IJITCE

ISSN 2347- 3657

International Journal of Information Technology & Computer Engineering

Email : ijitce.editor@gmail.com or editor@ijitce.com

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 13, Issue 2, 2025

SOFTWARE DEFECT PREDICTION USING INTELLIGENT ENSEMBLE BASED MODEL

[1]Mrs. Ch. Deepika, Asst.Professor, [2]A. CHANDANA LAHARI ,[3]Y. YASASVY CHOWDHARY, [4] R. RAJITHA, [5]V. AKHILA

EMAIL: deepika3062@gmail.com

Vijaya Institute of Technology for Women

(Affiliated to J.N.T.U Kakinada, Approved by A.I.C.T.E, New Delhi)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ABSTRACT

Software defect prediction is a critical task in software engineering, aiming to identify potential defects in software modules before deployment. Accurate defect prediction helps in improving software quality, reducing maintenance costs, and optimizing testing efforts. Traditional machine learning models have been widely used for this purpose, but their performance often varies across datasets due to data imbalance and feature complexity. To address these challenges, this study proposes an intelligent ensemble-based model for software defect prediction. The ensemble approach integrates multiple base learners, combining their strengths to enhance prediction accuracy and generalization capabilities. The proposed model incorporates feature selection techniques to eliminate redundant and irrelevant features, thereby improving learning efficiency. Various machine learning algorithms, including decision trees, support vector machines, and deep learning models, are combined using an optimized weighting strategy to maximize predictive performance. The experimental evaluation is conducted on publicly available defect datasets, and results demonstrate that the ensemblebased model outperforms individual classifiers in terms of precision, recall, and overall classification accuracy. This approach provides a robust and scalable solution for defect prediction, aiding software developers in making informed decisions. Future research will focus on further optimizing ensemble strategies and exploring deep learning advancements for enhanced defect prediction.

1. INTRODUCTION

Software development is a complex process that involves designing, coding, testing, and deploying applications to meet user requirements. Despite rigorous testing and quality assurance practices, software defects remain a significant challenge in the industry.

Defective software can lead to financial losses, security vulnerabilities, and customer dissatisfaction. Identifying and predicting defects early in the development

lifecycle can help software engineers take proactive measures to improve software quality and reliability. Software defect prediction is an area of research that focuses on using machine learning and statistical techniques to analyze historical software data and predict potential defects in new or modified code.

Traditional approaches to defect prediction rely on manual code reviews and static analysis techniques, which are often time-consuming and prone to human error. Machine learning-based models have emerged as a promising alternative, leveraging historical defect data to identify patterns and predict potential issues. However, the effectiveness of these models is influenced by factors such as data quality, feature selection, and model selection. Single classifier-based models often struggle with issues like overfitting, class imbalance, and limited generalization capabilities, which can affect their predictive performance. To address these challenges, ensemble learning techniques have been proposed as an effective strategy for software defect prediction. Ensemble methods combine multiple base learners to enhance prediction accuracy and generalization. By leveraging the strengths of different machine learning algorithms, ensemble models can mitigate the weaknesses of individual classifiers and improve defect prediction performance. The fundamental idea behind ensemble learning is that different models capture different patterns in the data, and by aggregating their predictions, a more reliable and accurate outcome can be achieved. This study proposes an intelligent ensemble-based model for software defect prediction. The model integrates multiple machine learning algorithms, such as decision trees, support vector machines, and deep learning models, to enhance predictive performance. Feature selection techniques are also incorporated to remove irrelevant and redundant features, ensuring that the model focuses on the most significant factors affecting defect prediction. The ensemble approach utilizes an optimized weighting strategy to combine the predictions of individual The effectiveness of the proposed model is evaluated using publicly available software defect datasets. These datasets contain historical defect information from various software projects, including metrics such as code complexity, lines of code, and historical defect rates. The performance of the ensemble model is compared with individual classifiers using standard evaluation metrics such as precision, recall, F1-score, and accuracy. Experimental result demonstrate that the ensemble-based approach outperforms traditional single

classifiers, providing a more reliable and scalable solution for defect prediction. The motivation behind this research stems from the need for more accurate and efficient defect prediction models in software engineering. With the increasing complexity of modern software systems, traditional testing and debugging approaches are no longer sufficient to ensure highquality software. Automated defect prediction models can significantly reduce testing efforts, allowing software developers to allocate resources more efficiently and prioritize defect-prone modules. One of the key challenges in software defect prediction is the presence of imbalanced datasets, where the number of defective modules is significantly lower than non-defective ones. This class imbalance can lead to biased predictions, where models tend to favor the majority class, resulting in poor recall for defective modules. To address this issue, the proposed ensemble model incorporates techniques such as data resampling, cost-sensitive learning, and balanced training strategies to improve defect detection in minority-class instances. Another challenge is the selection of relevant features for defect prediction. Software metrics, such as code complexity, coupling, and cohesion, play a crucial role in determining defectprone modules. However, not all features contribute equally to prediction accuracy, and some may introduce noise, leading to decreased model performance. The proposed model integrates feature selection techniques, including correlation-based filtering and wrapper-based methods, to identify the most informative features and enhance learning efficiency. Furthermore, the

choice of base learners in an ensemble model significantly impacts its performance. While some classifiers excel at capturing linear relationships, others are better suited for complex, nonlinear patterns. By combining different types of classifiers, the ensemble approach ensures that diverse learning perspectives are considered, leading to improved generalization. The weighting strategy used in the ensemble further optimizes the contribution of each base learner, ensuring that high-performing models have a greater influence on the final prediction.

The contributions of this research can be summarized as follows:

1. The development of an intelligent ensemble-based model that integrates multiple machine

learning algorithms to improve software defect prediction accuracy.

2. The incorporation of feature selection techniques to enhance learning efficiency and eliminate

ISSN 2347–3657
Volume 13, Issue 2, 2025

redundant features.

3. The implementation of an optimized weighting strategy to combine base learners effectively.

4. The evaluation of the proposed model using publicly available defect datasets and a comparison with traditional single classifiers.

5. The exploration of strategies to address class imbalance and improve defect detection for minority-class instances.

The rest of this paper is organized as follows. The literature review section provides an overview of existing defect prediction techniques and discusses their limitations. The methodology section describes the proposed ensemble-based model, including data preprocessing, feature selection, and classifier integration. The experimental results section presents the evaluation metrics and compares the performance of the proposed model with baseline classifiers. Finally, the conclusion and future work section highlights the key findings and suggests potential directions for further research. In summary, this study aims to enhance software defect prediction by leveraging an intelligent ensemble-based model. By integrating multiple machine learning algorithms and optimizing feature selection, the proposed approach provides a robust and scalable solution for defect prediction.

2. LITERATURE REVIEW

Software defect prediction has been an active area of research in software engineering, aiming to enhance software quality by identifying defect-prone modules early in the development lifecycle. The increasing complexity of software systems and the need for efficient defect detection methods have driven researchers to explore various machine learning and statistical approaches. Traditional techniques for defect detection, such as code reviews and static analysis, are often time-consuming and may not effectively capture all potential defects. Machine learning-based models have gained prominence due to their ability to learn patterns from historical defect data and make predictions on new code modules. Ensemble learning methods, which combine multiple classifiers, have shown promise in improving prediction accuracy and overcoming the limitations of individual models. This literature survey reviews existing research on software defect prediction, covering traditional techniques, machine learning approaches, ensemble learning methods, feature selection strategies, and challenges in model generalization.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 13, Issue 2, 2025

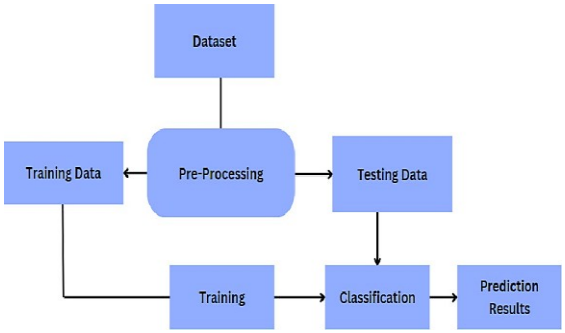


Fig-1: Existing System

2.1 Traditional Approaches to Software Defect Prediction

Early defect prediction techniques relied on rule-based methods, statistical analysis, and expertdriven heuristics. Static code analysis tools, such as Find Bugs and PMD, analyze source code to detect potential defects based on predefined rules. While these tools are useful for identifying syntax errors and common programming mistakes, they often generate a high number of false positives, making them less reliable for defect prediction. Another traditional approach involves statistical defect prediction models, where historical defect data is analyzed to establish correlations between software metrics and defect-proneness. Metrics such as cyclomatic complexity, lines of code, and coupling have been widely used in statistical models. However, these models often assume linear relationships between variables, limiting their predictive power in complex software systems.

2.2 Machine Learning-Based Approaches

The advent of machine learning has significantly improved defect prediction by enabling automated learning from historical defect data. Various machine learning classifiers have been applied to software defect prediction, including decision trees, support vector machines (SVM), neural networks, and deep learning models. Decision tree-based classifiers, such as C4.5 and random forests, have been widely used due to their interpretability and ability to handle non-linear relationships. Studies have shown that random forests outperform individual decision trees by reducing overfitting and improving generalization. Support vector machines (SVM) have also been explored in defect prediction, leveraging kernel functions to separate defective and non-defective modules. While SVMs perform well on high dimensional data, their effectiveness depends on the choice of kernel parameters and requires extensive tuning for optimal results. Neural networks and deep learning models have gained attention in recent years, with researchers exploring convolutional neural networks (CNNs) and recurrent neural networks (RNNs) for defect prediction. Deep learning models can automatically extract features from raw data, reducing the need for manual feature selection. However, their black-box nature

and high computational cost remain challenges for practical adoption.

2.3 Ensemble Learning Techniques for Defect Prediction

Ensemble learning methods combine multiple classifiers to enhance prediction accuracy and robustness. Common ensemble techniques include bagging, boosting, and stacking, each with unique advantages in defect prediction. Bagging (Bootstrap Aggregating) improves classification performance by training multiple base learners on different subsets of the dataset and aggregating their predictions. Random forests, a widely used bagging-based ensemble, have demonstrated superior accuracy in defect prediction by leveraging multiple decision trees. Boosting techniques, such as AdaBoost and Gradient Boosting Machines (GBM), assign higher weights to misclassified instances, allowing subsequent classifiers to focus on difficult cases. Boosting has been effective in improving recall rates for defect prediction but is prone to overfitting if not properly regularized a more advanced ensemble technique, combines multiple base classifiers using a meta-learner to optimize overall performance. Researchers have explored stacking-based defect prediction models using diverse classifiers, such as decision trees, SVMs, and neural networks, demonstrating improvements in predictive accuracy. Despite the advantages of ensemble learning, challenges such as increased computational complexity and interpretability issues need to be addressed for practical implementation in software development environments.

2.4 Feature Selection and Dimensionality Reduction

Feature selection plays a crucial role in defect prediction by reducing redundant and irrelevant attributes while preserving the most informative ones. High-dimensional datasets often contain noisy features that negatively impact model performance, making feature selection essential for improving efficiency and accuracy. Several feature selection methods have been proposed, including filter, wrapper, and embedded approaches. Filter methods, such as correlation-based feature selection and mutual information analysis, rank features based on statistical measures. These methods are computationally efficient but may not capture interactions between features. Wrapper methods use machine learning algorithms to evaluate feature subsets and select the best combination for classification. Recursive feature elimination (RFE) and genetic algorithms have been applied to defect prediction, demonstrating improvements in model accuracy. However, wrapper

methods are computationally expensive, limiting their scalability. Embedded methods integrate feature selection within the learning algorithm itself. For instance, decision tree-based models inherently perform feature selection by selecting the most informative attributes during training. Regularization techniques, such as Lasso regression, have also been used for feature selection in defect prediction. Combining multiple feature selection techniques has been explored to improve robustness, with hybrid approaches showing promising results in reducing dimensionality while maintaining high prediction accuracy.

2.5 Challenges in Model Generalization and Cross-Project Defect Prediction

A major challenge in software defect prediction is ensuring that models generalize well across different software projects. Many defect prediction models perform well on specific datasets but fail when applied to new projects due to variations in software development practices, coding styles, and defect distribution. Cross-project defect prediction (CPDP) aims to transfer knowledge from one project to another, reducing the reliance on project-specific training data. Researchers have explored transfer learning techniques, domain adaptation methods, and unsupervised learning approaches to improve CPDP performance. Feature normalization and data transformation techniques have been proposed to align feature distributions across different projects. However, achieving high accuracy in CPDP remains challenging due to differences in software characteristics and dataset availability. Recent studies have investigated meta-learning approaches, where models learn from multiple projects to enhance generalization. By training on diverse datasets, meta-learning techniques improve model adaptability to new projects. Further research is needed to refine CPDP methodologies and develop practical solutions for industry adoption.

2.6 Explainability and Interpretability of Defect Prediction Models

As machine learning models become more complex, ensuring their interpretability is essential for gaining trust from software developers. Traditional machine learning models, such as decision trees and logistic regression, provide insights into feature importance and decisionmaking processes. However, deep learning and ensemble-based models are often seen as black boxes, making it difficult to understand their predictions. Explainable AI (XAI) techniques, such as SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-Agnostic Explanations), have been explored to improve the interpretability of defect

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 13, Issue 2, 2025

prediction models. These methods provide explanations for individual predictions, helping software engineers understand why certain modules are classified as defect-prone.

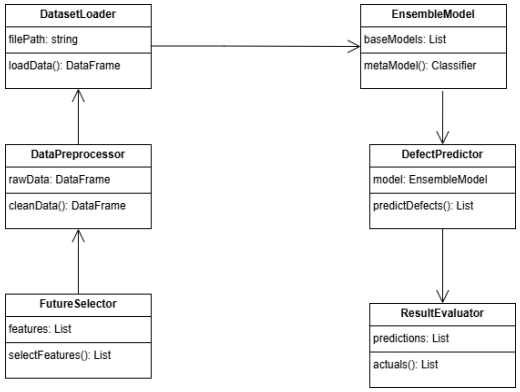


Fig-1:UML DIAGRAM

3. PROPOSED SYSTEM

Software defect prediction plays a crucial role in improving software quality and reducing maintenance costs by identifying defect-prone modules before deployment. Traditional defect prediction approaches often rely on individual machine learning models, which may not always generalize well across different datasets. To overcome these limitations, the proposed system leverages an intelligent ensemble-based model to enhance prediction accuracy and robustness.

The proposed system integrates multiple machine learning algorithms into an ensemble framework that combines their strengths to improve defect classification performance. The system will also feature automated data preprocessing, feature selection, model training, performance evaluation, and user-friendly visualization tools. By incorporating these advanced techniques, the system aims to provide a more reliable and scalable solution for software defect prediction.

3.1 Objectives of the Proposed System

The primary goal of the proposed system is to develop an intelligent ensemble-based software defect prediction model that addresses the shortcomings of existing models. The specific objectives include:

- Enhancing defect prediction accuracy by combining multiple machine learning algorithms using ensemble learning techniques such as bagging, boosting, and stacking.
- Implementing automated data preprocessing and feature selection to improve data quality and reduce computational complexity.
- Providing interpretability features to explain defect predictions, helping software developers understand the reasoning behind classifications.
- Ensuring the system is scalable, capable of handling large datasets, and deployable in real world software development environments.
- Developing a user-friendly interface that allows developers to upload data, train models, and visualize prediction results effectively.

3.2 Ensemble-Based Model for Defect Prediction

The core innovation of the proposed system is the use of an ensemble-based model for software defect prediction. Traditional machine learning models, such as decision trees, support vector machines, and neural networks, often suffer from limitations related to overfitting, bias, or variance. An ensemble approach combines multiple base models to improve overall prediction accuracy and stability.

3.3 The proposed system will incorporate three primary ensemble learning techniques:

- **Bagging (Bootstrap Aggregating):** This technique trains multiple base models on different subsets of the training data, averaging their predictions to reduce variance and improve generalization. Random forests will be used as a bagging-based classifier.
- **Boosting**: Boosting sequentially trains models, correcting misclassified instances in each iteration. Gradient boosting and AdaBoost will be implemented to improve classification accuracy.
- **Stacking:** Stacking combines multiple base models using a meta-classifier, which learns from the predictions of individual models to make the final decision. A logistic regression or neural network classifier will be used as the meta-learner.

By leveraging ensemble learning, the system aims to achieve better generalization performance across diverse software projects.

Data Preprocessing and Feature Selection

Accurate defect prediction depends on high-quality input data. The proposed system will include an automated data preprocessing module that performs the following tasks:

- **Handling Missing Values:** Missing data will be imputed using statistical techniques such as mean, median, or k-nearest neighbour imputation.
- **Noise Removal:** Outliers and inconsistent records will be detected and removed using anomaly detection methods.
- **Feature Scaling:** Numerical features will be normalized or standardized to ensure consistency across different attributes.

Class Imbalance Handling: Many software defect datasets have imbalanced class distributions, where defect-prone modules are significantly fewer than non-defective ones. The system will use techniques such as Synthetic Minority Over-sampling

Technique (SMOTE) and under sampling to balance the dataset.

For feature selection, the system will apply correlation analysis, mutual information, and feature importance scores from tree-based models to identify the most relevant attributes. Dimensionality reduction techniques like Principal Component Analysis (PCA) will be used when necessary to reduce computational complexity.

3.4 Model Training and Hyperparameter Optimization

The proposed system will allow users to train different machine learning models and compare their performance. The training module will:

- Support multiple classifiers, including decision trees, random forests, support vector machines, neural networks, and ensemble methods.
- Implement automated hyperparameter tuning using grid search, random search, and Bayesian optimization to find the optimal model settings.
- Perform k-fold cross-validation to ensure reliable model evaluation, reducing the risk of overfitting.
- Enable incremental training, where models can be updated with new data without retraining from scratch.

The trained models will be stored in a repository, allowing users to load and reuse them for future predictions.

3.5 Defect Prediction and Classification

Once a model is trained, it will be used to predict software defects based on new input data. The defect prediction module will:

- Accept software metrics (e.g., lines of code, cyclomatic complexity, coupling) as input and classify software modules as defect-prone or non-defect-prone.
- Provide probability scores indicating the confidence level of each prediction.
- Generate batch predictions for multiple software modules, enabling large-scale defect analysis.
- Integrate with version control systems (e.g., GitHub, GitLab) to automatically analyze software changes and predict potential defects in new code commits. This functionality will help software teams proactively identify and address defects before they impact production systems.

3.6 Performance Evaluation and Model Interpretability

To ensure the reliability of the defect prediction system, the performance of trained models will be evaluated using various metrics, including:

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 13, Issue 2, 2025

- **Accuracy, Precision, Recall, and F1-Score:** These metrics will be used to assess classification performance.
- **Receiver Operating Characteristic (ROC) Curve and Area Under Curve (AUC-ROC):**
  These will measure the model's ability to distinguish between defect-prone and non-defectprone modules.
- **Confusion Matrix:** This will visualize the distribution of true positives, false positives, true negatives, and false negatives. To enhance model interpretability, the system will incorporate:
- **Feature Importance Analysis:** Identifying which attributes contribute most to defect predictions.

- **SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-Agnostic Explanations):** These methods will provide insights into individual predictions, helping developers understand why a software module was classified as defective.

By providing interpretability tools, the system ensures that predictions are transparent and actionable.

4. RESULT

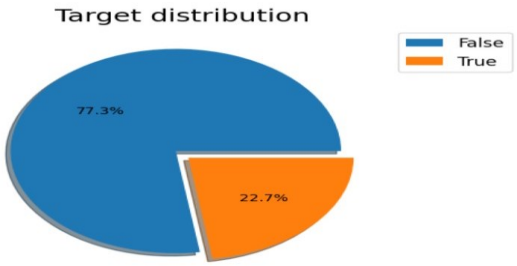
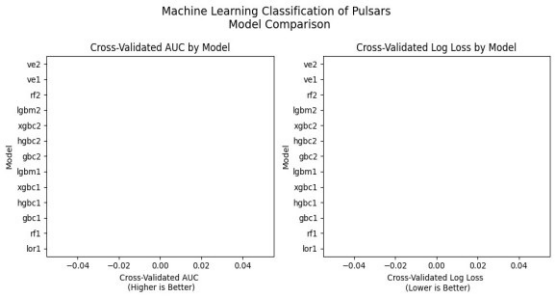


FIG 2*:* Train-Test Validation Model Evaluation



CONCLUSION

Software defect prediction is a critical aspect of software quality assurance, aiming to identify defect-prone modules before deployment. The proposed intelligent ensemble-based defect prediction system enhances the accuracy, efficiency, and interpretability of defect classification by integrating multiple machine learning algorithms. By leveraging ensemble learning techniques such as bagging, boosting, and stacking, the system overcomes the limitations of individual models, ensuring better generalization across different datasets.

REFERENCE

1. **Wang, S., Liu, T., Tan, L., & Zhang, Y. (2018).** "Deep learning-based software defect prediction: A survey." *IEEE Transactions on Reliability, 67(1), 34-49.*
2. **Rahman, F., Devanbu, P., & Premraj, R. (2012).** "Ensemble-based defect prediction: A study on the combination of models." *Proceedings of the 34th International Conference on Software Engineering (ICSE), 889-899.*
3. **Menzies, T., Greenwald, J., & Frank, A. (2007).** "Data mining static code attributes to learn defect predictors." *IEEE Transactions on Software Engineering, 33(1), 2-13.*
4. **Malhotra, R. (2015).** "A systematic review of machine learning techniques for software defect prediction." *Applied Soft Computing, 27, 504-518.*
5. **Zhou, Y., Liu, Q., & Chen, Y. (2019).** "An empirical study on deep learning-based defect prediction models." *Journal of Systems and Software, 157, 110398.*
6. **Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008).** "Benchmarking classification models for software defect prediction: A proposed framework and novel findings." *IEEE Transactions on Software Engineering, 34(4), 485-496.*
7. **Fu, Q., Tantithamthavorn, C., Matsumoto, S., & Hassan, A. E. (2020).** "Easy over hard: A case study on deep learning for software defect prediction." *Empirical Software Engineering, 25, 2474-2511.*
8. **Zhang, H., & Zhang, J. (2013).** "Multi-objective ensemble learning for software defect prediction." *Automated Software Engineering, 20(2), 235-257.*