International Journal for Novel Research in Economics , Finance and Management
www.ijnrefm.com
ISSN:3048-7722
Volume 4, Issue 4, July-Aug-2026, PP: 01-5

A Machine Learning-Based Framework for Early Software Bug Prediction

G.Preethi[1], K . Arundhathi[2]

[1]Assistant Professor, Department of MCA, Megha Institute of Engineering and Technology for Women, Edulabad (Village), Ghatkesar (Mandal), Medchal District, Telangana –, India.

[2]MCA Student, Department of MCA, Megha Institute of Engineering and Technology for Women, Edulabad (Village), Ghatkesar (Mandal), Medchal District, Telangana –, India.

Abstract – **The rapid growth of software applications across industrial, commercial, and scientific domains has significantly increased the demand for reliable and high-quality software systems. , identifying defects during the early stages of development has become an essential requirement for reducing maintenance costs, improving software reliability, and enhancing overall product quality. Traditional software testing approaches primarily depend on manual inspection and extensive debugging procedures, which are often time-consuming, resource-intensive, and incapable of detecting all potential defects before software deployment. Recent advancements in Machine Learning (ML) have introduced intelligent data-driven techniques capable of analyzing historical software metrics to predict defect-prone modules, thereby supporting proactive quality assurance throughout the software development life cycle. This study presents an intelligent proposed framework utilizes software engineering metrics collected from the JM1 repository to train multiple supervised machine learning models for identifying defective software modules. The framework incorporates comprehensive data preprocessing, including label encoding, one-hot encoding, feature scaling, and dataset standardization before model training. Seven supervised learning algorithms, enabling objective evaluation of each classifier's predictive capability. The proposed framework is evaluated using widely accepted performance measures including Accuracy, Precision, Recall, F1-Score, and Root Mean Square Error (RMSE). Experimental analysis demonstrates that the Random Forest classifier provides the highest overall prediction performance, achieving approximately 81% classification accuracy together with superior precision, recall, F1-score, and the lowest RMSE among the evaluated models. methodology for software defect prediction. The developed framework offers a reliable, reducing maintenance effort, and supporting more efficient software engineering practices.**

Keywords **- Machine Learning, Software Bug Prediction, Software Defect Prediction, Early Bug Detection, Software Quality Assurance, Predictive Analytics**

I. INTRODUCTION

Software quality has become one of the most important factors influencing the success of modern software systems. As software applications continue to expand in size, functionality, and complexity, maintaining reliability throughout the software development life cycle has become increasingly challenging. Software defects introduced during design or implementation can lead to system failures, increased maintenance costs, security vulnerabilities, and reduced user satisfaction if they are not detected at an early stage. Consequently, predicting software defects before deployment has emerged as a critical research area aimed at improving software quality while minimizing development time and operational expenses.

Conventional software testing and debugging techniques primarily rely on manual code inspection, testing procedures, and developer expertise to identify faulty software modules. Although these approaches remain important, they often require significant human effort engineering datasets has encouraged researchers to investigate Machine Learning (ML) techniques capable of automatically learning defect patterns from historical software metrics. These intelligent predictive models enable software engineers to prioritize testing activities by identifying modules with a higher probability of containing defects.

Among various software engineering datasets, the NASA PROMISE JM1 dataset has become one of the most widely used benchmark datasets for software defect prediction. The dataset contains software metrics describing source code characteristics, including McCabe complexity metrics, Halstead metrics, lines of code measurements, and branch-related attributes, together with defect labels indicating whether individual software modules contain faults. These software metrics provide valuable information for supervised machine learning algorithms to distinguish defective modules from fault-free components based on historical development data.

This study proposes an intelligent proposed framework applies comprehensive preprocessing techniques, including label encoding, one-hot encoding, feature scaling, and data standardization, before training multiple supervised learning algorithms. Seven classification models— and 20% testing data to ensure reliable evaluation of prediction performance.

upload software metric datasets, perform automated preprocessing, train prediction models, and analyze software defect predictions through an interactive interface. The effectiveness of the proposed

frameworkevaluated models, confirming its suitability for early software defect prediction and intelligent software quality assurance.

II. LITERATURE SURVEY

Software defect prediction has become an active area of research because of its significantthroughout the development life cycle. Early software defect prediction techniques primarily relied on statistical analysis, software complexity metrics, and manually designed prediction rules to identify fault-prone software modules. Although these traditional approaches provided useful insights into software quality, they frequently struggled often produced limited prediction accuracy when applied to large-scale software projects.

The advancement of Machine Learning (ML) has introduced more effective solutions for software defect prediction by enabling automated analysis of historical software engineering data. Researchers have extensively investigated supervised learning algorithms such as Naïve Bayes, Decision Trees, Support Vector Machines, Logistic Regression, Artificial Neural Networks, Random Forest, and K-Nearest Neighbors for identifying defect-prone software modules. These algorithms learn predictive relationships directly from software metrics and have demonstrated substantial improvements over conventional rule-based.

Among these techniques, Random Forest has consistently demonstrated superior performance because of its ensemble learning strategy, robustness against overfitting, and ability to analyze high-dimensional software metrics. Several comparative studies have reported that Random Forest achieves higher prediction accuracy than individual classifiers such as Decision Trees, Naïve Bayes, and Logistic Regression when applied to benchmark datasets including the NASA PROMISE JM1 repository. Similarly, Support Vector Machines have shown strong classification capability for binary software defect prediction by constructing optimal decision boundaries between defective and non-defective software modules. Artificial Neural Networks have further improved prediction performance by learning complex nonlinear relationships among software engineering metrics.

The NASA JM1 dataset has becomeprediction models. The dataset contains software metrics based on McCabe complexity measures, Halstead software metrics, lines of code measurements, and branch count attributes, providing comprehensive information for training supervised machine learning models. Numerous studies have utilized this dataset to compare different classification algorithms using Comparative experimental analyses consistently demonstrate that ensemble learning approaches generally outperform conventional single-classifier models in software defect prediction tasks.

Despite considerable progress in software bug prediction, several challenges remain unresolved, including dataset imbalance, feature redundancy, model generalization across different software projects, and computational efficiency. Many existing studies evaluate only restricting comprehensive performance comparison. The proposed study addresses these limitations by developing an intelligent software bug prediction framework that integrates systematic preprocessing, multiple supervised learning algorithms, and extensive comparative evaluation using the NASA JM1 dataset.effective software engineering practices.

III. PROPOSED METHODOLOGY

Unlike conventional debugging approaches that depend primarily on manual testing and developer experience, the proposed framework automatically analyzes software engineering metrics or fault-free. The system integrates comprehensive data preprocessing, feature transformation, supervised machine learning, and comparative model evaluation to improve software quality while reducing debugging effort and maintenance costs. The overall framework consists of six major stages: dataset acquisition, The prediction process begins with dataset acquisition, where software engineering metrics are obtained from the NASA PROMISE JM1 dataset. The dataset contains 10,885 software modules described by 22 software metrics, including McCabe complexity metrics, Halstead software metrics, lines of code measurements, branch count, and defect labels indicating whether a module contains software faults. These software metrics provide quantitative information describing source code complexity and Following data collection, the dataset undergoes data preprocessing to improve data quality before model development. Missing values, duplicate records, and inconsistent entries are identified and corrected where necessary. The target variable is transformed using label encoding, categorical variables are processed using one-hot encoding, and feature scaling is performed to normalize the software metrics. These preprocessing operations ensure consistent numerical representation of all features and improve the learning capability of the classification models while reducing bias caused by different feature scales.

The trained classifiers subsequently perform software bug prediction by classifying software modules as either defective or non-defective based on their software metrics. Comparative analysis is performed across all implemented algorithms to determine the most suitable prediction model. Experimental observations demonstrate that the Random Forest classifier provides superior prediction performance because of its ensemble learning capability, robustness against overfitting, and ability to capture complex relationships among software engineering metrics.

International Journal for Novel Research in Economics , Finance and Management
www.ijnrefm.com
ISSN:3048-7722
Volume 4, Issue 4, July-Aug-2026, PP: 01-5

Finally, the optimized prediction model is integrated into a Django-based web application that enables users to upload software metric datasets, preprocess data, train machine learning models, and obtain software defect predictions through an interactive interface. The proposed framework is evaluated using Accuracy, Precision, Recall, F1-Score, and Root Mean Square Error (RMSE). Experimental findings confirm that the developed system provides reliable, scalable, and computationally efficient software bug prediction, supporting software developers in improving software quality and reducing maintenance effort.

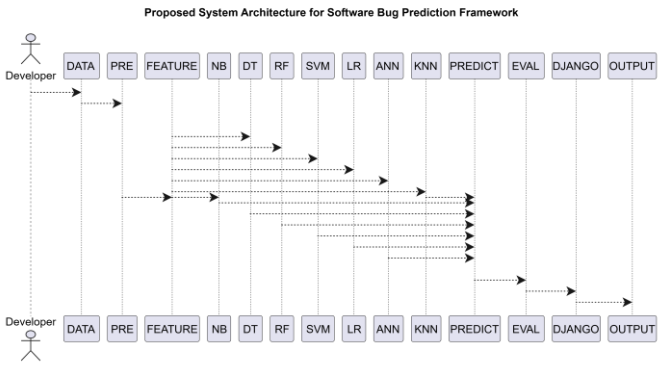

Fig. 1. Proposed system architecture

IV. METHODOLOGY

The proposed methodology integrates software metric analysis, data preprocessing, supervised machine learning, and performance evaluation to develop an efficient software bug prediction framework. The complete methodology consists of six sequential stages: dataset preparation, data preprocessing, feature transformation, model training, software defect prediction, and performance evaluation. This systematic workflow enables the framework to accurately identify fault-prone software modules while maintaining computational efficiency and prediction reliability.

The first stage involves dataset preparation, where software engineering data are collected from the NASA PROMISE JM1 dataset. The dataset contains 10,885 software instances described using 22 software metrics, including McCabe complexity measurements, Halstead software metrics, source code size metrics, branch count, and defect labels. These metrics quantitatively represent software module characteristics and provide the basis for machine learning-based.

During collected dataset is carefully prepared before model training. Missing values and duplicate entries are removed to improve dataset consistency, while label encoding converts the target defect labels into numerical form suitable for supervised learning. Categorical attributes are transformed using one-hot encoding, and feature scaling standardizes the software metrics to ensure uniform numerical representation across all variables. These preprocessing operations improve classifier stability and reduce the influence of feature magnitude during model learning.

Following preprocessing, the framework performs feature transformation by separating the predictor variables from the target defect labels. The software metrics are used as input features, while the defect attribute serves as the output class indicating defective or non-defective software modules. The processed dataset is then partitioned into 80% training data and 20% testing data, enabling objective evaluation of model performance on previously unseen software samples.

The prepared training dataset is subsequently supplied to seven Each classifier independently learns defect prediction patterns from the software metrics and generates prediction models capable of classifying software modules according to their likelihood of containing defects. Comparative evaluation across multiple algorithms enables identification of the most accurate prediction model.

Finally, the developed models are evaluated using standard classification measures includingupload software metric datasets, perform automated bug prediction, visualize performance statistics, and identify defect-prone software modules through an interactive software quality assessment platform.

V. EXPERIMENTAL RESULTS AND DISCUSSION

The proposed software bug prediction framework was experimentally evaluated using the NASA PROMISE JM1 dataset to investigate the effectiveness of various supervised machine learning algorithms in identifying defective software modules. The primary objective of the experimental analysis was to compare the predictive capability of multiple classifiers and determine the most reliable algorithm for early software defect prediction. The experimental study considered both classification performance and prediction reliability using standard software quality evaluation metrics.

The experimental dataset consisted of 10,885 software modules described using 22 software engineering metrics, including McCabe complexity measures, Halstead metrics, source code measurements, branch count, and software defect labels. Prior to model training, the dataset underwent preprocessing through label encoding, one-hot encoding, feature scaling, and data standardization to ensure consistent numerical representation of software metrics. The available data were subsequently generalization.

Seven Comparative experimental analysis demonstrated that all implemented models achieved satisfactory software

defect prediction performance, although significant differences were observed among the evaluated classifiers. Among all evaluated algorithms, the Random Forest classifier demonstrated the best overall performance. Experimental results indicated that Random Forest achieved approximately 81% classification accuracy, 78.8% precision, 81.9% recall, an F1-score of approximately 77.8%, and the lowest RMSE value of approximately 0.425, outperforming Naïve Bayes, Decision Tree, Artificialexhibited comparatively lower testing performance. The ensemble learning mechanism of Random Forest contributed to improved generalization capability and reduced prediction error when compared with individual classification algorithms.

The proposed framework was further compared with previously published software defect prediction studies conducted using the same JM1 dataset. Comparative evaluation confirmed that the developed methodology achieved improved prediction accuracy over several earlier implementations, demonstrating the effectiveness of comprehensive preprocessing, comparative classifier evaluation, and ensemble learning. The optimized prediction model was subsequently integrated into a Django-based web application, enabling automated software defect prediction through an interactive interface.

VI. CONCLUSION

Integrates systematic data preprocessing, feature transformation, comparative machine learning analysis, and automated software defect prediction to support early fault detection during the software development life cycle. By utilizing 22 software engineering metrics together with supervised learning algorithms, the developed system effectively predicts software defects while reducing the dependence on manual testing and debugging activities.

The experimental evaluation compared seven swhen compared with individual classification techniques.

The proposed framework offers several practical advantages for software developers, testers, and quality assurance teams. further improves accessibility by allowing users to upload software metric datasets, perform automated software bug prediction, and visualize prediction results through an interactive interface.

In conclusion, the proposed machine learning framework provides an accurate, scalable, and computationally efficient solution integration of systematic preprocessing, comparative evaluation of multiple classifiers, and Random Forest-based prediction establishes an effective software quality assurance framework capable of supporting reliable software engineering practices. The developed system also provides a strong foundation for future intelligent software analytics platforms incorporating advanced ensemble learning techniques, explainable artificial intelligence, and automated software quality assessment for next-generation software development environments.

REFERENCES

1. A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, “Software bug prediction using machine learning approach,” International Journal of Advanced Computer Science and Applications, vol. 9, no. 1, pp. 78–86, 2018.
2. S. Delphine Immaculate, M. Farida Begam, and M. Floramary, “Software bug prediction using supervised machine learning algorithms,” in Proc. Int. Conf. Data Science and Communication (IconDSC), Bangalore, India, 2019, pp. 1–7.
3. D. Sharma and P. Chandra, “Software fault prediction using machine learning techniques,” in Smart Computing and Informatics, Springer, 2018, pp. 541–549.
4. K. Tanaka, A. Monden, and Z. Yücel, “Prediction of software defects using automated machine learning,” in Proc. 20th IEEE/ACIS Int. Conf. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Toyama, Japan, 2019, pp. 490–494.
5. X. Zhao, W. Zhang, and N. Ai, “Analysis of attribute selection method based on IDOE in software fault prediction,” in Proc. Int. Conf. Intelligent Computing, Automation and Systems (ICICAS), Chongqing, China, 2019, pp. 741–745.
6. Z. B. G. Aydin and R. Samli, “Performance evaluation of machine learning algorithms on NASA defect prediction datasets,” in Proc. 5th Int. Conf. Computer Science and Engineering (UBMK), Diyarbakir, Turkey, 2020, pp. 1–3.
7. R. P. and P. Kambli, “Predicting software bugs using artificial neural network-based machine learning techniques,” in Proc. IEEE Int. Conf. Innovation in Technology (INOCON), Bengaluru, India, 2020, pp. 1–5.
8. 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, 2007.
9. C. L. Prabha and N. Shivakumar, “Software defect prediction using machine learning techniques,” in Proc. 4th Int. Conf. Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2020, pp. 728–733.
10. C. M. Manjula, L. F. Prasad, and A. Arya, “Software defect prediction using data mining and machine learning techniques,” International Journal of Database Theory and Application, vol. 8, no. 3, pp. 179–190, 2015.
11. A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, “Software defect prediction analysis using machine learning techniques,” Sustainability, vol. 15, no. 6, Art. no. 5517, 2023.

12. M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in Proc. IEEE Working Conf. Mining Software Repositories (MSR), Cape Town, South Africa, 2010, pp. 31–41.
13. R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Applied Soft Computing, vol. 27, pp. 504–518, 2015.
14. B. T. Jijo and A. M. Abdulazeez, "Classification based on decision tree algorithm for machine learning," Journal of Applied Science and Technology Trends, vol. 2, no. 1, pp. 20–28, 2021.
15. I. F. B. Tronto, J. D. S. da Silva, and N. Sant'Anna, "Artificial neural network prediction systems in software project management," Journal of Systems and Software, vol. 81, no. 3, pp. 356–367, 2008.
16. M. Hammad, A. Alqaddoumi, H. Al-Obaidy, and K. Almseidein, "Predicting software faults using K-nearest neighbors classification," International Journal of Computing and Digital Systems, vol. 8, no. 5, pp. 475–484, 2019.
17. M. A. Ibraigheeth and S. A. Fadzli, "Software project failure prediction using logistic regression modeling," in Proc. 2nd Int. Conf. Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 2020, pp. 1–5.
18. O. F. Arar and K. Ayan, "A feature-dependent Naïve Bayes approach for software defect prediction," Applied Soft Computing, vol. 59, pp. 197–209, 2017.
19. M. Efendioglu, A. Sen, and Y. Koroglu, "Bug prediction of SystemC models using machine learning," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 3, pp. 419–429, 2019.
20. I. Mehmood et al., "A novel approach to improve software defect prediction accuracy using machine learning," IEEE Access, vol. 11, pp. 63579–63597, 2023.
21. D. P. Ramesh Babu and P. Krishna Rao, "Reliable and secured banking management system through machine learning," Journal of Applied Science and Computations, vol. 6, no. 4, pp. 377–382, Apr. 2019.
22. D. P. Ramesh Babu, "Automated student document analysis and query system using OCR and conversational AI," pp. 358–366, 2026.