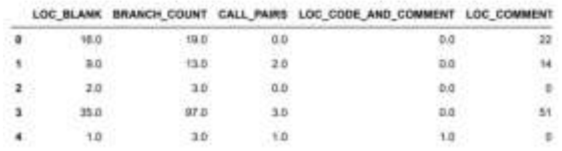
	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	0.0	5.0	3.0	2.0	2
1	19.0	3.0	1.0	2.0	0
2	0.0	9.0	0.0	0.0	0
3	2.0	15.0	2.0	1.0	9
4	5.0	5.0	1.0	0.0	0

Fig.2 Dataset for CM1



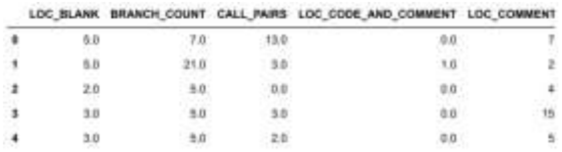
	LOC_BLANK	BRANCH_COUNT	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CYCLOMAT
0	1.0	7.0	0.0	0.0	
1	5.0	37.0	6.0	6.0	
2	2.0	1.0	0.0	0.0	
3	16.0	1.0	0.0	0.0	
4	0.0	7.0	0.0	0.0	

Fig.3 Datasets for JM1



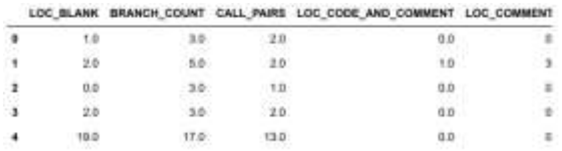
	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	16.0	19.0	0.0	0.0	22
1	8.0	13.0	2.0	0.0	14
2	2.0	3.0	0.0	0.0	8
3	35.0	97.0	3.0	0.0	51
4	1.0	3.0	1.0	1.0	8

Fig.4 Datasets for MC2



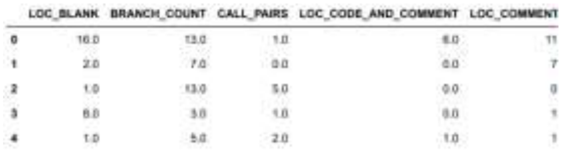
	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	6.0	7.0	13.0	0.0	7
1	8.0	21.0	3.0	1.0	2
2	2.0	5.0	0.0	0.0	4
3	3.0	5.0	3.0	0.0	15
4	3.0	5.0	2.0	0.0	5

Fig.5 Datasets for MW1



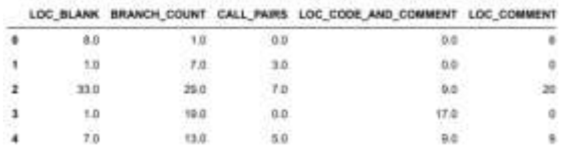
	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENT
0	1.0	3.0	2.0	0.0	8
1	2.0	5.0	2.0	1.0	3
2	0.0	3.0	1.0	0.0	8
3	2.0	3.0	2.0	0.0	0
4	10.0	17.0	13.0	0.0	8

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	6.0	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	25.0	7.0	9.0	20
3	1.0	19.0	0.0	17.0	0
4	7.0	13.0	5.0	9.0	6

Fig.8 Datasets for PC4

ii) Pre-Processing:

a) Data Processing: Data processing involves crucial steps to ensure the dataset's quality and suitability for predictive modeling. Initially, duplicate data entries are identified and removed to avoid redundancy and ensure consistency. Unnecessary or irrelevant features are dropped through cleaning to streamline the dataset. Label encoding is applied to convert categorical variables into numerical format, enabling compatibility with machine learning algorithms. These preprocessing steps enhance the dataset's integrity and lay a strong foundation for accurate and efficient model training.

b) Feature Selection: Feature selection focuses on isolating the most relevant variables for prediction. The input features (X) and target variable (y) are carefully identified and extracted from the dataset. This ensures the machine learning model receives only the necessary data for training and testing, reducing noise and computational overhead. By refining the dataset to include only critical features and the corresponding output labels, feature selection optimizes the model's performance and predictive accuracy.

c) Oversampling with SMOTE: Synthetic Minority Over-sampling Technique (SMOTE) addresses class imbalance in the dataset by generating synthetic samples for the minority class. This method creates new data points by interpolating between existing samples, ensuring a balanced distribution across classes. By enhancing the representation of the minority class, SMOTE prevents bias in machine learning models and improves their ability to identify defective modules accurately, especially in datasets with significant class imbalances.

d) Feature Selection using PSO: Particle Swarm Optimization (PSO) is employed for feature selection to identify the most informative subset of features. Inspired by the social behavior of particle swarms, PSO iteratively evaluates combinations of features to maximize the model's performance. By retaining only the most relevant attributes and discarding redundant or irrelevant ones, PSO enhances the computational efficiency and predictive accuracy of the defect prediction model, ensuring an optimal balance between dataset complexity and performance.

iii) Training & Testing:

Training and testing involve splitting the processed dataset into two subsets: one for training the model and the other for evaluation. The training subset is used to build and optimize the model, allowing it to learn patterns from the data. Once the model is trained, it is tested on the testing subset, which it has never seen before, to assess its generalization ability. This approach ensures that the model can make accurate predictions on new, unseen data, providing a reliable measure of its performance.

iv) Algorithms:

Random Forest: Random Forest is an ensemble learning method that constructs multiple decision trees during training and merges their outputs to improve accuracy and control overfitting. In this project, it is used to predict software defects by aggregating the results of multiple trees to provide a more stable and accurate classification of whether a module is defective or non-defective.

SVM (Support Vector Machine): SVM is a supervised learning model that finds the optimal hyperplane to separate classes in high-dimensional space. In this project, SVM is employed to classify software modules based on quality attributes, distinguishing between defective and non-defective modules with a high level of precision.

Naive Bayes: Naive Bayes is a probabilistic classifier based on Bayes' theorem, assuming independence between features. It is used in this project to predict software defects by calculating the likelihood of defectiveness based on historical data, making it effective for handling categorical data and large datasets.

MLP (Multi-Layer Perceptron): MLP is a type of artificial neural network with multiple layers of neurons, used for classification tasks. In this project, MLP is applied to learn complex patterns in the data and predict software defects by modeling non-linear relationships between features and the target variable.

Adaptive Voting Classifier (RF + SVM + NB + MLP): The Adaptive Voting Classifier combines predictions from Random Forest, SVM, Naive Bayes, and MLP using a weighted voting mechanism to improve overall prediction accuracy. This ensemble method leverages the strengths of each individual model to provide a robust prediction for software defects.

Stacking Classifier (DT + RF with LightGBM): The Stacking Classifier integrates Decision Trees (DT), Random Forest (RF), and LightGBM to create a meta-model that improves predictive performance by learning from the outputs of base models. In this project, it is used to enhance defect prediction accuracy by combining diverse models and leveraging their individual strengths.

4. EXPERIMENTAL RESULTS

Accuracy: The accuracy of a test is its ability to differentiate the patient and healthy cases correctly. To estimate the accuracy of a test, we should calculate the proportion of true positive and true negative in all evaluated cases. Mathematically, this can be stated as:

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

Precision: Precision evaluates the fraction of correctly classified instances or samples among the ones classified as positives. Thus, the formula to calculate the precision is given by:

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

Recall: Recall is a metric in machine learning that measures the ability of a model to identify all relevant instances of a particular class. It is the ratio of correctly predicted positive observations to the total actual positives, providing insights into a model's completeness in capturing instances of a given class.

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

F1-Score: F1 score is a machine learning evaluation metric that measures a model's accuracy. It combines the precision and recall scores of a model. The accuracy metric computes how many times a model made a correct prediction across the entire dataset.

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

Sensitivity: Sensitivity is a measure of how well a test or instrument can identify a condition in a subject. It's calculated by comparing the number of people who test positive for a condition to the actual number of people who have the condition.

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

Specificity: It is calculated by identifying the number of people who test negative for a condition and dividing it by the total number of people without the condition, which includes those who tested negative and the false positives—those who tested positive but did not have the disease.

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

AUC-ROC Curve: The AUC-ROC Curve is a performance measurement for classification problems at various threshold settings. ROC plots the True Positive Rate against the False Positive Rate. AUC quantifies the overall ability of the model to distinguish between classes, where a higher AUC indicates better model performance.

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

Tables (1 to 7) evaluate the performance metrics—accuracy, precision, recall, F1-score, AUC Score, Specificity and Sensitivity—for each algorithm. Across all datasets, the Stacking Classifier consistently outperforms all other algorithms. The tables also offer a comparative analysis of the metrics for the other 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

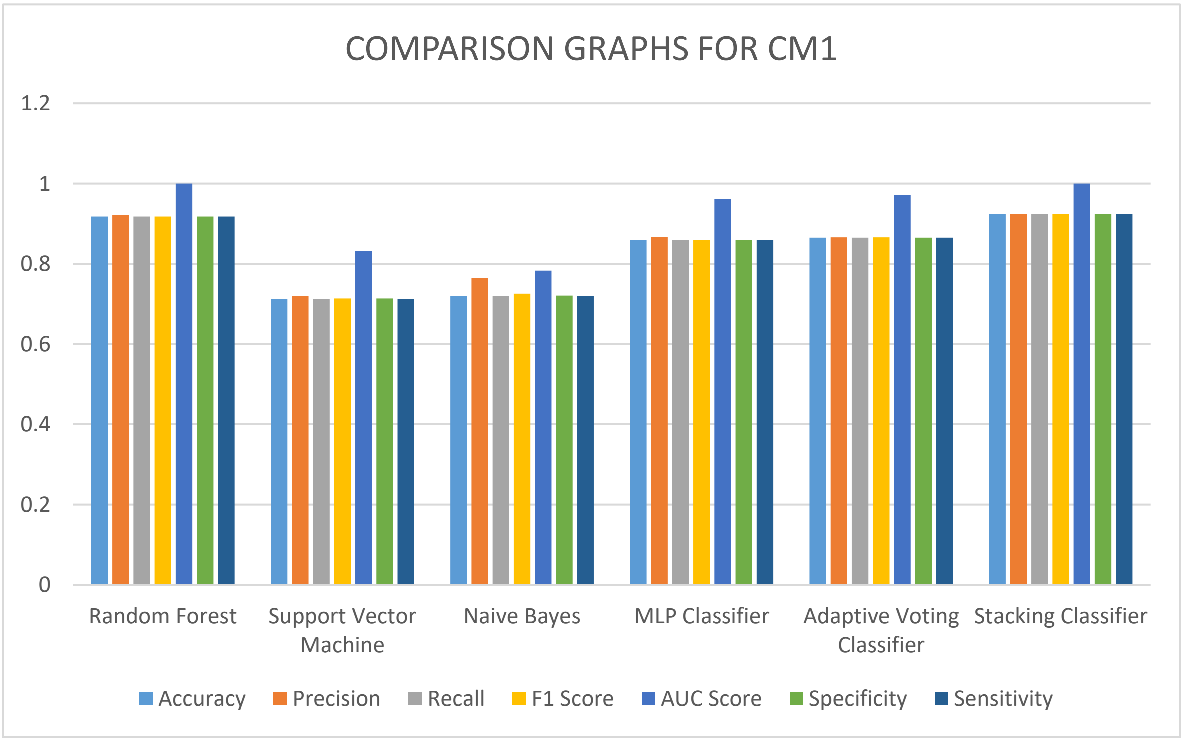


Table.2 Performance Evaluation Metrics for JM1

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

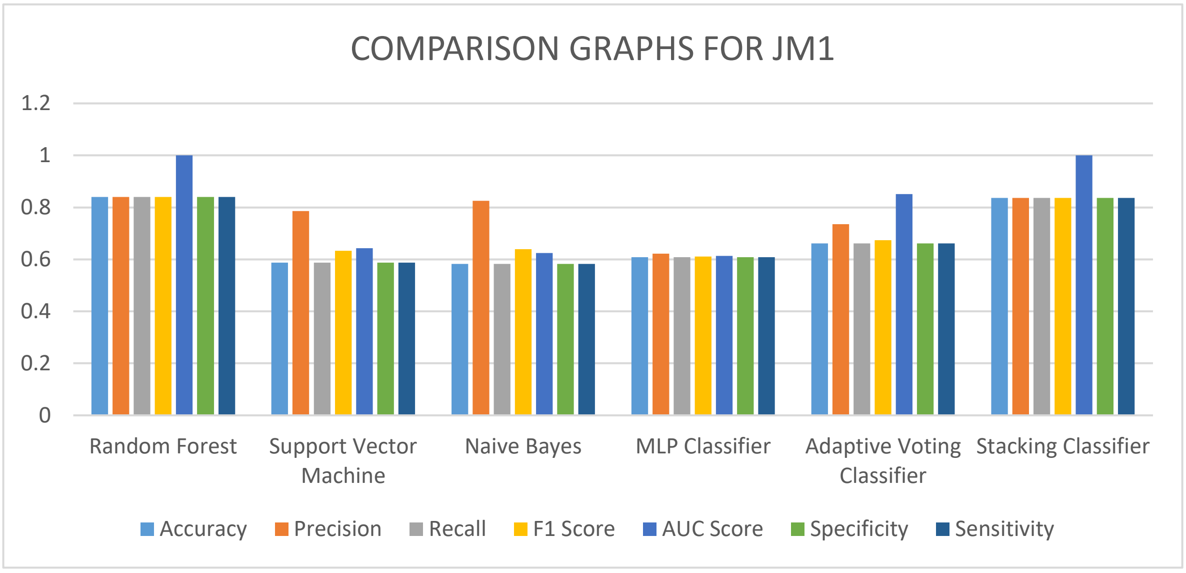


Table.3 Performance Evaluation Metrics for MC2

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

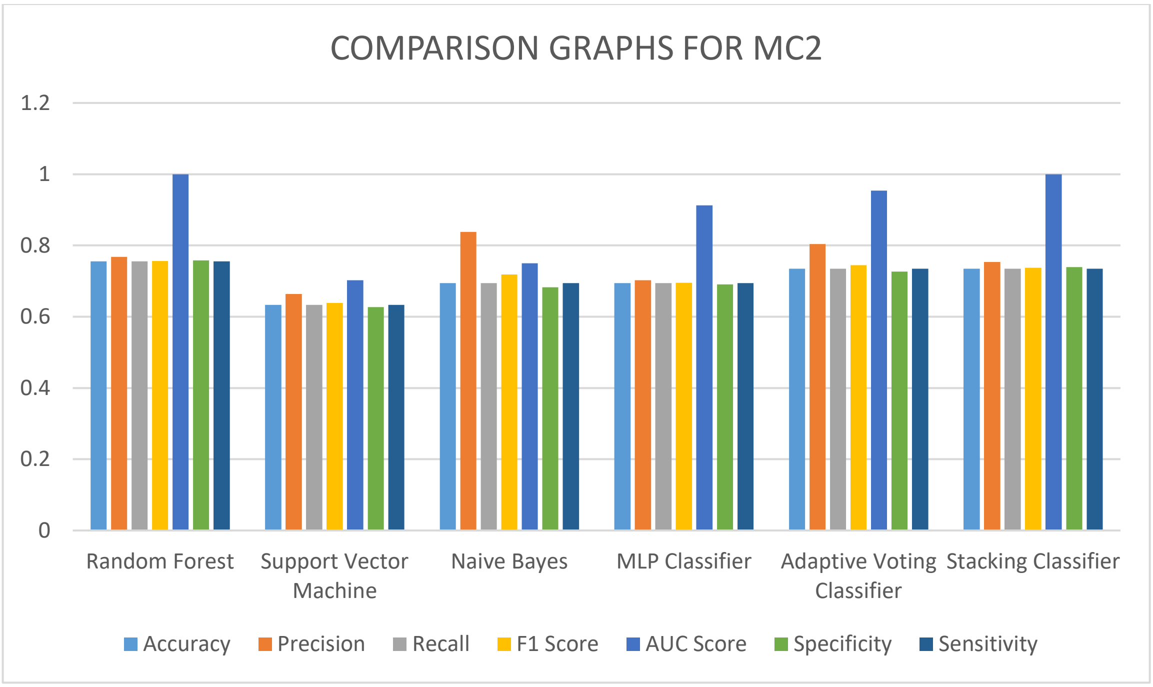


Table.4 Performance Evaluation Metrics for MW1

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

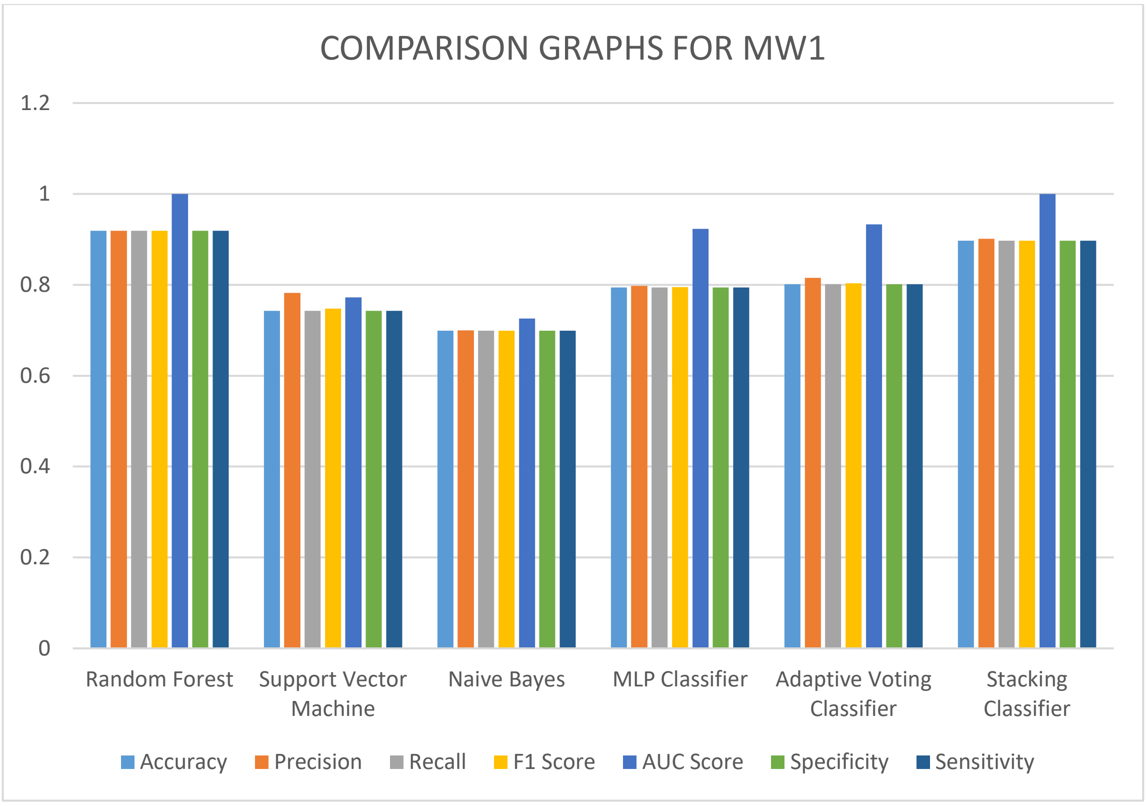


Table.5 Performance Evaluation Metrics for PC1

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

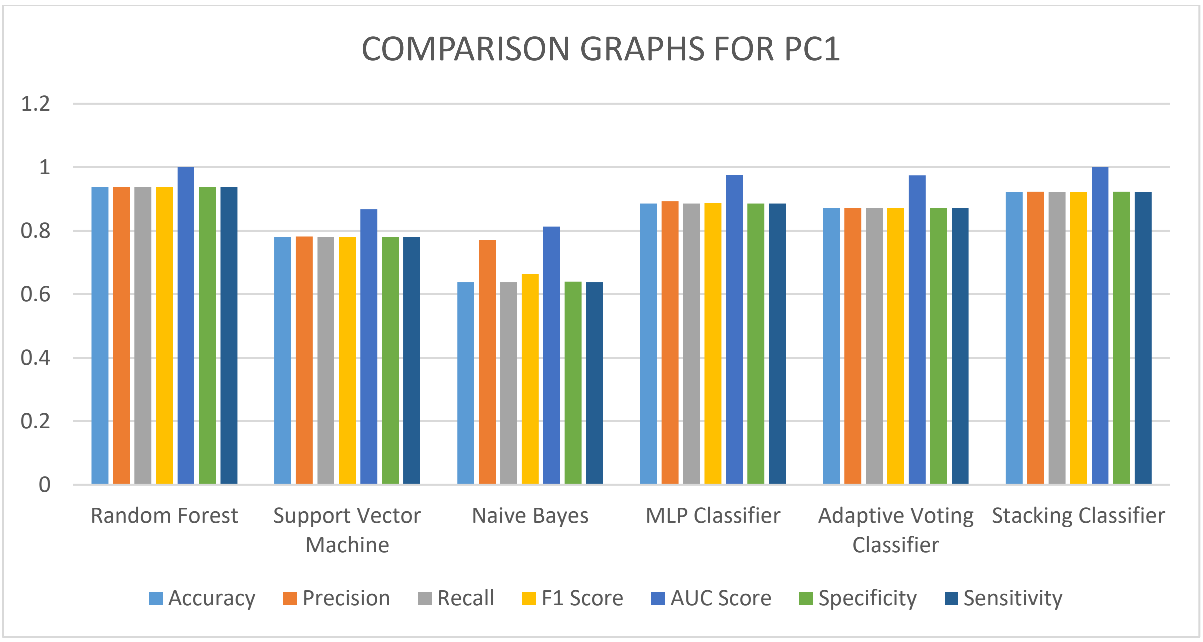


Table.6 Performance Evaluation Metrics for PC3

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

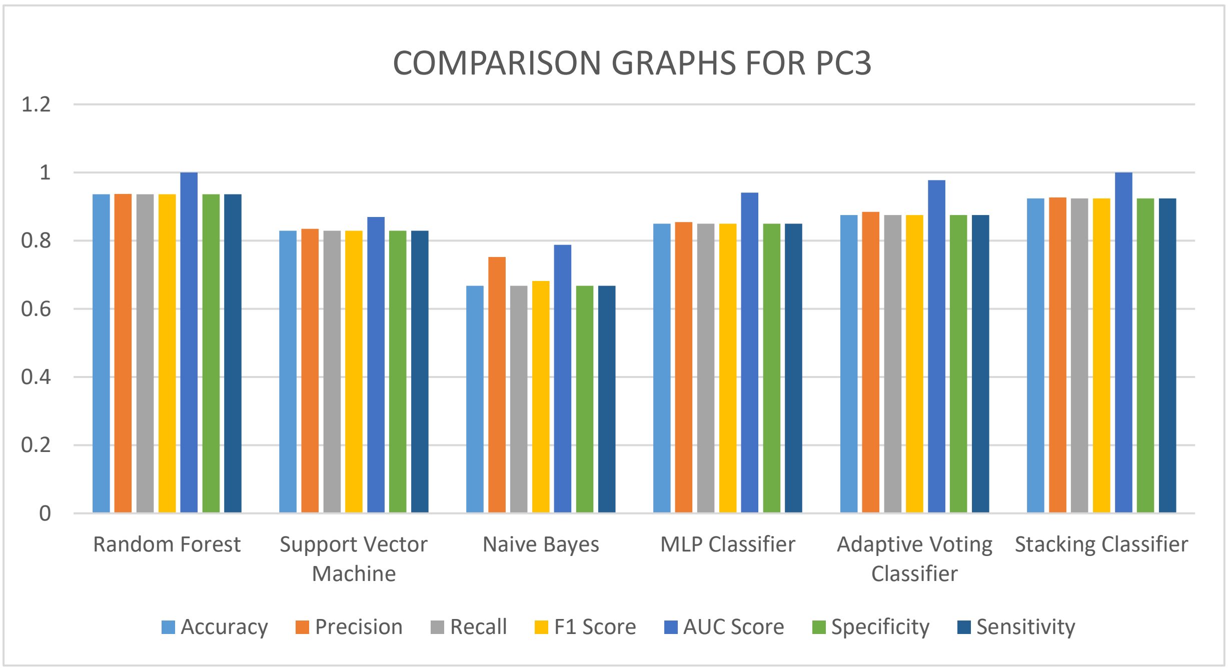
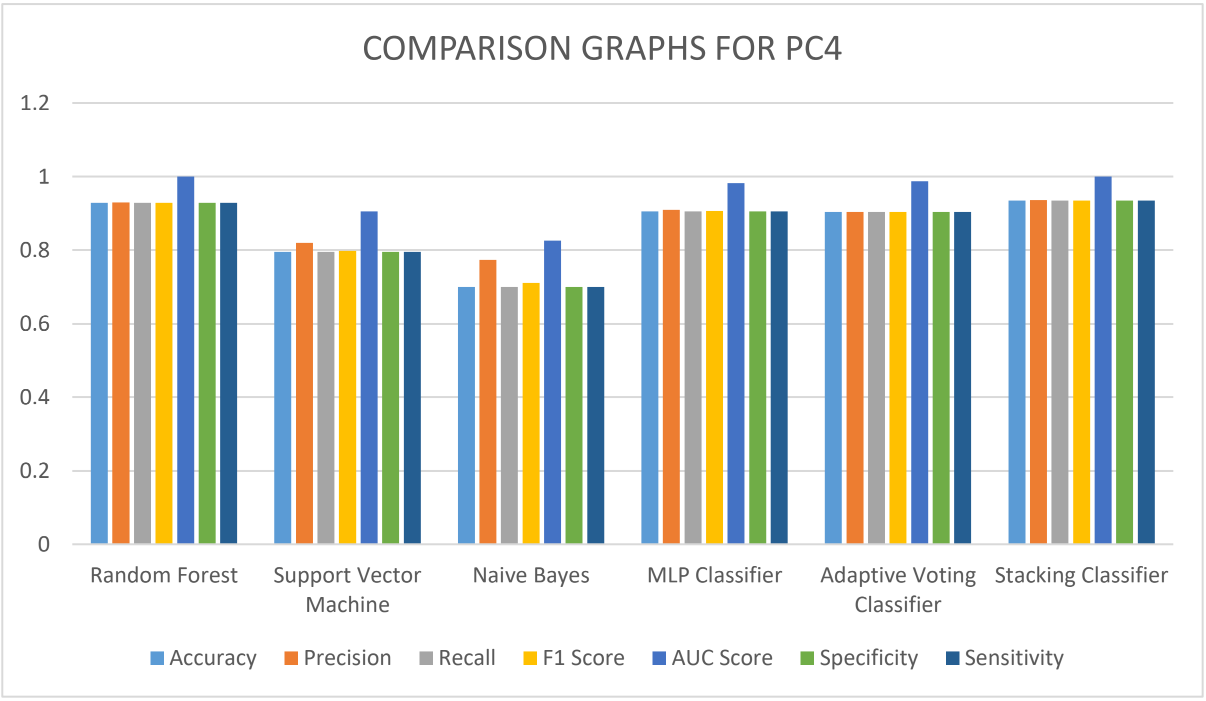


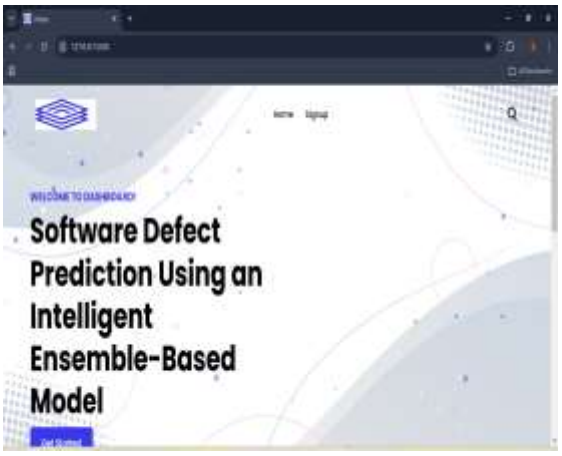
Table.7 Performance Evaluation Metrics for PC4

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



Accuracy is represented in light blue, precision in orange, recall in grey, F1-Score in light yellow, AUC in blue, Specificity in green and Sensitivity in dark blue in Graphs (1 to 7). In comparison to the other models, the Stacking Classifier shows superior performance across all datasets, achieving the highest values. The graphs above visually illustrate these findings.



WELCOME TO DASHBOARD!

Software Defect Prediction Using an Intelligent Ensemble-Based Model

Fig.9 Home Page

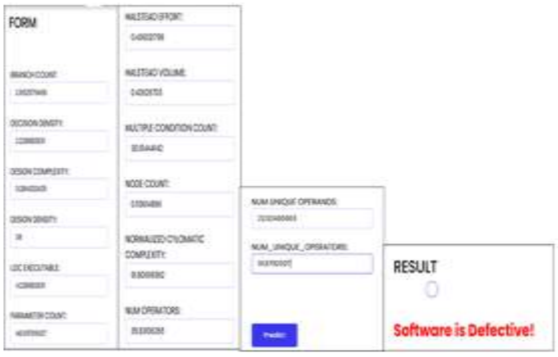
In the above figure 9, this is a user interface dashboard, it is a welcome message for navigating page.

WELCOME TO DASHBOARD!

Software Defect

Fig.10 User input Page

In the above figure 10, this is a user input page, using this user can upload Data from DatasetCM1.



FORM

BRANCH COUNT:	[illegible]
DECISION DENSITY:	[illegible]
DESIGN COMPLEXITY:	[illegible]
DESIGN DENSITY:	[illegible]
LOC EXECUTABLE:	[illegible]
PARAMETER COUNT:	[illegible]
HALSTEAD EFFORT:	[illegible]
HALSTEAD VOLUME:	[illegible]
MULTIPLE CONDITION COUNT:	[illegible]
NODE COUNT:	[illegible]
NORMALIZED CYCLOMATIC COMPLEXITY:	[illegible]
NUM OPERATORS:	[illegible]
NUM UNIQUE OPERANDS:	[illegible]
NUM_UNIQUE_OPERATORS:	[illegible]

RESULT

Software is Defective!

Fig.11 Test result for CM1

In the above figure 11, this is a result screen, in this user will get output.

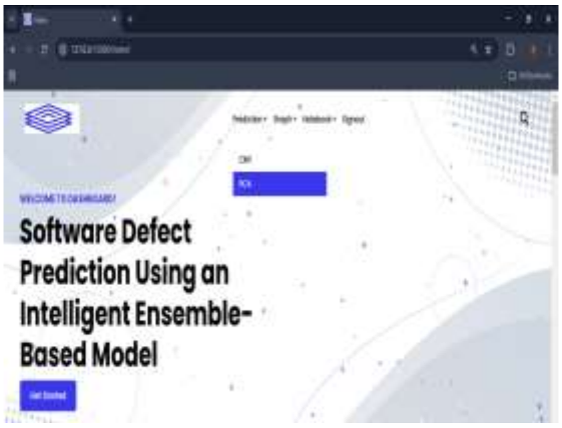
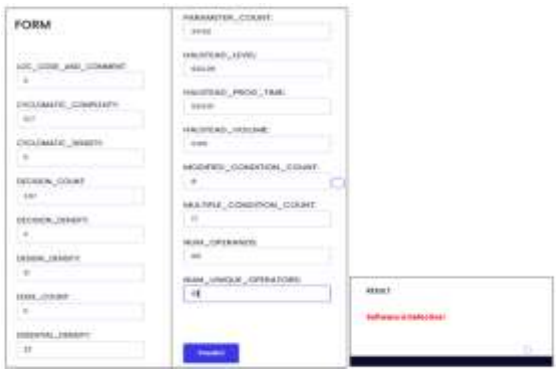

Prediction
Graph
Notebook
Signout
CM1
PC4
WELCOME TO DASHBOARD
Software Defect Prediction Using an Intelligent Ensemble-Based Model
Get Started

Fig.12 User input Page

In the above figure 12, this is a user input page, using this user can upload data from PC4 dataset.



FORM

LOC_CODE_AND_COMMENT:	[illegible]
CYCLOMATIC_COMPLEXITY:	[illegible]
CYCLOMATIC_DENSITY:	[illegible]
DECISION_COUNT:	[illegible]
DECISION_DENSITY:	[illegible]
DESIGN_DENSITY:	[illegible]
EDGE_COUNT:	[illegible]
ESSENTIAL_DENSITY:	[illegible]
PARAMETER_COUNT:	[illegible]
HALSTEAD_LEVEL:	[illegible]
HALSTEAD_PROG_TIME:	[illegible]
HALSTEAD_VOLUME:	[illegible]
MODIFIED_CONDITION_COUNT:	[illegible]
MULTIPLE_CONDITION_COUNT:	[illegible]
NUM_OPERANDS:	[illegible]
NUM_UNIQUE_OPERATORS:	[illegible]

RESULT

Software is Defective!

Fig.13 Test result for PC4

In the above figure 13, this is a result screen, in this user will get output.

5. CONCLUSION

Software defect prediction aims to identify faulty modules before the testing phase, enabling a focus on testing only those modules likely to have defects. An efficient defect prediction model significantly reduces software development costs by optimizing resources during quality assurance activities. Using datasets such as CM1, JM1, MC2, MW1, PC1, PC3, and PC4, the system integrates advanced techniques like SMOTE for addressing class imbalance and Particle Swarm Optimization (PSO) for feature selection, ensuring balanced and relevant input data for training. Among the various approaches evaluated, the Stacking Classifier demonstrated the highest accuracy across all datasets, effectively leveraging diverse learning patterns to enhance prediction capabilities. This combination of techniques and high-performance ensemble learning underscores the importance of identifying defective modules with precision, streamlining testing processes, and contributing to the delivery of high-quality software within reduced timelines and costs.

6. FUTURE SCOPE

Future work can explore the integration of deep learning models, such as CNNs and LSTMs, to capture complex patterns in software defect data. Incorporating explainable AI techniques could enhance the interpretability of predictions, aiding developers in understanding defect causes. Expanding the approach to larger, real-world datasets and dynamic software systems would improve scalability and generalizability. Additionally, combining advanced sampling methods with optimization algorithms may further refine defect prediction performance, supporting cost-effective, high-quality software development in diverse domains.

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.

AUTHOR

Mrs. Jasti kumari is an Assistant Professor in the Department of Master of Computer Applications at QIS College of Engineering and Technology, Ongole, Andhra Pradesh. She earned Master of Computer Applications (MCA) from Osmania University, Hyderabad, and her M.Tech in Computer Science and Engineering (CSE) from Jawaharlal Nehru Technological University, Kakinada (JNTUK). Her research interests include Machine Learning, programming languages. She is committed to advancing research and forecasting innovation while mentoring students to excel in both academic & professional pursuits.