Journal of Information and Organizational Sciences

Volume 50, Number 1 (2026)
Journal homepage: jios.foi.hr

JIOS

DOI: 10.31341/jios.50.1.13
Open Access

UDC 004.415.538
Original Scientific Paper

An Advanced Stacked Ensemble Framework for Enhanced Software Bug Predictions

Sabrina Nasrin Saima[1], Abdus Salam[2*], Mir Md. Kawsur[2] and K. M. Imtiaz-Ud-Din[2]

[1]Department of Computer Science & Engineering, Stamford University Bangladesh, Dhaka, Bangladesh
[2]Department of Computer Science, American International University-Bangladesh, Dhaka, Bangladesh
[*]Correspondence: abdus.salam@aiub.edu

PAPER INFO

Paper history:

Received 03 March 2026
Accepted 29 June 2026

Citation:
Saima, S. N., Salam, A., Kawsur, M. M., & Imtiaz-Ud-Din, K. M. (2026). An advanced stacked ensemble framework for enhanced software bug predictions. In Journal of Information and Organizational Sciences, vol. 49, no. 1, pp. 247-262

ABSTRACT

Software Bug Prediction (SBP) is a critical activity in software engineering that seeks to identify defect-prone modules before release, enabling focused testing and reducing post-release failures. With the greater scale and complexity of software, data-driven prediction techniques are necessary to maintain reliability and limit maintenance costs. Machine learning methods have led to recent advances in SBP by inducing defect patterns from historical data. However, existing models struggle to address class imbalance, feature redundancy, and limited generalization across projects. Furthermore, most previous studies focus on single classifiers, which are unable to capture complex data patterns effectively. These limitations reduce the utility of the real world and make it difficult to maintain consistent cross-project performance. To address these issues, we propose a new stacked ensemble framework that combines three base learners (Random Forest, XGBoost, CatBoost) and a Multi-Layer Perceptron model as a meta-learner. We performed a systematic ablation study on four preprocessing pipelines (SMOTE + RFE, SMOTE + Boruta, ADASYN + RFE, ADASYN + Boruta) to evaluate the state-of-the-art model. Empirical evaluation across ten benchmark datasets from the NASA and PROMISE repositories shows consistent gains over individual classifiers and competitive literature baselines, with the propose framework achieving a maximum accuracy of 96.52% while providing notable improvements in bug-prediction robustness.

Keywords: Software Bug Prediction, Stacked Ensemble, Neural Network, Meta-Model, Data Balancing, NASA Datasets, PROMISE Datasets

1. Introduction

The size and complexity of a software system has grown rapidly, which has greatly raised the possibility that flaws may be introduced during the development phase. Massive functional failures, safety risks, financial losses, and reputational harm can all result from undetected bugs in software. The quality assurance of software is therefore now a major imperative for organizations striving to develop reliable, flexible, and cost-effective products. One of the best means to determine software quality is Software Bug Prediction (SBP), the process of identifying defect-prone modules prior to deployment to employ testing resources in a more efficient manner (Al-Fraihat et al., 2024). So, anticipating software defects is one of the most important steps in the software development phase (Siva et al., 2023).

During the last two decades, a large number of machine learning (ML) methods have been utilized for SBP, ranging from classic classifiers such as Decision Trees and Naive Bayes to more recent algorithms such as Support Vector Machines, Random Forests etc. (Hammouri et al., 2018). In addition to classification, Ensemble Modelling (EM) has become a viable method to improve overall performance by integrating predictions from several machine learning models (Ali et al., 2024). These models perform well; however, their performance is often constrained by several lingering challenges. The most important challenge among them is class imbalance, where the vast majority of non-defective modules are so large in number compared to defective ones that the models are biased in favor of the majority class (Iqbal et al., 2019; Siva et al., 2023). Software datasets also have irrelevant or redundant features, such as noisy or overlapping data, which decrease the accuracy of predictions and increase computational cost (Das et al., 2024). One of the primary reasons why software bug prediction performs poorly is an imbalanced dataset (Tong et al., 2022). Another concern is the lack of generalizability, as models trained on one dataset do not perform well on other projects due to variations in software characteristics, metrics, and defect distributions (Gray et al., 2013). These issues point to the need for more powerful and robust solutions for software defect prediction.

To address such problems, this study proposes a stacked ensemble framework for software bug prediction. The method combines three best-performing base classifiers with a meta-classifier. The framework also has some advanced pre-processing methods, including feature selection and class balancing, to enhance predictive performance and reducing overfitting. To enhance the strength and generalizability of software bug prediction, this work tries to answer three key questions. First, can a stacked ensemble model outperform individual classifiers in software bug prediction? Second, do combined feature selection and class balance techniques achieve measurable improvements over raw datasets? Lastly, can the proposed framework outperform current methods in various project contexts and generalize successfully across different datasets? These are the questions that motivate our study, guiding a systematic evaluation of ten NASA and PROMISE datasets to discover how algorithmic and preprocessing advances interact to impact predictive accuracy and stability in software defect detection. The key contributions of this study are summarized as follows:

- We introduce a new stacked ensemble framework that combines three machine learning models, namely Random Forest, XGBoost, and CatBoost, with Multi-Layer Perceptron (MLP) as a neural network-based meta-classifier for enhanced software bug prediction.
- We perform a first-of-its-kind ablation study to analyze four combinations of pre-processing techniques: SMOTE + RFE, SMOTE + Boruta, ADASYN + RFE, and ADASYN + Boruta, in an attempt to determine which combination of class balancing and feature selection methods provides the optimal performance when combined with the proposed framework.
- We validate the effectiveness of the proposed framework through extensive experiments on ten benchmark datasets (NASA and PROMISE) using multiple evaluation metrics, demonstrating clear improvements over raw datasets.
- Our proposed framework is compared with ten baseline classifiers and the state-of-the-art method, demonstrating high accuracy and good generalization in different software defect prediction scenarios.

The remainder of this paper is structured as follows. Section 2 presents the related work in ensemble learning and software bug prediction. Section 3 describes the proposed methodology and experimental setup. Section 4 provides the results and analysis. Finally, Section 5 concludes the paper and suggests future work.

2. Related Work

Software Bug Prediction has been an area of active research during the last few decades, motivated by increasing software system complexity and increasing demands for dependable, maintainable, and secure software products. Early prediction of bugs is crucial for reducing post-release failures, reducing maintenance costs, and improving customer satisfaction. Rules-based and manual inspection methods have been helpful but are tedious, error-prone, and not scalable. Software bug prediction has been transformed by machine learning, which identifies defect-prone modules by learning patterns from historical metrics. Over the years, researchers have explored a wide range of techniques, from traditional classifiers to advanced deep learning models and hybrid optimization approaches, each of which aims to improve the accuracy and reliability of predictions. This section presents constant challenges, highlights gaps that drive ongoing research, and summarizes significant achievements.

2.1. Traditional Machine Learning Techniques

Machine Learning based bug/defect prediction models utilized algorithms such as Naive Bayes, Decision Tree, Artificial Neural Networks (ANN), Support Vector Machine (SVM), J48, Random Forest, Logistic Regression, AdaboostM1, XGBoost (Ali et al., 2024; Hammouri et al., 2018; Iqbal et al., 2019; Meenakshi & Singh, 2019; Naidu et al., 2022) etc. PROMISE datasets were used to conduct experiments, which confirmed the effectiveness of these models, which were even up to 90% accurate (Hammouri et al., 2018). Similarly, research (Iqbal et al., 2019) examined numerous algorithms using NASA datasets, including MLP, Radial Basis Function (RBF), SVM, and K-Nearest Neighbors (KNN). It found that RBF performed best in a number of instances, closely followed by MLP and SVM. Despite these successes, limitations remain, where models tuned for one dataset often degrade significantly when applied to others that cause performance inconsistency. Also, effect of class imbalance problem where datasets usually have significantly fewer faulty modules compared to non-faulty ones, with models leaning towards the majority class.

2.2. Data Preparation and Balancing Techniques

The quality and quantity of software metrics play a crucial role in determining the accuracy of defect prediction models. Over the years, researchers have used a variety of datasets for software bug prediction. To provide a clearer overview of dataset usage in prior studies, a summary is presented in Table 1. Among the many techniques that have been widely applied in dealing with class imbalance is the Synthetic Minority Oversampling Technique (SMOTE) (Tong et al., 2022) and adaptive variants of it. Adaptive SMOTE (ASMO) (Das et al., 2024), improved the performance of the classifier in modelling minority classes, as indicated in (Rizwan et al., 2017), where the maximum accuracy of 89.4% was achieved with a bagging ensemble on multiple datasets. XGBoost with SMOTE was tuned in another paper (Gupta et al., 2020) to enhance minority detection rates, while author (Ferenc et al., 2020) an introduced a new BugHunter dataset and recommended it, offering temporally precise bug labels, but fewer studies. While pre-processing has been demonstrated to work, no consensus exists on the optimal mixture of cleaning, balancing, and feature selection techniques to yield consistent, cross-dataset performance.

Studies	Dataset(s) Used
(Al-Fraihat et al., 2024; Siva et al., 2023; Iqbal et al., 2019; Das et al., 2024; Gray et al., 2013; Dada et al., 2021; Amit et al., 2024; Wang et al., 2021; Gupta et al., 2020; Rizwan et al., 2017; Siva et al., 2024; Balogun et al., 2021)	NASA
(Siva et al., 2023; Hammouri et al., 2018; Ali et al., 2024; Tong et al., 2022; Wang et al., 2021; Iqbal & Aftab, 2020; Siva et al., 2024)	PROMISE
(Puranik et al., 2016; Khleel & Nehez, 2023; Borandag, 2023)	Eclipse JDT Core
(Khleel & Nehez, 2023)	Bugcatchers Bug
(Khleel & Nehéz, 2023)	GitHub Bug
(Mahfoodh & Obediat, 2020)	Mozilla Core
(Daza, 2025)	Software Defect

Table 1. Summary of Datasets Used in Previous Studies.

2.3. Bug Prediction using Ensemble Learning

Ensemble learning has proven to be a promising method for improving prediction accuracy and generalizability (Wang et al., 2021; Bala et al., 2022). Bagging, boosting, and stacking techniques combine multiple classifiers into one to tap their strength in complementary directions (Sharma et al., 2023). Several ensemble models, including Vote, Bagging, Random Forest, and AdaBoostM1, were optimized using hyper parameter tuning in a study by (Al-Fraihat et al., 2024), which achieved an accuracy of over 81% on the JM1 dataset. Similarly to this, a different study (Dada et al., 2021) used a stacked architecture with Random Forest acting as the meta-learner and KNN, GLMNet, and Linear Discriminant Analysis (LDA) as the base learner, which achieved an average accuracy of 88% on a few chosen datasets. However, most of the ensemble methods (Al-Fraihat et al., 2024; Dada et al., 2021; Borandag, 2023) focus solely on accuracy, passing metrics such as ROC-AUC, precision, recall, and effort-aware metrics, particularly useful in real-world bug prediction contexts. Moreover, most of them address neither all meaningful challenges such as class imbalance, redundant features, nor overfitting in a single framework.

2.4. Hybrid Optimization and Deep Learning Techniques

Recent times have seen the integration of deep learning into SBP. Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) models have shown strong performance, particularly in managing large datasets (Borandag, 2023). Hybrid frameworks further enhance predictability by linking optimization algorithms with a deep learning architecture. Examples include Adaptive Artificial Jelly Optimization with LSTM (Siva et al., 2023) and Adaptive Golden Eagle Optimizer with LSTM (Siva et al., 2024), both of which utilize feature selection to improve learning efficacy and precision. Additional hybrid methods combine dimensionality reduction with ensemble methods, for instance, study (Amit et al., 2024) employed Principal Component Analysis (PCA) with AdaBoost and Random Forest to achieve 91.17% accuracy for different datasets without changing the fold count. Feature selection approaches based on bio-inspired algorithms, such as Spider Wasp Optimization (Das et al., 2024), have also been shown to improve prediction accuracy, although often at the cost of substantial computational overhead. In addition, current studies have shown that deep learning architectures such as CNN and LSTM networks can learn to learn discriminative features from software measures automatically and outperform most traditional machine learning models in defect prediction tasks (Shin et al., 2021). Similarly, a study (Alsghaier & Akour, 2020) integrated the Genetic Algorithm (GA) with a Support Vector Machine (SVM) classifier, showing that hybrid combinations can significantly enhance software fault prediction accuracy across large and small-scale datasets. Despite these advancements, many hybrid and deep learning methods still demand significant computational resources and are prone to overfitting when applied to small datasets, which limits their practical applicability.

To bridge these gaps, our work suggests a stacked ensemble approach that utilizes three very potent base learners, such as XGBoost, CatBoost, and Random Forest, within a Multi-Layer Perceptron (MLP) meta-classifier. The distinguishing feature of this approach is its focus on pre-processing as an end-to-end, carefully considered pipeline. Along with that, we explore how multiple pre-processing strategies can cooperate to increase the precision of bug prediction. By examining these attributes in an ablation study, our findings can provide actionable guidance to software engineers and researchers looking to build more resilient, generalizable bug prediction models. By testing this configuration on ten diverse sets of datasets from the NASA and PROMISE repositories, we aim to build a solution that not only performs well, but also repeatedly, on a variety of software project types

3. Methodology

This section presents our proposed stacked ensemble framework for software defect prediction. We begin by describing the NASA and PROMISE datasets used in this study, followed by the data pre-processing and stratified dataset splitting procedures. The ensemble learning approach is then detailed, including the selection of base classifiers and the meta-model, along with the rationale for their inclusion. We also introduce the ablation study design, which evaluates the effect of different combinations of class balancing and feature selection techniques. Finally, the evaluation metrics used to assess the performance of the proposed study are outlined. An overview of the complete methodology is illustrated in Figure 1.

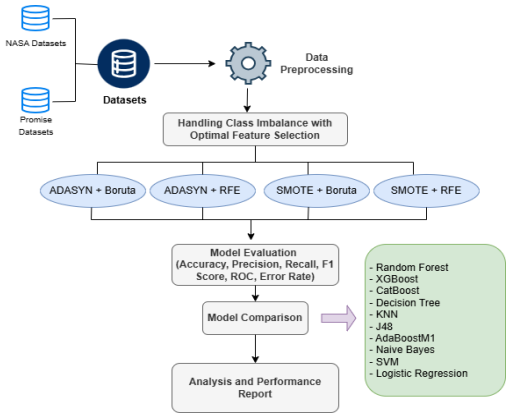


Figure 1. Overall Workflow of the proposed study.

3.1. Dataset

In this study, we used ten publicly available datasets that are widely recognized benchmarks in software defect prediction. Five datasets from the NASA Metrics Data Program (MDP), namely JM1, PC4, MW1, KC3, and MC2 and five from the PROMISE repository, including Camel-1.6, Ant-1.7, Prop-6, Synapse-1.2 and Xalan-2.4, were chosen because they are among the most frequently cited and extensively studied in the software defect prediction literature. The datasets were obtained from well-established public sources. The PROMISE datasets were obtained from the PROMISE'19 Replication Package hosted on Figshare, a curated and DOI-assigned release. The NASA datasets were gathered from the NASA Defect Dataset GitHub repository by Chakkrit Tantithamthavorn, which contains original MDP datasets. These repositories are publicly available.[1] Both of the sources are well-cited in previous research, guaranteeing reproducibility and authenticity of our experiments. Each dataset is composed of software modules described by multiple static attributes, such as size measures, complexity metrics, and coupling indicators. Along with these independent variables, every module is also labeled (Yes or No) with a defect indicator that is the prediction target. A summary of the datasets used in this study, including the number of modules, features, and class distributions before pre-processing, is presented in Table 2.

Source	**NASA**					**PROMISE**				
Project Name	JM1	PC4	KC3	MW1	MC2	Camel-1.6	Ant-1.7	Xalan-2.4	Prop-6	Synapse-1.2
Instances	10878	1458	458	403	161	965	745	723	644	234
Defective	2102	178	43	31	52	188	166	110	61	64
Non-Defective	8776	1280	415	372	109	777	579	613	583	170
Features	22	41	41	41	41	20	20	20	20	20

Table 2. Summary of raw datasets before pre-processing.

3.2. Data Pre-processing

To ensure the input data are clean, balanced, and appropriate for model training, effective data pre-processing is a crucial step in the suggested model for software bug prediction. The pre-processing steps have been designed to systematically handle data inconsistencies, label harmonization, balanced class distributions, select useful features, and ablation studies utilizing four combinations of feature selection and class balancing.

3.2.1. Data Cleaning

The first was that NASA's initial datasets were accessed in the Weka Attribute-Relation File Format (ARFF) format. Therefore, all NASA datasets were converted to the CSV format. Data cleansing is the process of discovering and handling duplicates, null values, and missing entries. Incomplete or missing reading modules were not included as they can contribute bias and noise to the training. To ensure that each software module appears only once in the analysis, duplicate records were also removed, which can be found in publicly available datasets that were also removed. This was done to provide a clean base for further pre-processing.

3.2.2. Label Harmonization

Since NASA and PROMISE datasets both retained defect data in different formats, label harmonization was necessary. The defective labels in the NASA datasets were categorical (Y for defective and N for non-defective). These were converted to numeric form (1 for defective, 0 for non- defective). The defect data in the PROMISE datasets were provided as bug counts. To normalize these to NASA format, we binarized the counts: modules with one or more defects were labeled defective (1), and no-defect modules were labeled as non-defective (0). The conversion gave us a consistent target variable for all datasets.

[1] PROMISE Dataset: https://figshare.com/articles/dataset/Replication_Package_-_PROMISE_19/ 8852084?file=16224071; NASA Dataset: https://github.com/klainfo/NASADefectDataset

3.2.3. Class Balancing

In Class imbalance, defective modules are significantly outnumbered by non-defective modules, which is a typical issue in software defect prediction. This imbalance relies on machine learning models to predict the majority class. To counteract this, we employ two resampling techniques: the Synthetic Minority Oversampling Technique (SMOTE) and Adaptive Synthetic Sampling (ADASYN). SMOTE generates synthetic samples in the feature space of the minority examples, whereas ADASYN tunes its sampling to target hard-to-learn regions, creating more representative synthetic instances. Both methods are employed only on the training dataset, without any inclusion of information from the test dataset.

3.2.4. Feature Selection

To reduce redundancy and improve model interpretability, we employed two feature selection methods. RFE (Recursive Feature Elimination) and Boruta. RFE is a wrapper-based method that removes the least contributory features step by step based on their contribution to model performance. In this study, a Random Forest classifier was used as the estimator. The iteration continues until the top 15 important features are selected. In contrast, Boruta is a random feature selection algorithm whereby all the important features are selected through comparison between the importance of original features and the random forest importance score using shadow features. Features that score higher in significance when compared to their corresponding shadow features are classified as important.

3.2.5. Ablation Study

To contrast the effect of different pre-processing methods, we conducted an ablation study combining class balancing and feature selection in four configurations: SMOTE + RFE, SMOTE + Boruta, ADASYN + RFE, and ADASYN + Boruta. Each configuration was executed on the datasets and validated using the proposed stacked ensemble model. The controlled test allowed us to identify the optimal pre-processing pipeline and provided a better understanding of how balancing and feature selection in combination play a role in model performance. To demonstrate the effect of class balancing and feature selection, Figure 2 illustrates the class distribution after applying four preprocessing combinations (SMOTE + RFE, SMOTE + Boruta, ADASYN + RFE, and ADASYN + Boruta). It can be observed that the applied preprocessing techniques significantly reduce class imbalance, resulting in a more balanced distribution between defective and non-defective instances, where SMOTE-based techniques produce a perfectly balanced distribution, whereas ADASYN-based techniques result in a slight imbalance due to adaptive sampling. For brevity, only JM1 is presented, while the same preprocessing steps were consistently applied to all NASA and PROMISE datasets.

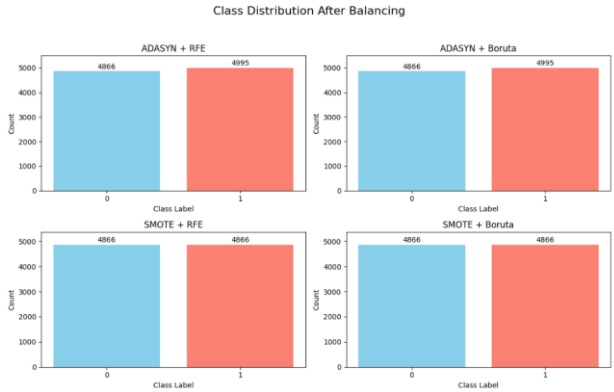


Figure 2. Class Distribution of the Training Dataset after Applying Preprocessing Techniques.

3.2.6. Data Partitioning

The dataset was partitioned into two distinct subsets: training and testing. The allocation of the dataset followed a 70:30 split, with 70% of the data reserved for training and another 30% reserved for testing.

3.3. Experimental Setup

The experimental setting for this study used Python 3.10.2 as the primary programming language within the Google Colab (cloud-based Jupyter Notebook) environment. For model development, we used TensorFlow 2.1.0 and Keras 2.3.1. All experiments were performed on a Linux-based runtime equipped with a cloud-based CPU (~2.20 GHz), 12 GB RAM, and the default Colab hardware configuration. Data pre-processing, machine learning pipeline construction, and visualization were performed using Scikit-learn, Pandas, NumPy, Matplotlib, and Seaborn libraries.

3.4. Evaluation Metrics

To comprehensively evaluate the performance of the proposed stacked ensemble model, some of the most widely accepted metrics were used, including Accuracy, Precision, Recall, F1-Score, ROC-AUC, and Error Rate. Taken collectively, the metrics provide an equitable idea about the model's ability to identify defective modules while minimizing incorrect predictions. They also help to make the evaluation objective and consistent, even when dealing with datasets with imbalances of classes, giving a better representation of the reliability of the model and overall robustness.

3.5. Proposed Framework

In this section, we introduce a new stacked ensemble framework to enhance the reliability and predictive power of software defect prediction models. As shown in Figure 3, our approach operates in two principal layers: training and testing. The training layer is further subdivided into three sequential stages: data preprocessing, base classification, and meta-learning, and the testing layer (prediction stage) predicts the software as defective or non-defective.

3.5.1. Base Classification Step

In the base classification stage, three well-established machine learning models, Random Forest, XGBoost, and CatBoost, are individually trained on the pre-processed training sets. The models were chosen because of their complementary strengths; Random Forest ensures strength against overfitting by means of bootstrap aggregation and random feature selection. XGBoost efficiently fits complex non-linear connections and performs gradient boosting. CatBoost is effective in feature interaction management and is highly efficient even with minimal parameter tuning. Each of the three base learners, generate probabilistic predictions (class probabilities) to being in each of the classes, which were then used as informative inputs to the meta-learner.

3.5.2. Meta-Learner Step

The meta-learning phase uses a Multi-Layer Perceptron (MLP) model as the meta-classifier. To create the meta-features required by the stacking model, 5-fold stratified cross-validation was performed on the training dataset. In each cross-validation iteration, the base models (Random Forest, XGBoost, and CatBoost) were trained on the data subset and used to predict the unseen folds. These out-of-fold (OOF) predictions were collected from the base models and combined to create the meta-features. This approach helps reduce the risk of data leakage and enables the meta-model to learn from more reliable, unbiased predictions. The meta-learner architecture contains one input layer, two hidden layers, and one output layer. The input layer obtains probability predictions from the three input features of Random Forest, XGBoost, and CatBoost. There are two hidden layers that are fully connected (16 and 8 nodes) with ReLU functions, which is an activation function, and a dropout layer was used to prevent overfitting. The output layer consists of a single node that uses a sigmoid activation function to determine the highest possible probability that a software module is defective. During training, the meta-learner MLP was minimized by the Adam optimizer using a binary cross-entropy loss function because it is well suited for binary classification problem. The model was trained up to 15 epochs with a batch size of 32 to achieve stable convergence. To prevent overfitting and enhance generalization, an early stopping feature was employed by monitoring the validation loss with a patience of 5 epochs, restoring the best-performing model weights when stopped with L2 regularization factor of 0.001 and a dropout of 0.5 were employed. The dataset was split with 70:30 ratio where this splitting ratio ensured consistency with previous work in the area while providing a well-defined setup for comparing the various preprocessing techniques. A stratified 10-fold cross-validation was also done to test for the reliability of the model, the

holdout splitting technique remained the main approach for evaluation throughout the experiment. All of these hyper-parameters contributed to stable training, effective optimization, and improved generalization performance of the stacked ensemble model.

3.5.3. Prediction Stage

In the prediction or testing phase, the new instances are first passed to the trained base classifiers to obtain their class probabilities can be retrieved. These are then combined and fed into the MLP meta-model, which generates the final binary classification, that is, whether each module is defective or non-defective. Thus, the stacked ensemble strategy proposed in this study has certain advantages. As the ensemble effectively employ the best attributes of three different base learners most efficiently, the ensemble is not only more predictive, but also more robust and stable than any single classifier.

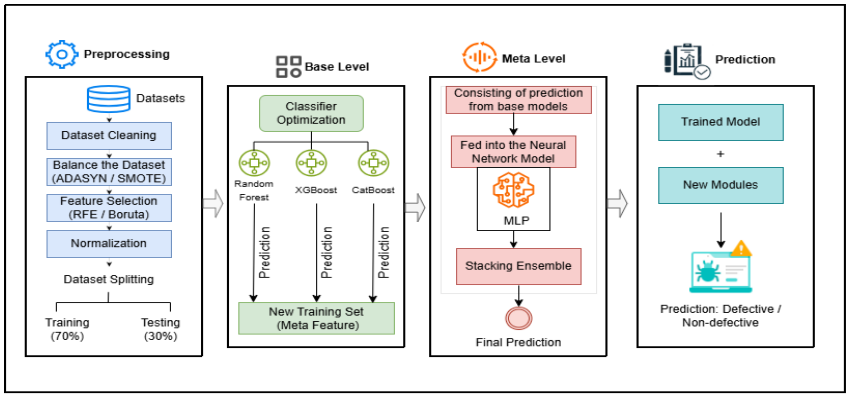


Figure 3. Proposed Stacked Ensemble Model Architecture.

4. Experimental Result Analysis

This section presents the experimental results of the proposed stacked ensemble model and compares its performance with ten baseline machine learning classifiers on ten datasets (JM1, KC3, MC2, MW1, PC4 and Ant-1.7, Camel-1.6, Synapse-1.2, Prop-6, Xalan-2.4). Performance is measured using a range of different metrics like accuracy, precision, recall, f1-score, ROC-AUC, and error rate to provide a highlight overview of the model's effectiveness. The results are organized into three sections that are presented below.

4.1. Performance Analysis of Proposed Bug Prediction Model

Firstly, we show the results of an ablation study in which we compare the impact of four different combinations of pre-processing on the performance of the model. Although accuracy has been employed as an initial metric because the proposed framework is designed for software bug prediction, where overall prediction correctness is an important consideration, and accuracy is commonly used in comparing various software defect prediction that is widely used in prior studies, it is not entirely indicative of performance in imbalanced datasets. Therefore, additional evaluation metrics, including precision, recall, F1-measure, and ROC-AUC, were also considered to ensure the quality of the models. In several cases, despite comparatively lower accuracy, classifiers achieved higher recall or ROC-AUC values, which signifies better ability in detecting defects in modules.

Technique Combination	Accuracy (%)	F1 Score	Recall	Precision	ROC-AUC	Error Rate (%)	Dataset
ADASYN + RFE	72.59	0.41	0.43	0.38	0.64	27.41	**JM1**
ADASYN + Boruta	72.31	0.40	0.42	0.37	0.64	27.69	
SMOTE + RFE	72.51	**0.41**	**0.44**	0.38	0.64	27.49	
SMOTE + Boruta	**72.81**	0.41	0.43	**0.38**	**0.65**	**27.19**	
ADASYN + RFE	**77.37**	**0.41**	0.42	**0.42**	0.66	**22.63**	**KC3**

ADASYN + Boruta	76.79	0.38	**0.44**	0.39	**0.68**	23.21	
SMOTE + RFE	74.74	0.31	0.33	0.31	0.60	25.26	
SMOTE + Boruta	76.79	0.32	0.35	0.36	0.65	23.21	
ADASYN + RFE	55.19	**0.51**	**0.68**	0.38	0.64	44.81	**MC2**
ADASYN + Boruta	54.68	0.36	0.39	**0.50**	0.52	45.32	
SMOTE + RFE	**58.85**	0.44	0.52	0.44	**0.66**	**41.15**	
SMOTE + Boruta	53.65	0.45	0.57	0.42	0.55	46.35	
ADASYN + RFE	85.78	0.36	0.40	0.33	0.36	14.22	**MW1**
ADASYN + Boruta	**87.34**	0.39	0.38	0.46	0.39	**12.66**	
SMOTE + RFE	86.97	0.39	**0.40**	0.40	0.63	13.03	
SMOTE + Boruta	85.78	**0.39**	0.38	**0.47**	**0.69**	14.22	
ADASYN + RFE	89.51	0.63	0.67	0.62	0.88	10.49	**PC4**
ADASYN + Boruta	**90.75**	0.66	**0.71**	**0.65**	**0.89**	**9.25**	
SMOTE + RFE	89.75	0.63	0.66	0.62	0.88	10.25	
SMOTE + Boruta	90.52	**0.67**	0.68	0.65	0.86	9.48	

Table 3. Ablation study results for NASA datasets using four combinations techniques (10-fold CV).

Technique Combination	Accuracy (%)	F1 Score	Recall	Precision	ROC-AUC	Error Rate (%)	Dataset
ADASYN + RFE	84.52	0.28	0.22	0.39	0.64	15.48	**Ant-1.7**
ADASYN + Boruta	**85.11**	0.27	0.23	**0.40**	**0.65**	**14.89**	
SMOTE + RFE	84.23	0.27	0.23	0.39	0.63	15.77	
SMOTE + Boruta	84.22	**0.29**	**0.25**	0.38	0.61	15.78	
ADASYN + RFE	85.53	0.19	0.15	0.28	0.53	14.47	**Camel-1.6**
ADASYN + Boruta	86.45	0.24	0.19	**0.39**	0.57	13.55	
SMOTE + RFE	**86.79**	**0.25**	**0.19**	0.37	**0.60**	**13.21**	
SMOTE + Boruta	86.33	0.22	0.17	0.36	0.59	13.67	
ADASYN + RFE	72.67	0.51	0.57	0.48	0.74	27.33	**Synapse-1.2**
ADASYN + Boruta	**73.93**	0.59	0.65	0.55	0.76	**26.07**	
SMOTE + RFE	72.81	**0.60**	0.65	**0.61**	**0.77**	27.19	
SMOTE + Boruta	72.79	0.59	**0.66**	0.60	0.75	27.21	
ADASYN + RFE	76.93	**0.24**	**0.30**	**0.22**	0.62	23.07	**Prop-6**
ADASYN + Boruta	75.73	0.20	0.26	0.18	0.59	24.27	
SMOTE + RFE	**77.65**	0.21	0.25	0.20	**0.64**	**22.35**	
SMOTE + Boruta	74.29	0.20	0.29	0.17	0.57	25.71	
ADASYN + RFE	84.64	0.20	**0.19**	0.22	0.59	15.36	**Xalan-2.4**
ADASYN + Boruta	83.24	0.17	0.16	0.19	0.61	16.76	
SMOTE + RFE	85.11	0.18	0.15	0.22	0.61	14.89	
SMOTE + Boruta	**85.69**	**0.21**	0.16	**0.27**	**0.63**	**14.31**	

Table 4. Ablation study results for PROMISE datasets using four combinations techniques (10-fold CV).

Technique Combination	Accuracy (%)	F1 Score	Recall	Precision	ROC-AUC	Error Rate (%)	Dataset
ADASYN + RFE	**83.58**	**0.85**	**0.89**	0.81	**0.87**	**16.42**	**JM1**
ADASYN + Boruta	81.68	0.83	0.87	0.79	0.86	18.32	
SMOTE + RFE	83.49	0.84	0.87	**0.82**	0.86	16.51	
SMOTE + Boruta	80.96	0.82	0.84	0.79	0.85	19.04	
ADASYN + RFE	83.82	0.83	0.76	0.90	0.88	16.18	**KC3**
ADASYN + Boruta	80.88	0.82	0.85	0.78	0.85	19.12	
SMOTE + RFE	83.82	0.85	0.91	0.79	0.91	16.18	
SMOTE + Boruta	**92.65**	**0.93**	**0.91**	**0.94**	**0.93**	**7.35**	
ADASYN + RFE	80.00	0.81	0.88	0.75	0.80	20.00	**MC2**
ADASYN + Boruta	85.71	0.86	0.88	0.83	0.84	14.29	

SMOTE + RFE	**91.43**	**0.91**	**0.94**	**0.89**	**0.90**	**8.57**	
SMOTE + Boruta	80.00	0.80	0.82	0.78	0.86	20.00	
ADASYN + RFE	89.47	0.89	0.89	0.89	0.91	10.53	**MW1**
ADASYN + Boruta	91.58	0.91	0.89	0.93	0.94	8.42	
SMOTE + RFE	91.67	0.92	**0.96**	0.88	0.93	8.33	
SMOTE + Boruta	**93.75**	**0.94**	0.94	**0.94**	**0.96**	**6.25**	
ADASYN + RFE	95.83	0.96	0.98	0.94	0.97	4.17	**PC4**
ADASYN + Boruta	96.25	0.96	**0.99**	0.94	**0.98**	3.75	
SMOTE + RFE	94.68	0.95	0.98	0.92	0.97	5.32	
SMOTE + Boruta	**96.52**	**0.97**	0.98	**0.95**	0.97	**3.48**	

Table 5. Ablation study results for NASA datasets using four combinations techniques (70:30 split).

Technique Combination	Accuracy (%)	F1 Score	Recall	Precision	ROC-AUC	Error Rate (%)	Dataset
ADASYN + RFE	93.98	0.94	0.95	0.91	0.96	6.02	**Ant-1.7**
ADASYN + Boruta	**94.58**	**0.95**	**0.95**	**0.94**	0.96	**5.42**	
SMOTE + RFE	93.29	0.93	0.95	0.92	0.95	6.71	
SMOTE + Boruta	94.51	0.95	0.95	0.94	**0.97**	5.49	
ADASYN + RFE	**95.87**	**0.94**	**0.94**	**0.97**	**0.97**	**4.13**	**Camel-1.6**
ADASYN + Boruta	93.58	0.93	0.90	0.97	0.97	6.42	
SMOTE + RFE	93.15	0.93	0.92	0.94	0.95	6.85	
SMOTE + Boruta	93.61	0.94	0.93	0.94	0.96	6.39	
ADASYN + RFE	79.55	0.78	0.73	0.84	0.82	20.45	**Synapse-1.2**
ADASYN + Boruta	**81.82**	**0.82**	**0.82**	0.82	**0.85**	**18.18**	
SMOTE + RFE	75.00	0.72	0.64	0.82	0.79	25.00	
SMOTE + Boruta	**81.82**	0.81	0.77	**0.85**	0.78	18.18	
ADASYN + RFE	83.33	0.82	0.76	0.89	0.90	16.67	**Prop-6**
ADASYN + Boruta	82.35	0.84	0.94	0.76	0.92	17.65	
SMOTE + RFE	**91.18**	**0.91**	0.90	**0.92**	0.93	**8.82**	
SMOTE + Boruta	88.24	0.89	**0.98**	0.82	**0.97**	11.76	
ADASYN + RFE	89.94	**0.91**	**0.98**	0.85	0.93	10.06	**Xalan-2.4**
ADASYN + Boruta	89.94	0.90	0.94	0.87	0.93	10.06	
SMOTE + RFE	**90.00**	0.90	0.94	**0.87**	0.92	**10.00**	
SMOTE + Boruta	89.41	0.90	0.94	0.86	**0.93**	10.59	

Table 6. Ablation study results for PROMISE datasets using four combinations techniques (70:30 split).

Tables 5 and 6, present the results of the proposed stacked ensemble model on each of the ten datasets from the NASA and PROMISE repositories respectively, using 70:30 splits. For each dataset, we performed an ablation study by evaluating four pre-processing combinations: ADASYN + RFE, ADASYN + Boruta, SMOTE + RFE, and SMOTE + Boruta. These tables present the performance of different preprocessing combinations, allowing comparative evaluation across multiple metrics for each dataset. SMOTE-based techniques, SMOTE + Boruta and SMOTE + RFE, were superior to most datasets, and PC4 achieved the highest accuracy of 96.52% using SMOTE + Boruta. The SMOTE + RFE model performed better when calculating the accuracy, while the recall rate using SMOTE + Boruta reached 98%. This observation is also seen for MW1, where the SMOTE + RFE model produced a recall score of 96%, indicating the robustness of the framework. ADASYN-combination pairs also performed well, especially giving the higher results in JM1 (ADASYN + RFE) and Camel-1.6 (ADASYN + RFE with 95.87% accuracy). It is clear that SMOTE tends to perform better through balancing the synthetic samples generated, where ADASYN also works well on certain datasets. The ablation study shows that preprocessing choices are data-dependent, with SMOTE + Boruta outperformed overall. The 10-fold cross-validation technique was also used, and the results are illustrated in Tables 3 and 4. Compared to the 70:30 splits, the performance obtained using 10-fold CV was generally lower for several datasets, particularly the smaller ones. This difference can be due to the variability in data partitioning and the smaller training dataset size within each fold. While the 70:30 split produced comparatively higher results in the present study, the 10-fold cross-validation may provide a more balanced evaluation of model performance.

To provide an intuitive overview of the ablation study, Figure 4 illustrates the highest accuracy achieved for each dataset, selected from the four combinations of feature selection and class balance given in Tables 5 and 6 for each dataset where the SMOTE + Boruta combination technique worked best for PC4 (NASA datasets) and ADASYN + RFE for Camel-1.6 (PROMISE datasets). By showing which pre-processing methods worked best for each dataset, this combined overview allows an efficient evaluation of the effectiveness of different pre-processing methods. The final result for the particular dataset was determined taking the maximum accuracy achieved in each instance; this provided the basis for subsequent comparisons with the proposed framework. The highest performance was shown for PC4 (96.52%) with SMOTE + Boruta, followed by Camel-1.6 (95.87%) with ADASYN + RFE, implying that the SMOTE-based methods perform better compared to ADASYN for NASA datasets, while ADASYN performs similarly on selected PROMISE datasets. The horizontal bar chart represents the best performing datasets, with PC4 and Camel-1.6 shown in orange to emphasize the highest accuracies, while the remaining datasets are displayed in sky blue.

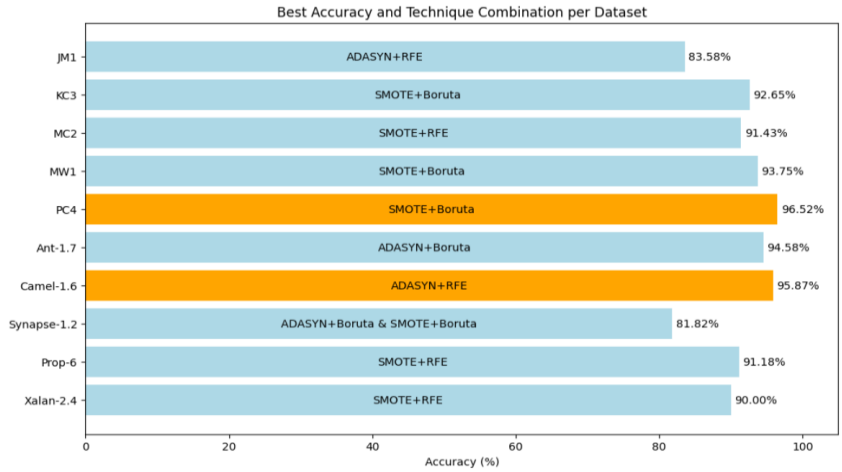
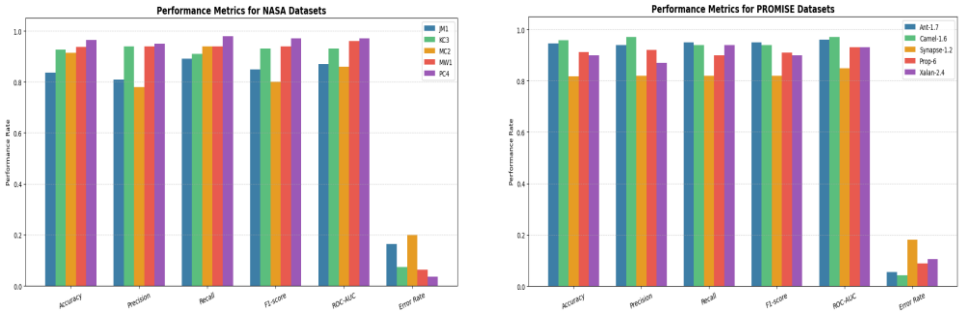


Figure 4. Comparative best Accuracy of the framework across Different Technique Combination.



(a) (b)

Figure 5. Performance Metrics of the Proposed Framework. (a) NASA Datasets (b) PROMISE Datasets.

To provide a comprehensive comparison of the proposed stacked ensemble model with both datasets, Figure 5 demonstrates the bar chart plots of relevant performance metrics, including Accuracy, Precision, Recall, F1-score, ROC-AUC, and Error Rate. From the figure, it is clear that the proposed stacked ensemble framework has proven its consistency by performing very well on NASA and PROMISE datasets, especially with respect to ROC-AUC and F1-score. Although some minor variations have been observed in precision and error rates, almost all other measures exhibit equally good performance on all datasets. For each dataset, the different combinations of preprocessing and feature selection techniques were evaluated, and the higher-performing combination was selected with respect to multi-metric evaluation. These top-performing results are displayed in Figure 5 for each dataset.

4.2. Performance Analysis of Proposed Framework with Base Models

Next, we compare the performance of the proposed stacked ensemble model with ten baseline classifier models. Random Forest, XGBoost, CatBoost, Decision Tree, KNN, J48, AdaBoostM1, Naive Bayes, SVM, and Logistic Regression. The comparison indicates the effectiveness of the proposed method in predictive strength and generalization across different datasets. Additionally, we presented the results before and after pre-processing for the three base classifiers employed in the framework.

Datasets	Proposed Framework	Random Forest	XGBoost	CatBoost	Decision Tree	KNN	J48	Ada-BoostM1	Naïve Bayes	SVM	Logistic Regression
JM1	**83.58**	82.77	80.89	79.93	72.88	71.47	73.08	73.08	55.62	57.16	62.84
KC3	**92.65**	88.24	89.71	89.71	79.41	61.76	86.76	86.76	60.29	54.41	80.88
MC2	**91.43**	82.86	74.29	82.86	68.57	57.14	71.43	71.43	65.71	66.67	65.71
MW1	**93.75**	92.19	90.53	91.58	86.32	69.47	88.42	88.42	82.11	77.89	93.68
PC4	**96.52**	94.79	95.21	95.63	90.83	68.13	90.62	90.62	75.83	59.17	86.88
Ant-1.7	**94.58**	94.58	93.29	94.18	87.20	79.27	89.02	89.02	70.73	70.73	75.61
Camel-1.6	**95.87**	93.61	93.15	94.06	89.95	72.60	89.04	89.04	63.01	54.34	74.89
Synapse-1.2	**81.82**	79.55	79.55	81.82	65.91	65.91	68.18	68.18	75.00	75.00	75.00
Prop-6	**91.18**	89.22	91.16	90.20	83.33	77.45	84.31	84.31	61.76	50.98	63.73
Xalan-2.4	90.00	92.30	91.72	**92.31**	77.51	81.66	79.88	79.88	65.68	71.60	75.15

Table 7. Accuracy (%) comparison of the proposed framework and base classifiers for different datasets.

Table 8 presents a comparative analysis between the base classifiers of the performance of the proposed framework (Random Forest, XGBoost and CatBoost) in raw datasets and their performance after applying the proposed preprocessing pipeline, alongside the results of the proposed stacked ensemble model. The proposed stacked ensemble model is evaluated only after preprocessing, since the meta-learning stage was explicitly designed to work on balanced (ADASYN/SMOTE) and feature-refined (RFE/Boruta) model were used determined through ablation studies. The improvements are clear across almost all datasets. Significant increases in accuracy of 5.59% and 9.60% for JM1 and KC3 demonstrate that addressing class imbalance and removing irrelevant features result in better classification models. The most significant result is from the MC2 dataset, with 19.64% improvement, highlighting the role of preprocessing for strongly imbalanced datasets, one of the core contributions of this study. The only small drop is seen for Prop-6 (-0.22%), which is likely due to its already balanced class distribution, where the selection of features may have removed slightly useful predictors. This comparison is informative as it demonstrates why pre-processing should be done before stacking. It shows that the suggested preprocessing steps continually improve the performance of the given classifiers, justifying the strength of our preprocessing procedures and enabling the final stacked ensemble to produce more accurate and generalizable predictions on benchmark datasets.

Datasets	After Preprocess				Before Preprocess			Improvement
	Proposed Framework	Random Forest	XGBoost	CatBoost	Random Forest	XGBoost	CatBoost	
JM1	**83.58**	82.77	80.89	79.93	77.25	76.98	77.99	+5.59%
KC3	**92.65**	88.24	89.71	89.71	74.58	83.05	81.36	+9.60%
MC2	**91.43**	82.86	74.29	82.86	65.79	71.79	68.42	+19.64%
MW1	**93.75**	92.19	90.53	91.58	91.74	90.08	91.74	+2.01%

PC4	**96.52**	94.79	95.21	95.63	91.09	90.41	90.84	+5.43%
Ant-1.7	**94.58**	94.58	93.29	94.18	86.67	87.41	86.67	+7.17%
Camel-1.6	**95.87**	93.61	93.15	94.06	85.23	85.80	87.50	+8.37%
Synapse-1.2	**81.82**	79.55	79.55	81.82	78.72	76.60	80.85	+0.97%
Prop-6	**91.18**	89.22	91.16	90.20	89.92	91.40	90.10	-0.22%
Xalan-2.4	90.00	92.30	91.72	**92.31**	89.21	88.49	87.77	+0.79%

Table 8. Accuracy (%) comparison of the stacked ensemble framework and base classifiers before and after preprocessing for different datasets.

Overall, the results are determinate that the use of feature selection and class balancing not only improves the performance of the basic classifiers but also strengthens the suggested stacked ensemble, which, after preprocessing, has the highest accuracy on nearly all of the datasets. This validates that the best accurate software defect prediction technique includes an adequate pre-processing step.

4.3. Comparative SOTA (state-of-the-art) Study with Proposed Framework

Finally, we perform a benchmarking against state-of-the-art (SOTA) methods reported in the literature on the same datasets. This demonstrates that our method not only performs competitively compared to individual machine learning models but also achieves promising results under the present experimental setup, relative to recently published approaches, indicating its potential for practical software defect prediction applications.

Table 9 compares the accuracy of our proposed stacked ensemble model with some existing work on the NASA and PROMISE datasets. Although the proposed study demonstrates competitive performance across all evaluated datasets, direct comparison with previous studies must be interpreted with caution due to the differences in methodology. Differences in dataset preprocessing, feature engineering, sampling strategies, and evaluation settings, etc. could affect the obtained results. Therefore, the comparative analysis intended to give a general indication of the performance level achieved. For the JM1 dataset, the existing works have reported accuracies ranging from 62.78% (Iqbal & Aftab, 2020) to 81.95% (Al-Fraihat et al., 2024), while the proposed framework achieves 83.58% under the present experimental setup. In the same way, on KC3, previous studies reported a highest accuracy of 86.84% (Dada et al., 2021), whereas our model achieves 92.65% in the current evaluation protocol. In MC2 and MW1, our proposed framework attains accuracies of 91.43% and 93.75%, respectively, which are competitive with or higher than previously reported results of 75.05% (Amit et al., 2024) and 89.33% (Ali et al., 2024).

Study	**JM1**	**KC3**	**MC2**	**MW1**	**PC4**	**Ant-1.7**	**Camel-1.6**	**Synapse-1.2**	**Prop-6**	**Xalan-2.4**
(Al-Fraihat et al., 2024)	81.95	-	-	-		-	-	-	-	-
(Siva et al., 2023)	80.68	-	-	-	-	88.82	87.66	-	-	67.84
(Ali et al., 2024)	79.12	-	68.42	89.33	87.14	-	-	-	-	-
(Iqbal et al., 2019)	73.96	-	64.86	82.66	86.08	-	-	-	-	-
(Dada et al., 2021)	81.11	86.84	-	-	-	-	-	-	-	-
(Amit et al., 2024)	-	-	75.05	-	-	-	-	-	-	-
(Iqbal & Aftab, 2020)	62.78	-	62.16	77.33	74.80	-	-	-	-	-
(Iqbal & Aftab, 2020)	80.68	-	-	-	-	88.82	87.66	-	-	67.84
(Mcmurray & Sodhro, 2023)	-	-	-	-	-	92.06	92.22	-	-	-
(Abubakar et al., 2023)	-	-	-	-	-	83.20	87.80	-	-	-

(Alsghaier & Akour, 2020)	-	90.35	67.10	91.83	88.21	-	-	71.40	-	-
Proposed Framework	**83.58**	**92.65**	**91.43**	**93.75**	**96.52**	**94.58**	**95.87**	**81.82**	**91.18**	**90.00**

Table 9. Comparison of classification accuracy (%) of the proposed stacked ensemble framework with previous studies on NASA and PROMISE datasets.

In the case of PROMISE datasets, the same trend is observed. In Ant-1.7 and Camel-1.6, previous studies reported optimal accuracies of 88.82% (Siva et al., 2023) and 87.80% (Siva et al., 2024), while our model reaches 94.58% and 95.87%, respectively. In particular, for Prop-6, we were unable to find the values of published accuracy metrics in the reviewed studies, but other evaluation metrics were applied. Thus, our result fills this gap and can act as a baseline for future comparative studies. Our accuracy of 91.18% in Prop-6 is a good starting point for future studies. Generally, these results demonstrate that using a stacked ensemble not only improves prediction performance on individual datasets, but also generalizes well to a very diverse set of software projects, and our approach is a proper reference point for future work.

5. Conclusion & Future Work

This study presents a stacked ensemble based approach to software bug prediction (SBP), aiming to improve software quality and reduce development costs. The methodology combines the predictive strengths of Random Forest, XGBoost, and CatBoost as base classifiers, with a neural network meta-model to enhance decision-making. Extensive pre-processing steps, including data balancing using ADASYN, feature selection with RFE (Recursive Feature Elimination), were employed to ensure data quality. The model was evaluated on five NASA datasets and five PROMISE datasets, and the results demonstrate that the proposed ensemble approach outperforms individual classifiers in accuracy, precision, recall, and F1 score. The best performance is obtained for PC4 using SMOTE + Boruta at 96.52%, and Camel-1.6 using ADASYN + RFE at 95.87%. The results showed that the performance of the combination using SMOTE was higher than the performance of the combination using ADASYN for the NASA datasets, but the performance was comparable for the selected PROMISE datasets. Most noteworthy is the improvement obtained for the MC2 dataset, where the improvement is as high as 19.64%, thus emphasizing the importance of pre-processing for imbalanced datasets that is one of the major contribution of this work. Although the framework achieves promising results, further research can explore the integration of other advanced machine learning techniques, such as deep learning architectures or transformers, to enhance predictive performance. Additionally, testing the model on larger and more diverse datasets from various domains could generalize its applicability. Finally, incorporating real-time prediction mechanisms and explainable AI (XAI) techniques would increase the usability of the model in practical software development environments.

References

Jardas Anonic, J., Srok, A. & Vretenar, N. (2024). Blended Learning in the Spotlight of Educational Management – A Scientometric Analysis, 48(2), 279-294.

Abubakar, H., Umar, K., Auwal, R., Muhammad, K., & Yusuf, L. (2023). Developing a machine learning-based software fault prediction model using the improved whale optimization algorithm. Engineering Proceedings, 56(1), 334. https://doi.org/10.3390/ASEC2023-16307.

Albattah, W., & Alzahrani, M. (2024). Software defect prediction based on machine learning and deep learning techniques: An empirical approach. AI, 5(4), 1743–1758. https://doi.org/10.3390/ai5040086.

Al-Fraihat, D., Sharrab, Y., Al-Ghuwairi, A. R., Alshishani, H., & Algarni, A. (2024). Hyper-parameter optimization for software bug prediction using ensemble learning. IEEE Access, 12, 51869–51878. https://doi.org/10.1109/ACCESS.2024.3380024.

Ali, M., Mazhar, T., Arif, Y., Al-Otaibi, S., Ghadi, Y. Y., Shahzad, T., Khan, M. A., & Hamam, H. (2024). Software defect prediction using an intelligent ensemble-based model. IEEE Access, 12, 20376–20395.

Alsghaier, H., & Akour, M. (2020). Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier. Software: Practice and Experience, 50(4), 407–427. https://doi.org/10.1002/spe.2784.

Amit, K. S., Aditya, K. N., Ankita, N., & Debasish, P. (2024). A NASA dataset based automated software bug prediction model using feature reduction techniques. International Journal of Engineering Research & Technology (IJERT), 13(2), 1–6. https://doi.org/10.17577/IJERTV13IS020071.

Balogun, A. O., Basri, S., Mahamad, S., Abdulkadir, S. J., Capretz, L. F., Imam, A. A., Almomani, M. A., Adeyemo, V. E., & Kumar, G. (2021). Empirical analysis of rank aggregation-based multi-filter feature selection methods in software defect prediction. Electronics, 10(2), 179. https://doi.org/10.3390/electronics10020179.

Borandag, E. (2023). Software fault prediction using an RNN-based deep learning approach and ensemble machine learning techniques. Applied Sciences, 13(3), 1639. https://doi.org/10.3390/app13031639.

Dada, E. G., Oyewola, D. O., Joseph, S. B., & Duada, A. B. (2021). Ensemble machine learning model for software defect prediction. Advances in Machine Learning and Artificial Intelligence, 2, 11–21.

Das, H., Das, S., Gourisaria, M. K., Khan, S. B., Almusharraf, A., Alharbi, A. I., & Mahesh, T. R. (2024). Enhancing software fault prediction through feature selection with spider wasp optimization algorithm. IEEE Access, 12, 20376–20395. https://doi.org/10.1109/ACCESS.2024.3435333.

Daza, A. (2025). Software defect prediction based on a multiclassifier with hyperparameters: Future work. Results in Engineering, 25, 104123. https://doi.org/10.1016/j.rineng.2025.104123.

Ferenc, R., Gyimesi, P., Gyimesi, G., Tóth, Z., & Gyimóthy, T. (2020). An automatically created novel bug dataset and its validation in bug prediction. Journal of Systems and Software, 169, 110691.

Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2013). The misuse of the NASA metrics data program data sets for automated software defect prediction. In Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2013) (pp. 96–103). https://doi.org/10.1049/ic.2011.0012.

Gupta, A., Sharma, S., Goyal, S., & Rashid, M. (2020). Novel XGBoost tuned machine learning model for software bug prediction. In Proceedings of the 2020 International Conference on Intelligent Engineering and Management (ICIEM) (pp. 376–380). https://doi.org/10.1109/ICIEM48762.2020.9160152.

Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. International Journal of Advanced Computer Science and Applications, 9(2), 78–83.

Bala, Y. Z., Samat, P. A., Sharif, K. Y., & Manshor, N. (2022). Current software defect prediction: A systematic review. Proceedings of the 2022 Applied Informatics International Conference (AiIC), 117–121. https://doi.org/10.1109/AiIC54368.2022.9914586.

Iqbal, A., & Aftab, S. (2020). A classification framework for software defect prediction using multi-filter feature selection technique and MLP. International Journal of Modern Education and Computer Science, 12(1), 18–25. https://doi.org/10.5815/ijmecs.2020.01.03.

Iqbal, A., Aftab, S., Ali, U., Nawaz, Z., Sana, L., Ahmad, M., & Husen, A. (2019). Performance analysis of machine learning techniques on software defect prediction using NASA datasets. International Journal of Advanced Computer Science and Applications, 10(5), 1–6. https://doi.org/10.14569/IJACSA.2019.0100538. 12(1), 18–25. https://doi.org/10.5815/ijmecs.2020.01.03.

Khleel, N. A. A., & Nehéz, K. (2023). Overview of modern software bug prediction approaches. Doktoranduszok Fóruma, 55, 55–61.

Mahfoodh, H., & Obediat, Q. (2020). Software risk estimation through bug reports analysis and bug-fix time predictions. In Proceedings of the 2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT) (pp. 1–6). https://doi.org/10.1109/3ICT51146.2020.9312003.

Mcmurray, S., & Sodhro, A. H. (2023). A study on ML-based software defect detection for security traceability in smart healthcare applications. Sensors, 23(7), 3470. https://doi.org/10.3390/s23073470.

Meenakshi, D., & Singh, S. (2019). Software bug prediction using machine learning approach. International Journal of Advanced Computer Science and Applications, 10(5), 1–6.

Naidu, G. G., Balamurugan, M. S., & Savitha, M. S. (2022). A novel XGBoost tuned machine learning model for software bug prediction. International Journal for Research Trends and Innovation, 7(6), 764–768.

Puranik, S., Deshpande, P., & Chandrasekaran, K. (2016). A novel machine learning approach for bug prediction. Procedia Computer Science, 93, 924–930. https://doi.org/10.1016/j.procs.2016.07.271.

Rizwan, S., Tiantian, W., Xiaohong, S., & Salahuddin. (2017). Empirical study on software bug prediction. In Proceedings of the 2017 International Conference on Software and e-Business (pp. 55–59). https://doi.org/10.1145/3178212.3178221.

Sharma, T., Jatain, A., Bhaskar, S., & Pabreja, K. (2023). Ensemble machine learning paradigms in software defect prediction. Procedia Computer Science, 218, 199–209. https://doi.org/10.1016/j.procs.2023.01.002.

Shin, J., Aleithan, R., Nam, J., Wang, J., & Wang, S. (2021). Explainable software defect prediction: Are we there yet? arXiv. https://doi.org/10.48550/arXiv.2111.10901.

Siva, R., Hariharan, B., & Premkumar, N. (2023). Automatic software bug prediction using adaptive artificial jelly optimization with long short-term memory. Wireless Personal Communications, 132(3), 1975–1998. https://doi.org/10.1007/s11277-023-10694-9.

Siva, R., Kaliraj, S., Hariharan, B., & Premkumar, N. (2024). Automatic software bug prediction using adaptive golden eagle optimizer with deep learning. Multimedia Tools and Applications, 83(1), 1261–1281. https://doi.org/10.1007/s11042-023-16666-2.

Tong, H., Lu, W., Xing, W., Liu, B., & Wang, S. (2022). SHSE: A subspace hybrid sampling ensemble method for software defect number prediction. Information and Software Technology, 142, 106747.

Wang, S., Wang, J., Nam, J., & Nagappan, N. (2021). Continuous software bug prediction. In Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (pp. 1–12). https://doi.org/10.1145/3475716.3475790.