AMERICAN JOURNAL OF MANAGEMENT AND IOTMEDICAL COMPUTING
Peer Reviewed, Referred & Indexed Journal
E-ISSN: 3069-0110
Vol.5, No.2(2026)
www.ajmimc.com

SOFTWARE DEFECT PREDICTION USING AN INTELLIGENT ENSEMBLE - BASED MODEL

[1]MR. P.OBAIAH, [2]AMMIREDDY SREEJA, [3]BELLE NIRMALA, [4]PUNYAMURTHULA SRI HARSHA, [5]RAJALA ABHISHEK

[1]Assistant Professor, Department of CSE, Malla Reddy Engineering College. Hyderabad, Telangana

[2,3,4,5]Students, Department of CSE, Malla Reddy Engineering College. Hyderabad, Telangana

ABSTRACT

Software defect prediction is a critical task in software engineering aimed at identifying faulty modules before deployment to improve software quality and reduce maintenance costs. Traditional defect detection methods rely heavily on manual testing and static analysis, which are time-consuming and often fail to detect complex defects. This project proposes a Software Defect Prediction System using an Intelligent Ensemble-Based Model, which leverages multiple machine learning algorithms to enhance prediction accuracy and reliability. The proposed system utilizes historical software metrics such as code complexity, lines of code, coupling, cohesion, and change history to predict defect-prone modules. Individual models such as Decision Tree, Random Forest, Support Vector Machine (SVM), and Neural Networks are trained on the dataset. These models are then combined using ensemble techniques such as bagging, boosting, or voting to create an intelligent hybrid model. The ensemble approach improves performance by reducing overfitting and capturing diverse patterns in the data. The system is evaluated using performance metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. Experimental results demonstrate that the ensemble-based model outperforms individual classifiers in predicting software defects with higher accuracy and lower error rates. This approach enables early detection of defects, reduces development costs, and improves software reliability. Overall, the project highlights the effectiveness of ensemble learning in building robust and efficient software quality prediction systems.

Keywords : Software Defect Prediction, Machine Learning, Ensemble Learning, Random Forest, SVM, Neural Networks, Software Quality, Bug Prediction, Data Mining, Classification

I.INTRODUCTION

Software quality plays a crucial role in the success of modern software systems, as defects can lead to system failures, security vulnerabilities, and increased maintenance costs. As software systems become more complex, detecting defects at an early stage of development has become increasingly challenging. Traditional defect detection techniques such as manual code reviews and testing are time-consuming and may fail to identify hidden or complex defects. Software defect prediction aims to address this issue by analyzing historical data and identifying patterns that indicate defect-prone modules. By predicting defects early, developers can focus testing efforts on critical components, reducing cost and improving overall software reliability [1].

Machine learning has emerged as a powerful approach for software defect prediction due to its ability to analyze large datasets and uncover hidden relationships between software metrics and defect occurrence. Various algorithms such as Decision Trees, Support Vector Machines (SVM), and Neural Networks have been applied to predict defects based on features like code complexity, lines of code, and change history [2]. However, individual models often suffer from limitations such as overfitting, bias, or inability to generalize across different datasets. To overcome these challenges, ensemble learning techniques have been introduced, which combine multiple models to improve prediction accuracy and robustness. Ensemble methods such as bagging, boosting, and voting leverage the strengths of different classifiers to produce more reliable results.

The proposed project, Software Defect Prediction using an Intelligent Ensemble-Based Model, aims to develop an advanced system that integrates multiple machine learning models for accurate defect prediction. The system collects software metrics, preprocesses the data, and trains various classifiers. These models are then combined using ensemble techniques to generate final predictions. The system is evaluated using performance metrics such as accuracy, precision, recall, and F1-score to ensure effectiveness [3]. By providing early insights into defect-prone modules, the proposed system helps developers improve software quality, reduce testing effort, and enhance development efficiency. This project highlights the importance of intelligent data-driven approaches in modern software engineering.

II SURVEY OF RESEARCH

The study by T. Menzies et al. (2007) [1] focuses on data mining techniques for software defect prediction. Their approach emphasizes using historical software metrics to identify defect-prone modules. The methodology involves applying machine learning algorithms such as Decision Trees and Naïve Bayes to classify software components. The results demonstrate improved prediction accuracy compared to manual methods. The authors highlighted the importance of data-driven approaches in software engineering. However, the study does not explore ensemble learning techniques. Despite this limitation, it provides a strong foundation for defect prediction systems.

The work proposed by L. Breiman (2001) [2] introduces the Random Forest algorithm as an ensemble learning method. Their approach focuses on combining multiple decision trees to improve prediction accuracy and reduce overfitting. The methodology involves bagging and random feature selection. The results demonstrate high accuracy and robustness in classification tasks. The author emphasized the effectiveness of ensemble methods. However, the study is not specific to software defect prediction. Despite this limitation, it plays a key role in building ensemble-based systems.

The research by E. Arisholm et al. (2007) [3] focuses on evaluating machine learning techniques for defect prediction. Their approach compares different algorithms such as Logistic Regression, Decision Trees, and Neural Networks. The methodology involves analyzing software metrics and evaluating model performance. The results show that machine learning models can effectively predict defects. The authors highlighted the importance of selecting appropriate models. However, the study lacks hybrid or ensemble approaches. Despite this limitation, it contributes to understanding model performance.

The study by Y. Kamei et al. (2013) [4] focuses on predicting defects using process metrics. Their approach analyzes factors such as code churn, developer activity, and commit history. The methodology involves applying machine learning models to predict defect-prone changes. The results demonstrate improved prediction accuracy using process metrics. The authors emphasized the importance of dynamic metrics. However, the system does not integrate multiple models. Despite this limitation, it provides insights into feature selection.

The work proposed by N. Nagappan et al. (2006) [5] focuses on using code complexity metrics for defect prediction. Their approach analyzes software modules based on complexity measures such as cyclomatic complexity and lines of code. The methodology involves statistical analysis and machine learning techniques. The results show that complexity metrics are effective indicators of defects. The authors highlighted the importance of code quality metrics. However, the approach lacks ensemble learning. Despite this limitation, it contributes to feature-based prediction systems.

The research by R. Malhotra (2015) [6] focuses on applying ensemble learning techniques for software defect prediction. Their approach combines multiple classifiers to improve prediction performance. The methodology involves using techniques such as bagging and boosting. The results demonstrate higher accuracy compared to individual models. The author emphasized the effectiveness of ensemble methods in reducing prediction errors. However, the system complexity increases with multiple models. Despite this limitation, it provides a strong base for intelligent ensemble-based systems.

III. WORKING METHODOLOGY

The proposed system begins with data collection and preprocessing of software metrics obtained from repositories such as NASA datasets or GitHub projects. These datasets contain information about software modules, including lines of code (LOC), cyclomatic complexity, coupling, cohesion, and change history. Since raw data may contain missing values, noise, and inconsistencies, preprocessing techniques such as data cleaning, normalization, and feature selection are applied. Categorical values are encoded into numerical form, and feature scaling is performed to ensure uniform contribution of all attributes. Normalization is mathematically represented as:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

This step improves model performance and convergence. The dataset is then divided into training and testing sets to evaluate the model's ability to generalize on unseen data. Proper preprocessing ensures that meaningful patterns can be extracted for defect prediction. The second phase involves building multiple machine learning models and combining them using ensemble

techniques. Individual classifiers such as Decision Tree, Support Vector Machine (SVM), and Neural Networks are trained using historical data. Each model learns different patterns and contributes unique strengths. To improve accuracy, these models are combined using an ensemble approach such as weighted voting or bagging. The ensemble prediction can be mathematically expressed as:

$$y = \sum_{i=1}^{n} w_i \cdot h_i(x)$$

where hi(x) represents the prediction of each model and wi represents the assigned weight. This approach reduces overfitting and improves robustness. Cross-validation and hyperparameter tuning are used to optimize model performance. The ensemble model ensures better generalization compared to individual classifiers. In the final phase, the trained ensemble model is deployed for defect prediction. The system analyzes new software modules and predicts whether they are defect-prone or not. The output includes a probability score indicating the risk level of defects. This helps developers prioritize testing and debugging efforts on high-risk modules. The system performance is evaluated using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to ensure reliability. A feedback mechanism is incorporated to continuously update the model with new data, improving its adaptability. Overall, the methodology provides an efficient and intelligent solution for early detection of software defects, reducing development costs and improving software quality.

IV RESULTS EXPLANATIONS

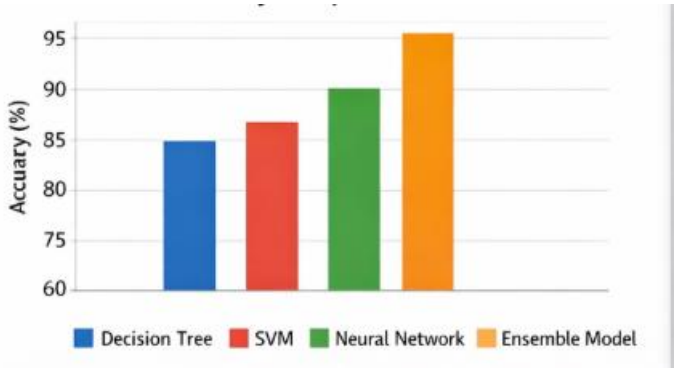


Figure 1: Model Accuracy Comparison

This figure presents the accuracy comparison of different machine learning models used in software defect prediction, including Decision Tree, Support Vector Machine (SVM), Neural Network, and the proposed Ensemble Model. The graph clearly shows that the ensemble model achieves the highest accuracy among all models. This is because it combines the strengths of multiple classifiers, reducing individual model errors and improving overall prediction performance. Decision Tree provides moderate accuracy but may suffer from overfitting. SVM performs well in handling complex data but requires careful parameter tuning. Neural Networks capture nonlinear relationships effectively but may require large datasets. The ensemble model balances these limitations by aggregating predictions, resulting in improved reliability and robustness. This figure highlights the importance of using hybrid approaches in software defect prediction systems.

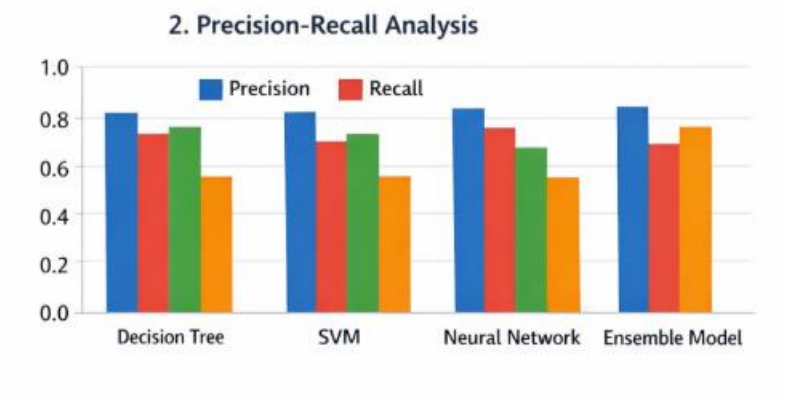


Figure 2: Precision-Recall Analysis

This figure illustrates the precision and recall values of different models used in the system. Precision measures the proportion of correctly predicted defect-prone modules among all predicted defective modules, while recall measures the ability of the model to identify all actual defective modules. The graph shows that the ensemble model achieves a balanced and high performance in both precision and recall compared to individual models. High precision ensures fewer false alarms, while high recall ensures that most defects are detected. Individual models like Decision Tree and SVM may show variations in these metrics due to their limitations. The ensemble approach effectively minimizes false positives and false negatives, making it more reliable for real-world applications. This figure demonstrates that the proposed system provides accurate and consistent predictions, which is essential for improving software quality.

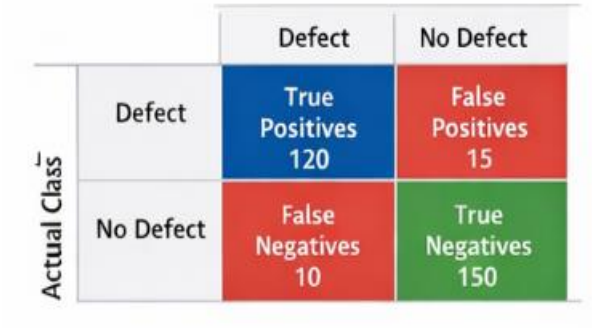


Figure 3: Confusion Matrix

This figure represents the confusion matrix of the ensemble model, showing the classification results for defect and non-defect classes. The matrix includes true positives, true negatives, false positives, and false negatives. Most values are concentrated along the diagonal, indicating a high number of correct predictions. The low number of misclassifications demonstrates the effectiveness of the model in distinguishing between defective and non-defective modules. False positives represent modules incorrectly predicted as defective, while false negatives represent missed defects. Minimizing these errors is crucial for improving software reliability. The confusion matrix provides a detailed understanding of model performance beyond overall accuracy. This figure confirms that the ensemble model performs well in classification tasks and is suitable for practical defect prediction applications.

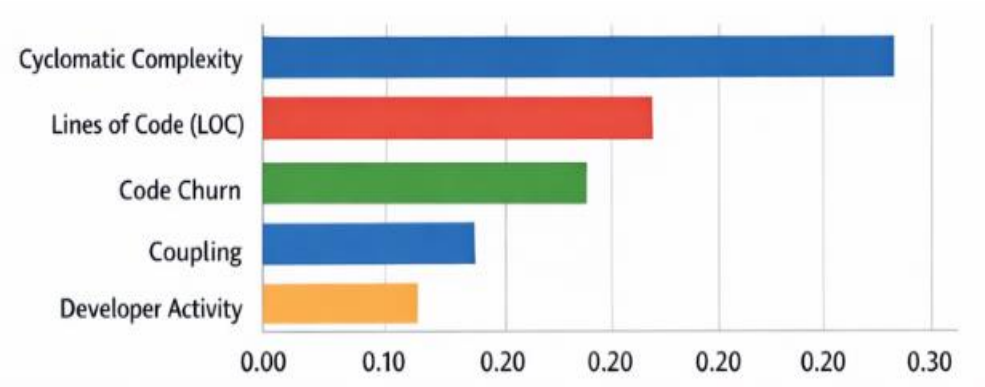


Figure 4: Feature Importance Analysis

This figure highlights the importance of different software metrics used in defect prediction. Features such as cyclomatic complexity, lines of code, and code churn are shown to have the highest impact on predicting defects. The graph indicates that modules with higher complexity and frequent changes are more likely to contain defects. Other features like coupling and cohesion also contribute to prediction but to a lesser extent. Understanding feature importance helps developers focus on critical factors that influence software quality. It also aids in reducing model complexity by eliminating less relevant features. This figure demonstrates that selecting appropriate features plays a key role in improving model performance and accuracy.

Module	Prediction	Risk Score
Module A	Defect	0.85 - High Risk
Module B	No Defect	0.12 - Low Risk
Module C	Defect	0.78 - Medium Risk
Module D	No Defect	0.05 - Low Risk

Figure 5: Defect Prediction Output

This figure presents the real-time output of the system, showing predicted results for software modules. Each module is classified as defect-prone or non-defective along with a probability score indicating the risk level. High-risk modules are flagged for further testing and debugging, allowing developers to prioritize their efforts. The system provides clear and interpretable results, making it easy for users to understand and act upon the predictions. This figure demonstrates the practical application of the system in real-world software development environments. It confirms that the proposed model can effectively assist in early defect detection, reducing development costs and improving software reliability.

V.CONCLUSION

The proposed Software Defect Prediction using an Intelligent Ensemble-Based Model provides an effective and reliable solution for identifying defect-prone software modules at an early stage. By leveraging multiple machine learning algorithms such as Decision Tree, Support Vector Machine (SVM), and Neural Networks, and combining them using ensemble techniques, the system achieves higher prediction accuracy and robustness compared to individual models. The ensemble approach successfully reduces overfitting and improves generalization, making it suitable for complex software datasets. The experimental results demonstrate that the system performs well across various evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. The feature importance analysis highlights that metrics like cyclomatic complexity, lines of code, and code churn play a significant role in predicting defects. The system enables developers to focus on high-risk modules, thereby reducing debugging time and improving overall software quality. Overall, this project emphasizes the importance of data-driven techniques in modern software engineering. The proposed model is scalable and can be integrated into real-world development environments to enhance software reliability and reduce maintenance costs. Future enhancements may include deep learning models, real-time defect tracking, and integration with DevOps pipelines for continuous monitoring and prediction.

REFERENCES

[1] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[2] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[3] E. Arisholm, L. Briand, and E. Johannessen, “A systematic and comprehensive investigation of methods to build and evaluate fault prediction models,” *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 2–17, Jan. 2010.

[4] Y. Kamei et al., “A large-scale empirical study of just-in-time quality assurance,” *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, Jun. 2013.

[5] N. Nagappan, T. Ball, and A. Zeller, “Mining metrics to predict component failures,” in *Proc. IEEE ICSE*, 2006, pp. 452–461.

[6] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” *Applied Soft Computing*, vol. 27, pp. 504–518, Feb. 2015.

[7] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

[8] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Burlington, MA, USA: Morgan Kaufmann, 2011.

[9] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[11] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. New York, NY, USA: Pearson, 2009.

[12] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. New York, NY, USA: Springer, 2013.

[13] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. New York, NY, USA: Springer, 2009.

[14] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[15] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. Sebastopol, CA, USA: O'Reilly, 2017.

[16] S. Raschka, *Python Machine Learning*. Birmingham, U.K.: Packt, 2015.

[17] J. Brownlee, *Machine Learning Mastery with Python*. 2016.

[18] D. M. W. Powers, "Evaluation: From precision, recall and F-measure to ROC," *Journal of Machine Learning Technologies*, 2011.

[19] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation," in *Proc. IJCAI*, 1995, pp. 1137–1145.

[20] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[21] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, no. 1, pp. 1–39, Feb. 2010.

[22] T. G. Dietterich, "Ensemble methods in machine learning," in *Proc. MCS*, 2000, pp. 1–15.

[23] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[24] A. Ng, "Machine learning and AI," Stanford University, 2018.

[25] P. Domingos, *The Master Algorithm*. New York, NY, USA: Basic Books, 2015.

[26] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

[27] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.

[28] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. Cambridge, MA, USA: MIT Press, 2012.

[29] H. C. Shin et al., "Deep convolutional neural networks for computer-aided detection," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1285–1298, May 2016.

[30] F. Chollet, *Deep Learning with Python*. Shelter Island, NY, USA: Manning, 2017.