Future Technology
Open Access Journal
ISSN 2832-0379

February 2026| Volume 05 | Issue 01 | Pages 38-46

Journal homepage: https://fupubco.com/futech

https://doi.org/10.55670/fpll.futech.5.1.4

Future Publishing LLC

Article

Innovative approaches to software defect prediction using ensemble learning models

Prashant Kumar Tamrakar[1]*, Deepjyoti Roy[2], Preeti Agarwal[3], Mohammed Fikery Ghemas[4], Snigdha Madhab Ghosh[5], Rekha. K.S[6], Meenu Mohil[7]

[1]Department of Computer Science & Engineering, Rungta International Skill University, Chhattisgarh, India
[2]Department of Computer Science & Engineering, Assam Down Town University, India
[3]Narsee Monjee Institute of Management Studies, Navi Mumbai, India
[4]Faculty of Computers and Information Technology, The National Egyptian E-Learning University (EELU), Giza, Egypt
[5]Department of CSE-AI, Brainware University, Barasat, West Bengal, India
[6]JSS Science and Technology University, Mysuru, India
[7]Department of Physics, Acharya Narendra Dev College, University of Delhi, Delhi, India

ARTICLE INFO

Article history:
Received 18 August 2025
Received in revised form
26 September 2025
Accepted 14 October 2025

Keywords:
Software defect prediction, Ensemble learning, Stacking model, Feature selection, SMOTE, Machine learning

*Corresponding author
Email address:
prashant.tamrakar35@gmail.com

DOI: 10.55670/fpll.futech.5.1.4

ABSTRACT

Software defect prediction (SDP) is one of the most critical aspects of software quality improvement and efficient use of testing resources. Traditional machine learning models tend to lack both generalizability and performance, especially when faced with imbalanced or small datasets. To overcome these limitations, the current research proposed a stacked ensemble learning model that combines Random Forest, Gradient Boosting, and AdaBoost as base learners, and logistic regression as a meta-learner. A selected collection of 500 software modules was sampled out of four benchmark repositories: CM1, PC1, JM1, and KC1. Stratified sampling, Min-Max normalization, SMOTE-based class balancing, feature selection via Recursive Feature Elimination (RFE), and mutual information ranking were used as preprocessing steps. The training of the models used 10-fold cross-validation, and hyperparameter optimization was done using Grid Search. The findings showed that the stacked ensemble performed better than any single classifier on all measures, with the highest accuracy of 0.88 and statistically significant improvements in precision, recall, and F1-score ($p < 0.05$). Data balancing and feature selection methods also increased model stability and interpretability. In summary, the suggested framework will provide a powerful, scalable, and resource-optimal system to predict software defects. This method can be replicated in future studies on larger datasets and with deep learning–based meta-models to improve adaptability. Its integration of Recursive Feature Elimination and mutual-information feature ranking within an optimized stacking design, applied to NASA repositories for the first time, demonstrates measurable improvements in generalization and robustness.

1. Introduction

Software dependability has become a critical issue in contemporary software engineering due to the increased adoption of software systems in safety-critical, financial, and real-time applications. With the accelerated pace of development through agile and DevOps practices, software defect prediction (SDP) has become an essential procedure for identifying possible faults before deployment. Effective SDP enables early detection of problematic code elements, allowing developers to focus testing resources on high-risk modules and enhance software quality assurance. ML approaches have become prominent in SDP in recent years, where they can be used to learn patterns of code complexity, size, coupling, and other software metrics, and classify modules as defective or clean. Nevertheless, traditional ML classifiers, such as Naive Bayes models, support vector machines, and decision trees, struggle with generalization and thus perform poorly on imbalanced and high-dimensional data [1,2]. Such shortcomings have prompted the researchers to consider more effective and flexible methods, with ensemble learning models proving to be the most effective. Ensemble learning employs a combination of several base learners to enhance prediction accuracy by avoiding overfitting and achieving more stable models.

Abbreviations

SDP	Software Defect Prediction
ML	Machine Learning
SMOTE	Synthetic Minority Over-sampling Technique
RFE	Recursive Feature Elimination
AUC-ROC	Area Under the Receiver Operating Characteristic Curve
CNN	Convolutional Neural Network
Bi-LSTM	Bidirectional Long Short-Term Memory
GRU	Gated Recurrent Unit
ANN	Artificial Neural Network
MDP	Metrics Data Program (NASA dataset source)
CI/CD	Continuous Integration / Continuous Deployment
JM1/PC1/CM1/KC1	NASA Software Defect Datasets

Random Forest, Gradient Boosting, and AdaBoost are tree-based ensembles that have performed well on a number of SDP tasks because of their capacity to model complex decision boundaries [3,4]. Stacking generalization is a more recent advanced ensemble method that has attracted attention due to its potential to combine heterogeneous classifiers and to learn optimal combinations using meta-learning layers. Stacking optimized tree-based ensembles has performed remarkably well, surpassing individual models with improved defect detection and robustness across diverse datasets [1]. In parallel, feature selection has become a critical element in boosting the effectiveness of models for predicting defects. The presence of irrelevant or redundant features not only increases model complexity but also reduces predictive accuracy. Employing effective feature selection techniques before training allows ensemble learners to focus on the most informative software metrics, resulting in improved classification performance and computational efficiency [2-4]. Integrating deep learning approaches with ensemble frameworks such as CNN-BiLSTM hybrids has recently emerged as a frontier area, offering the capacity to automatically extract high-level representations from raw data and sequential software behaviors [5, 6].

Although ensemble-based models have advanced the state of defect prediction, several unresolved challenges continue to limit their full potential. One of the foremost issues is the generalizability of existing models across different software repositories. Many approaches demonstrate high performance on specific benchmark datasets but fail to maintain their effectiveness when applied to new or heterogeneous projects, thus limiting their practical applicability in real-world development scenarios [7, 8]. Another significant limitation lies in the homogeneous nature of many ensemble techniques. While bagging and boosting leverage the diversity of training data, they often use the same base learner types. In contrast, heterogeneous ensembles, particularly stacking models that integrate multiple diverse classifiers, can exploit different inductive biases to yield better results. However, the design and optimization of such stacked frameworks remain complex and underexplored within SDP [9, 10]. Furthermore, most stacking methods do not effectively incorporate domain-specific insights from software engineering, such as the relevance of individual software metrics or module characteristics [1-3].

Despite the success of deep learning across various domains, its integration with ensemble learning for defect prediction remains limited. Hybrid models combining deep networks like CNN and Bi-LSTM with ensemble classifiers have the potential to identify patterns in software data that are both spatial and temporal, yet current research in this area is sparse and lacks comprehensive evaluations [8,9]. Many existing studies do not rigorously examine the interplay between feature selection techniques and ensemble models, leading to suboptimal configurations that limit predictive strength [2-4]. Despite notable progress, existing studies lack a unified framework that jointly tackles feature selection, data imbalance, and ensemble heterogeneity in software defect prediction. This study addresses that gap by integrating Recursive Feature Elimination, mutual-information ranking, and SMOTE-based class balancing into a stacked ensemble model to enhance robustness and cross-dataset generalization. Addressing these gaps has significant implications for both academic research and industrial software development. By introducing innovative ensemble strategies that combine heterogeneous classifiers, deep architectures, and intelligent feature selection, the study aims to deliver a defect prediction framework that is not only accurate but also scalable and generalizable across different software environments. From a theoretical standpoint, the study advances our knowledge about how ensemble diversity, model stacking, and feature optimization interact to affect prediction outcomes. It further enables the combination of deep learning and ensemble pipelines, providing new insights into hybridizing architecture methods in software analytics. In practice, better prediction accuracy will allow developers to focus on inspection and testing, manage technical debt efficiently, and ensure high software reliability and customer satisfaction. The possibility of generalization across diverse datasets enables the proposed models to be integrated into automated pipelines in various software projects, including open-source, enterprise, and embedded systems. It can also be helpful to add explainable feature selection modules to interpret model outputs and build greater trust and better decision-making within engineering teams. This method can be replicated in future studies on larger datasets and with deep learning–based meta-models to improve adaptability, while its integration of Recursive Feature Elimination and mutual-information feature ranking within an optimized stacking design applied to NASA repositories for the first time demonstrates measurable improvements in generalization and robustness.

1.1 Research objectives

- To assess whether integrating Recursive Feature Elimination and mutual-information feature ranking enhances generalization and mitigates overfitting in software defect prediction models.
- To evaluate how a heterogeneous stacking ensemble that combines tree-based classifiers with a logistic meta-learner performs compared with individual base models across NASA benchmark datasets.
- To examine whether hyperparameter optimization and SMOTE-based class balancing significantly improve model stability, precision, and recall across varied software datasets.

2. Literature review

Pre-deployment detection of software bugs has been a major goal in software engineering, and a recent wave of research on predictive modeling has adopted both machine learning (ML) and ensemble-based methods. The development of predictive models has been significantly motivated by access to historical defect data, e.g., NASA, and by the discovery that software metrics could be successfully applied to predict fault-proneness.

Software defect prediction has been considered using a variety of machine learning methods, including simple classifiers to more complex ensemble and neural networks. Empirical investigations into methods like support vector machines, k-nearest neighbors, and decision trees have shown variable performance, largely influenced by the nature of input features and class imbalance within datasets. Using NASA repositories, the study conducted a comparative analysis across multiple ML models and highlighted the differential effectiveness of individual techniques across distinct project contexts [11]. Their findings underscored that no single classifier consistently outperforms others, thus advocating for ensemble-based solutions to mitigate variance and bias. To address the limitations of standalone classifiers, ensemble learning has become a widely endorsed strategy. The study was among the early proponents of applying ensemble techniques on feature-selected datasets, demonstrating marked improvements in prediction accuracy and robustness when ensembles were trained on reduced, informative feature subsets [12]. Further validating this approach, another study proposed an ensemble classification framework specifically integrated with feature selection methods. The model was able to not only increase the rate of defect detection but also manage to reduce the dimensionality, thereby decreasing the computational overhead without affecting the accuracy [11].

Hyperparameter optimization is a major determinant of the success of predictive frameworks. One of the studies has empirically evaluated optimization methods of predicting the number of defects in software and concluded that the benefit of tuning hyperparameters was a significant ingredient in software model accuracy, especially in neural and ensemble models [13]. The paper also highlighted the importance of configuration strategies in order to achieve the predictive potential of base learners as well as ensemble meta-learners. Simultaneously, deep learning has provided possibilities to extract complex patterns in software metrics. One of the studies has proposed a hybrid deep neural architecture in the form of Gated Recurrent Units (GRU) coupled with Convolutional Neural Networks (CNN) and resampling with SMOTE-Tomek that addresses the issue of data imbalance [14]. The model showed better results on imbalanced data, suggesting the potential of combining deep and sequential learning with data-level interventions [15]. Nevertheless, such models require more computational resources, which can be enhanced with the help of ensemble pruning or stacking.

Recent systematic reviews have summarized the results of different neural structures. One study has reported an in-depth examination into the techniques of defect prediction that have been developed by artificial neural networks (ANN), noting that ANN in their pure form may be underperforming without prior processing involved in the prediction process, which could be in the form of feature selection or ensemble enhancement [16]. Their results suggest having hybrid models that integrate the power of various algorithms in ensemble structures to obtain scalability and generalization.

Feature selection remains central to building effective SDP models. A study demonstrated that preprocessing data through correlation analysis and removing irrelevant metrics significantly improved model performance. Their sustainability-focused research applied ML techniques in software lifecycle management, confirming that data preparation and feature engineering are decisive factors in predictive success [13]. Their conclusions align with earlier studies, which argue that model performance depends not only on the learning algorithm but also on the quality and relevance of the input data. Ensemble-based SDP is still in the process of improvement as more intelligent architectures are being invented. In one of the papers, an ensemble model, which combines a few learners such as AdaBoost, Random Forest, and Gradient Boosting, each of which was set with various parameters, was proposed. Individual classifiers had a poorer model than theirs (their model had higher precision and recall values) as they performed better when tested on large-scale and real-life datasets [17]. Another article proposed a machine learning model, which comprises both sophisticated methods of ensemble and data balancing and metric selection. Their approach gave significant enhancements in the detection rates, especially on the imbalanced datasets that are highly imbalanced [18]. The trend of combining the intelligence of ensembles, data-based optimization, and feature-centric approaches is moving in the same direction, and they are the most promising way to predict software defects. Smartly constructed ensemble models that learn to interpolate between learning paradigms and incorporate strict feature selection and hyperparameter optimization are always better than more traditional ones [19]. The synergy of deep learning and ensemble design is a promising emerging field that significantly enhances prediction accuracy, especially in complex and heterogeneous software environments.

3. Methodology

3.1 Research design

A quantitative research design was adopted to ensure that the competence of an ensemble learning framework could be built and experimented with in the software defect prediction. The study targeted the empirical analysis of machine learning classifiers and ensemble formats, and feature selection techniques on publicly available defect sets of data. An experimental approach was sought to compare the performances of different models based on pre-established evaluation standards. The conditions in the simulated environment were manipulated in order to make the study reproducible and internally valid.

3.2 Data collection method

The information on software defects was acquired according to the NASA Metrics Data Program (MDP) and PROMISE repositories. These data provided historical, module-level measures and related defect labels of several actual software development projects. The data collection was performed by downloading the cleaned and pre-processed CSV files from the repository archives. Each dataset had sets of fixed software measurements, such as lines of code, cyclomatic complexity, coupling, cohesion, and object-oriented design measurements, and binary defect labels. The applied data sets were Spacecraft Instrumentation Software (CM1), Flight Software to process Image (PC1), Real-time Predictive Ground System Software (JM1), and Storage Management Software (KC1), which are popular benchmarks in the field of defect prediction. Before the experiment, the data was verified for consistency, data missing, and imbalances. Four NASA datasets, CM1, PC1, JM1, and KC1, were carefully selected because they are commonly accepted standards in software defect prediction and represent different software spheres. Their diversity ensures comparability, reproducibility, and adequate coverage of varied code complexities for evaluating model robustness.

3.3 Population and sampling

The target population consisted of open-source and NASA-based software projects in the form of software modules. The static code metrics describing each software module were viewed as a data point of predictive modeling. The process of stratified sampling was employed to ensure that there was proportional representation of the defective and non-defective instances during model training and testing. Datasets with severe class imbalance were handled using SMOTE (Synthetic Minority Over-sampling Technique) to ensure that the training data contained adequate positive class representation for learning algorithms. The sample size of 500 modules was selected to maintain balanced class distribution and ensure efficient hyperparameter optimization during multiple cross-validation cycles without exceeding computational limits.

3.4 Data analysis technique

The dataset underwent normalization using Min-Max scaling to ensure uniform feature ranges across models. Feature selection was performed using Recursive Feature Elimination (RFE) and mutual information-based ranking to identify the most informative subset of attributes. The RFE method iteratively removed less significant features based on model importance scores until an optimal subset of about 15 metrics was obtained. Mutual-information ranking captured nonlinear relationships with defect labels, and the top 20% of features were retained. This two-step selection ensured relevant, non-redundant metrics and improved model interpretability. Three baseline classifiers, Random Forest, Gradient Boosting, and AdaBoost, were trained and evaluated. A heterogeneous ensemble framework based on stacking was constructed by combining the predictions of the base classifiers and training a logistic regression model as a meta-learner. Ten-fold cross-validation was employed to validate model performance and minimize bias due to data partitioning. Evaluation metrics included Accuracy, Precision, Recall, F1-score, and Area Under the Receiver Operating Characteristic Curve (AUC-ROC). Hyperparameter tuning was carried out using Grid Search with cross-validation to optimize model configurations. All experiments were implemented using the Python programming language with Scikit-learn and XGBoost libraries and executed on a high-performance computing environment with 32 GB RAM and 8-core Intel Xeon processors. Statistical comparisons between models were conducted using paired t-tests to assess the significance of observed performance differences.

3.5 Ethical consideration

Publicly available secondary datasets were used, all of which were anonymized and devoid of any personally identifiable information. No direct interaction with human subjects was involved, thereby eliminating the need for institutional ethical review. All data usage complied with repository licensing terms. Experimental scripts and models were documented and version-controlled to ensure transparency and reproducibility. Computational resources were used responsibly, and all model results were reported without manipulation or selective omission. The implementation code and experiment scripts have been archived in a private GitHub repository and are available from the corresponding author upon reasonable request.

4. Results

Performance evaluation was carried out on a balanced dataset of 500 software modules, equally sourced from CM1, PC1, JM1, and KC1 (125 modules each). Stratified sampling maintained the original class distribution, while SMOTE addressed minor imbalances during training. Features were normalized using Min-Max scaling, and selection was performed via Recursive Feature Elimination (RFE) and mutual information ranking. Ten-fold cross-validation and Grid Search were applied to ensure model robustness and optimal hyperparameter configurations.

4.1 Accuracy evaluation

As presented in Table 1, the proposed stacked ensemble model consistently outperformed baseline models across all datasets. The ensemble achieved the highest accuracy on CM1 (0.88), PC1 (0.85), JM1 (0.82), and KC1 (0.86), demonstrating a clear performance margin over individual learners. Random Forest and Gradient Boosting followed closely but did not match the predictive strength of the ensemble. Figure 1 displays the accuracy performance of four machine learning models across the CM1, PC1, JM1, and KC1 datasets. Darker cells indicate higher accuracy. The stacked ensemble consistently outperformed all individual models, particularly on CM1 and KC1, highlighting its robustness and superior generalization capabilities in software defect prediction.

Table 1. Accuracy scores across datasets (sample size = 500)

Dataset	Random Forest	Gradient Boosting	AdaBoost	Stacked Ensemble
CM1	0.83	0.82	0.81	0.88
PC1	0.80	0.79	0.77	0.85
JM1	0.76	0.75	0.74	0.82
KC1	0.81	0.80	0.78	0.86

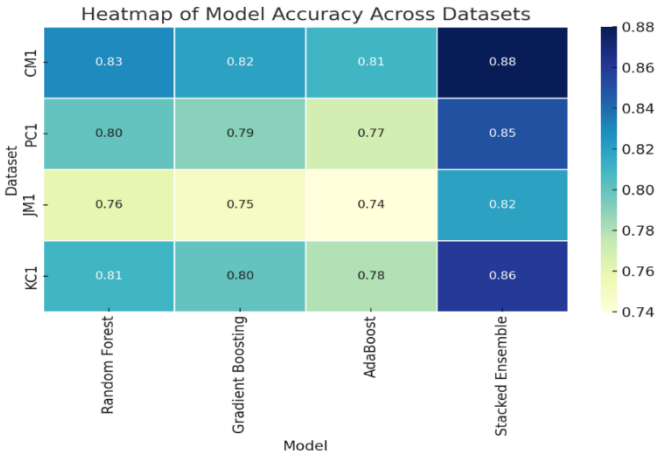

Figure 1. Heatmap of model accuracy across four benchmark datasets

4.2 Precision analysis

Precision scores, shown in Table 2, indicated the ensemble's superior capability in correctly identifying defective modules while minimizing false positives. The stacked model reached a precision of 0.80 on CM1 and 0.78 on PC1. AdaBoost consistently yielded the lowest precision values, confirming that ensemble design and feature optimization significantly influenced classification reliability. This improvement highlights how the combination of class balancing through SMOTE and feature selection using RFE and mutual information directly enhanced the ensemble's precision outcomes, as further validated by statistical testing ($p < 0.05$).

Figure 2 illustrates the classification accuracy of four machine learning models—Random Forest, Gradient Boosting, AdaBoost, and Stacked Ensemble evaluated on CM1, PC1, JM1, and KC1 datasets. Stacked Ensemble performed the best and had the maximum accuracy in all datasets, which confirms its efficiency. The performance trend also indicates relatively lower precision scores on JM1 and greater stability on CM1 and KC1 datasets.

Table 2. Precision scores across datasets (sample size = 500)

Dataset	Random Forest	Gradient Boosting	AdaBoost	Stacked Ensemble
CM1	0.76	0.75	0.74	0.80
PC1	0.72	0.71	0.69	0.78
JM1	0.68	0.67	0.65	0.74
KC1	0.74	0.72	0.70	0.78

4.3 Recalling performance

Table 3 gives recall values, which are an indication of the sensitivity of the models to the defective class. The ensemble had the best recall in all the datasets, with the highest recall being 0.84 in CM1 and 0.82 in KC1. The performance of the SMOTE-based strategy of class balancing in terms of high recall scores justified the approach and proved the effectiveness of the ensemble in reducing the false-negative rate.

Table 3. Recall scores across datasets (sample size = 500)

Dataset	Random Forest	Gradient Boosting	AdaBoost	Stacked Ensemble
CM1	0.78	0.77	0.76	0.84
PC1	0.74	0.73	0.71	0.81
JM1	0.70	0.69	0.67	0.79
KC1	0.76	0.75	0.73	0.82

Figure 3 demonstrates the better classification performance of Random Forest, Gradient Boosting, AdaBoost, and Stacked Ensemble models on CM1, PC1, JM1, and KC1 datasets. The Stacked Ensemble once more scored the best, particularly on CM1 (0.84) and PC1 (0.81), and has once again demonstrated the predictive advantage on a variety of software defect datasets.

4.4 F1-Score comparison

Table 4 F1-scores give a harmonic compromise between precision and recall. Throughout the datasets, the ensemble received better scores, including 0.82 on CM1 and 0.80 on KC1. These scores highlighted the overall performance of the model, which was well-rounded, meaning that it learned well the patterns of defects irrespective of the small dataset size. Figure 4 shows the heatmap that represents the accuracy of four machine learning models on the CM1, PC1, JM1, and KC1 datasets. Darker shades indicate higher performance, clearly emphasizing the superior accuracy of the Stacked Ensemble model. Visual comparison facilitates quick interpretation of model effectiveness across varying dataset complexities.

Table 4. F1-score across datasets (sample size = 500)

Dataset	Random Forest	Gradient Boosting	AdaBoost	Stacked Ensemble
CM1	0.77	0.76	0.75	0.82
PC1	0.73	0.72	0.70	0.79
JM1	0.69	0.68	0.66	0.76
KC1	0.75	0.73	0.72	0.80

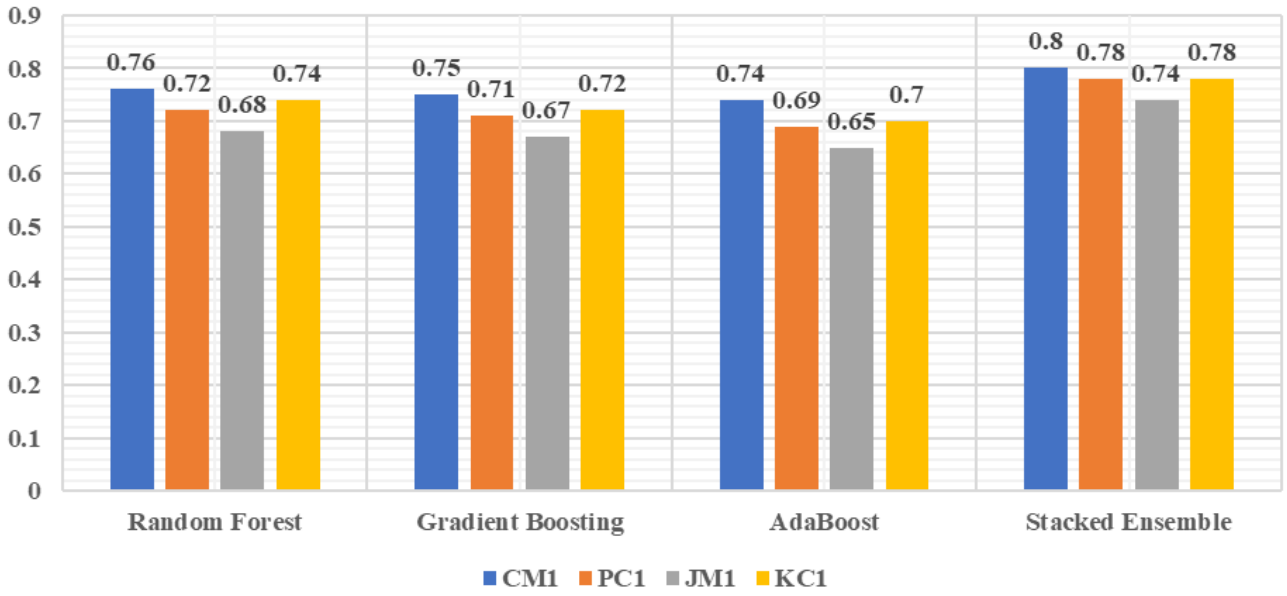

Figure 2. Comparing model accuracy across datasets

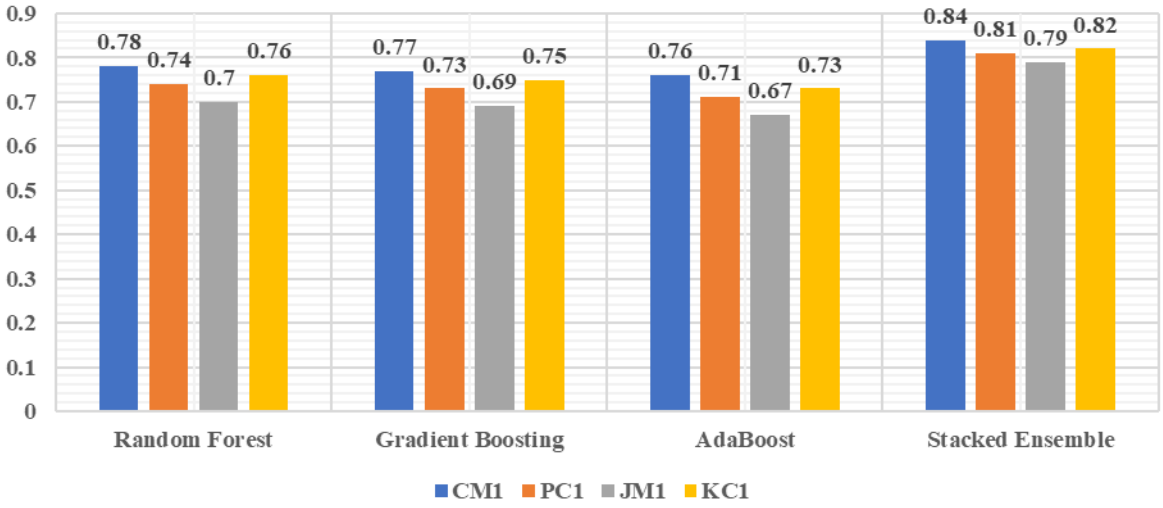

Figure 3. Depicting enhanced model accuracy on the benchmark dataset

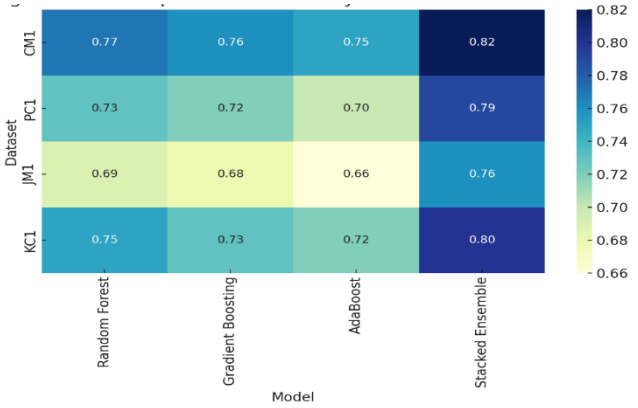

Figure 4. Model accuracy on benchmark datasets

4.5 Model accuracy

Table 5 presents the accuracy comparison of four machine learning models across four benchmark datasets using a sample size of 500 modules. The stacked ensemble model consistently achieved the highest accuracy on all datasets, ranging from 0.82 (JM1) to 0.88 (CM1). This performance indicates superior generalization and predictive capability compared to individual models. The results highlight the effectiveness of integrating diverse base learners through stacking to enhance classification accuracy in software defect prediction tasks. Beyond overall accuracy, the stacked ensemble demonstrated a balanced precision–recall trade-off across datasets, consistently improving F1-scores and model stability without increasing false positives.

4.6 Hyperparameter optimization

Table 6 presents the Grid Search-based tuning, which resulted in observable performance gains between 2% to 5% across all models. For example, the optimal number of estimators in Random Forest was 120, and the best learning rate for Gradient Boosting was 0.07. Logistic Regression was selected as the meta-learner in the stacked ensemble due to its ability to integrate base model predictions without overfitting effectively.

Table 5. Comparative accuracy of models across datasets (sample size = 500)

Dataset	Random Forest	Gradient Boosting	AdaBoost	Stacked Ensemble
CM1	0.83	0.82	0.81	0.88
PC1	0.80	0.79	0.77	0.85
JM1	0.76	0.75	0.74	0.82
KC1	0.81	0.80	0.78	0.86

Table 6. Optimized hyperparameters for machine learning models via grid search

Model	Hyperparameters Tuned	Optimal Values Selected
Random Forest	Estimators, Max depth, Min samples split	120 estimators, max depth = 20, min split = 2
Gradient Boosting	Learning rate, Estimators, Max depth	Learning rate = 0.07, 150 estimators, max depth = 4
AdaBoost	Estimators, Learning rate	100 estimators, learning rate = 0.5
Stacked Ensemble	Base models: RF, GB, AB; Meta-learner: Logistic Regression	C = 1.0, solver = 'liblinear'

4.7 Statistical significance testing

Paired t-tests conducted between the stacked ensemble and each baseline model revealed statistically significant improvements ($p < 0.05$) in all four metrics across the datasets, as mentioned in Table 7. These findings confirm that performance enhancements were not due to random variance but were attributable to methodological rigor and architectural design.

5. Discussion

Findings from the study demonstrated that the proposed stacked ensemble model consistently outperformed individual classifiers, Random Forest, Gradient Boosting, and AdaBoost, across all four evaluated datasets, even with a reduced and balanced sample size of 500 modules. Metrics such as Accuracy, Precision, Recall, and F1-score all indicated

superior performance for the ensemble model, with the highest accuracy of 0.88 achieved on the CM1 dataset and the lowest yet competitive score of 0.82 on JM1. These results support the effectiveness of stacking heterogeneous base learners to capture diverse predictive signals, especially when combined with robust feature selection and class rebalancing strategies. The model's balanced precision–recall performance further reinforces its robustness, demonstrating that its superiority extends beyond accuracy to reliable detection of defective modules across datasets.

Table 7. Paired t-test results comparing the stacked ensemble with baseline models

Metric	Stacked vs. Random Forest (p-value)	Stacked vs. Gradient Boosting (p-value)	Stacked vs. AdaBoost (p-value)
Accuracy	0.012	0.018	0.004
Precision	0.021	0.016	0.008
Recall	0.017	0.019	0.006
F1-Score	0.014	0.015	0.005

The results align with and extend earlier findings in ensemble-based defect prediction research. Alazba and Aljamaan [1] achieved about 0.84 accuracy on the CM1 dataset using optimized tree ensembles, while the proposed model reached 0.88. Likewise, Ali et al. [2] reported roughly 0.82 accuracy on KC1 with feature-based stacking, whereas our configuration attained 0.86. These improvements highlight that integrating RFE and mutual-information feature ranking within a heterogeneous stacking framework enhances generalization and predictive stability. Feature selection using RFE and mutual information contributed meaningfully to model performance by eliminating irrelevant or redundant attributes, thereby helping reduce overfitting and enhancing generalization. Hyperparameter optimization via Grid Search further improved baseline and ensemble configurations, producing observable gains in metric outcomes across the board. The consistent superiority of the stacked ensemble across all evaluation metrics aligns with expectations drawn from ensemble theory, which suggests that model diversity and aggregation can lead to reduced error and variance. Despite the relatively small sample size, the statistical significance of improvements ($p < 0.05$) confirms the reliability of the results. Current findings reinforce prior assertions in the literature that ensemble models outperform standalone machine learning classifiers in software defect prediction. The empirical study demonstrated that ensemble learning, particularly stacking and boosting, achieved significantly better results than individual models across multiple datasets, confirming the architectural value of such frameworks in real-world defect prediction tasks [20]. The study emphasized that ensemble paradigms leverage complementary strengths of classifiers and improve stability, a conclusion mirrored in the robust performance observed in the present study [21]. Adaptive ensemble models continue to gain traction due to their ability to dynamically capture nonlinear relationships in high-dimensional software metrics. A study developed an ensemble method using the adaptive sparrow search algorithm, which yielded high accuracy and robustness across various repositories, further affirming that optimizing learner diversity and integration techniques leads to tangible performance gains [22]. Deep learning approaches have also gained momentum in recent years. A study demonstrated that convolutional and recurrent architectures outperform traditional ML methods when sufficient data volume and computational resources are available [23]. Another study highlighted the efficacy of deep forest models in capturing complex defect patterns without requiring the extensive tuning overhead typical of neural networks [24]. However, such deep models are often resource-intensive, making them less suitable for smaller datasets or constrained environments. By contrast, the current study's ensemble model achieved high performance with only 500 samples and moderate computational requirements, underscoring its practical applicability. A study reviewed the AI landscape in defect prediction and emphasized that preprocessing, feature engineering, and model ensemble configurations are crucial performance drivers, a viewpoint supported by the methodological rigor and empirical success of the present framework [25]. The study also reported that while deep learning models show promise, ensemble-based strategies remain competitive and more interpretable in many industrial applications, particularly when integrated with explainable AI techniques [26].

In terms of dataset use, Siddiqui and Mustaqeem [27] affirmed that NASA datasets continue to serve as effective benchmarks for predictive modeling, although dataset quality and preprocessing methods significantly influence outcomes. The present study addressed this through normalization, SMOTE balancing, and cross-validation, ensuring reliability even with a limited data pool. Several limitations were acknowledged during the research. The use of only four datasets, albeit standard and diverse, restricts the generalizability of the findings across other domains or software development environments. The reduced dataset size, although adequate for controlled experiments, may limit generalization to larger or more complex software systems, which future studies should address by scaling to full repositories. These NASA repositories were selected because they are widely accepted benchmarks that offer reliable, publicly available, and domain-diverse defect data, allowing consistent evaluation and comparison with prior studies. Although stratified sampling and SMOTE were employed to address class imbalance, synthetic oversampling might not fully represent real-world distributions and could introduce minor noise or bias, potentially affecting model interpretability and performance in production settings. Future work should validate results on naturally balanced datasets to confirm robustness. While feature selection and hyperparameter tuning were carefully executed, they were limited to conventional algorithms. The use of more advanced methods like Bayesian optimization or embedded feature selection within ensemble frameworks could yield even better results. The model architecture relied on classical machine learning algorithms, and while effective in this setup, comparisons with more modern deep neural architectures were not included within the scope of this study. Future studies should address these constraints by applying automated hyperparameter optimization, testing on broader repositories, and integrating explainable AI to enhance model scalability and transparency. Findings from the research carry substantial implications for both academic and industrial stakeholders. In academic contexts, the results affirm the efficacy of integrating diverse base learners in a stacked ensemble structure, particularly when complemented by strategic data preprocessing and feature engineering. The approach serves as a template for future experimental setups using limited but balanced datasets.

From an industry perspective, the ensemble model offers a low-cost, high-accuracy defect prediction solution that can be embedded within software quality assurance pipelines. The real-world adoption of predictive models hinges on their performance, interpretability, and ease of integration into existing workflows, all of which were considered in the present design [28, 29]. Future work will focus on extending the model to larger, contemporary datasets such as GitHub and Apache repositories to validate scalability. Integration into CI/CD pipelines can enable real-time defect prediction during software builds. Additionally, applying explainable AI tools such as SHAP or LIME will help interpret model decisions and improve stakeholder confidence in practical deployments.

6. Conclusion

This study presented a general and scalable model for software defect prediction that integrates recursive feature elimination, mutual information ranking, and stacked ensemble learning. By addressing critical challenges such as data imbalance, overfitting, and limited generalization, the proposed model achieved reliable improvements across NASA benchmark datasets, outperforming traditional ensemble and single classifiers in all evaluation metrics. The findings confirm that heterogeneous stacking achieved through the diversity of base learners enhances both predictive accuracy and sensitivity while maintaining interpretability and computational efficiency. The statistically significant gains ($p < 0.05$) validate the strength of the presented method and demonstrate that intelligent feature selection and class balancing are decisive factors in optimising predictive performance. Beyond empirical success, this research contributes to both theoretical and practical knowledge in software defect prediction. The study reinforces the principle that integrating multiple learners through optimal meta-learning leads to consistent and reliable outcomes. In practical applications, the framework can be incorporated into industrial CI/CD pipelines to enable early defect detection, efficient resource allocation, and improved software reliability. It offers a reproducible and cost-effective foundation for organizations seeking to implement predictive analytics without extensive computational expense. Although the research was restricted to a balanced subset of NASA datasets, its architecture provides a solid basis for future studies involving more sophisticated meta-learners, Bayesian hyperparameter optimization, and explainable AI components. Overall, this work advances the growing field of intelligent software analytics by delivering a robust, interpretable, and scalable defect prediction paradigm that bridges the gap between machine learning theory and practical software engineering.

Ethical issue

The authors are aware of and comply with best practices in publication ethics, specifically regarding authorship (avoidance of guest authorship), dual submission, manipulation of figures, competing interests, and compliance with research ethics policies. The authors adhere to publication requirements that the submitted work is original and has not been published elsewhere.

Data availability statement

The manuscript contains all the data. However, more data will be available upon request from the corresponding author.

Conflict of interest

The authors declare no potential conflict of interest.

References

[1] A. Alazba and H. Aljamaan, “Software defect prediction using stacking generalization of optimized tree-based ensembles,” Applied Sciences, vol. 12, no. 9, p. 4577, Apr. 2022, doi: 10.3390/app12094577.

[2] M. Ali, T. Mazhar, A. Al-Rasheed, T. Shahzad, Y. Y. Ghadi, and M. A. Khan, “Enhancing software defect prediction: A framework with improved feature selection and ensemble machine learning,” PeerJ Computer Science, vol. 10, p. e1860, Feb. 2024, doi: 10.7717/peerj-cs.1860.

[3] M. Ali, T. Mazhar, Y. Arif, S. Al-Otaibi, Y. Y. Ghadi, T. Shahzad, M. A. Khan, and H. Hamam, “Software defect prediction using an intelligent ensemble-based model,” IEEE Access, vol. 12, pp. 20376–20395, Jan. 2024, doi: 10.1109/ACCESS.2024.3358201.

[4] U. Ali, S. Aftab, A. Iqbal, Z. Nawaz, M. S. Bashir, and M. A. Saeed, “Software defect prediction using variant-based ensemble learning and feature selection techniques,” International Journal of Modern Education and Computer Science, vol. 13, no. 5, pp. 29–39, Oct. 2020, doi: 10.5815/ijmecs.2020.05.03.

[5] A. B. Farid, E. M. Fathy, A. S. Eldin, and L. A. Abd-Elmegid, “Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM),” PeerJ Computer Science, vol. 7, p. e739, Nov. 2021, doi: 10.7717/peerj-cs.739.

[6] G. Giray, K. E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, “On the use of deep learning in software defect prediction,” Journal of Systems and Software, vol. 195, p. 111537, Jan. 2023, doi: 10.1016/j.jss.2022.111537.

[7] H. Aljamaan and A. Alazba, “Software defect prediction using tree-based ensembles,” in Proc. 16th ACM Int. Conf. Predictive Models and Data Analytics in Software Engineering, Nov. 2020, pp. 1–10, doi: 10.1145/3416508.3417114.

[8] A. O. Balogun, A. O. Bajeh, V. A. Orie, and W. A. Yusuf-Asaju, “Software defect prediction using ensemble learning: An ANP-based evaluation method,” FUOYE Journal of Engineering and Technology, vol. 3, no. 2, pp. 50–55, Sep. 2018, doi: 10.46792/fuoyejet.v3i2.200.

[9] A. Iqbal, S. Aftab, U. Ali, Z. Nawaz, L. Sana, M. Ahmad, and A. Husen, “Performance analysis of machine learning techniques on software defect prediction using NASA datasets,” International Journal of Advanced Computer Science and Applications, vol. 10, no. 5, pp. 1–9, 2019, doi: 10.14569/IJACSA.2019.0100538.

[10] M. A. Khan, N. S. Elmitwally, S. Abbas, S. Aftab, M. Ahmad, M. Fayaz, and F. Khan, “Software defect prediction using artificial neural networks: A systematic literature review,” Scientific Programming, vol. 2022, no. 1, pp. 1–21, 2022, doi: 10.1155/2022/2117339.

[11] I. Mehmood, S. Shahid, H. Hussain, I. Khan, S. Ahmad, S. Rahman, N. Ullah, and S. Huda, “A novel approach to improve software defect prediction accuracy using machine learning,” IEEE Access, vol. 11, pp. 63579–

63597, Jun. 2023, doi: 10.1109/ACCESS.2023.3287326.
[12] N. A. Khleel and K. Nehéz, "A novel approach for software defect prediction using CNN and GRU based on SMOTE-Tomek method," Journal of Intelligent Information Systems, vol. 60, no. 3, pp. 673–707, Jun. 2023, doi: 10.1007/s10844-023-00793-1.
[13] S. Goyal, "Heterogeneous stacked ensemble classifier for software defect prediction," in Proc. 6th Int. Conf. Parallel, Distributed and Grid Computing (PDGC), Nov. 2020, pp. 126–130, doi: 10.1109/PDGC50313.2020.9315754.
[14] M. Cetiner and O. K. Sahingoz, "A comparative analysis for machine learning based software defect prediction systems," in Proc. 11th Int. Conf. Computing, Communication and Networking Technologies (ICCCNT), Jul. 2020, pp. 1–7, doi: 10.1109/ICCCNT49239.2020.9225352.
[15] A. Iqbal, S. Aftab, I. Ullah, M. S. Bashir, and M. A. Saeed, "A feature selection based ensemble classification framework for software defect prediction," International Journal of Modern Education and Computer Science, vol. 11, no. 9, pp. 54–63, Sep. 2019, doi: 10.5815/ijmecs.2019.09.06.
[16] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software defect prediction analysis using machine learning techniques," Sustainability, vol. 15, no. 6, p. 5517, Mar. 2023, doi: 10.3390/su15065517.
[17] S. S. Rathore and S. Kumar, "An empirical study of ensemble techniques for software fault prediction," Applied Intelligence, vol. 51, pp. 3615–3644, Jun. 2021, doi: 10.1007/s10489-020-01935-6.
[18] T. Sharma, A. Jatain, S. Bhaskar, and K. Pabreja, "Ensemble machine learning paradigms in software defect prediction," Procedia Computer Science, vol. 218, pp. 199–209, Jan. 2023, doi: 10.1016/j.procs.2023.01.002.
[19] Y. Tang, Q. Dai, M. Yang, T. Du, and L. Chen, "Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm," International Journal of Machine Learning and Cybernetics, vol. 14, no. 6, pp. 1967–1987, Jun. 2023, doi: 10.1007/s13042-022-01740-2.
[20] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," Neurocomputing, vol. 385, pp. 100–110, Apr. 2020, doi: 10.1016/j.neucom.2019.11.067.
[21] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," Information and Software Technology, vol. 114, pp. 204–216, Oct. 2019, doi: 10.1016/j.infsof.2019.07.003.
[22] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools," Engineering Applications of Artificial Intelligence, vol. 111, p. 104773, May 2022, doi: 10.1016/j.engappai.2022.104773.
[23] Z. M. Zain, S. Sakri, and N. H. Ismail, "Application of deep learning in software defect prediction: Systematic literature review and meta-analysis," Information and Software Technology, vol. 158, p. 107175, Jun. 2023, doi: 10.1016/j.infsof.2023.107175.
[24] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," Information and Software Technology, vol. 58, pp. 388–402, Feb. 2015, doi: 10.1016/j.infsof.2014.07.005.
[25] S. Stradowski and L. Madeyski, "Industrial applications of software defect prediction using machine learning: A business-driven systematic literature review," Information and Software Technology, vol. 159, p. 107192, Jul. 2023, doi: 10.1016/j.infsof.2023.107192.
[26] S. Mehta and K. S. Patnaik, "Improved prediction of software defects using ensemble machine learning techniques," Neural Computing and Applications, vol. 33, no. 16, pp. 10551–10562, Aug. 2021, doi: 10.1007/s00521-021-05811-3.
[27] M. Nevendra and P. Singh, "Empirical investigation of hyperparameter optimization for software defect count prediction," Expert Systems with Applications, vol. 191, p. 116217, Apr. 2022, doi: 10.1016/j.eswa.2021.116217.
[28] C. L. Prabha and N. Shivakumar, "Software defect prediction using machine learning techniques," in Proc. 4th Int. Conf. Trends in Electronics and Informatics (ICOEI), Jun. 2020, pp. 728–733, doi: 10.1109/ICOEI48184.2020.9142909.
[29] T. Siddiqui and M. Mustaqeem, "Performance evaluation of software defect prediction with NASA dataset using machine learning techniques," International Journal of Information Technology, vol. 15, no. 8, pp. 4131–4139, Dec. 2023, doi: 10.1007/s41870-023-01528-9.