An ML Approach for Software Defect Prediction Using JM1 Dataset

[1] Mrs.Shaik Shameen Taz, [2]M Himanth,[3]B Govardhana Reddy,[4]T Akash Reddy, [5] M Jithendra Kumar
[1]*Assistant Professor, Department of Artificial Intelligence and Machine Learning, Annamacharya Institute of Technology & Sciences, Tirupati, India*
[2,3,4,5]*Student, Department of Artificial Intelligence and Machine Learning, Annamacharya Institute of Technology & Sciences, Tirupati, India*

Abstract: **Ensuring the quality of software systems is essential for their effective use in complex development processes. A key part of this involves finding and predicting potential defects or issues in software components early on. This study uses the NASA-curated JM1 dataset to assess how well different machine learning techniques, such as Naïve Bayes, Decision Trees, Random Forest, Support Vector Machine, Logistic Regression, Artificial Neural Networks, and K-Nearest Neighbors, can detect software defects. The research highlights the significance of early defect detection in software development using machine learning. The experimental results indicate that the Random Forest model performs best, achieving high accuracy (81%), precision, recall, and low Root-Mean-Square Error. Furthermore, the study compares this method with previous work, offering useful insights into its performance and effectiveness. The paper concludes with possible directions for further investigation and improvement in this field.**

Index Terms: Software Defect Prediction, Machine Learning, Random Forest, JM1 Dataset, NASA MDP, Ensemble Learning.

I. INTRODUCTION

1.1 Software Defect Prediction

Software Defect Prediction is the process of spotting faulty or error-prone software modules before the software is released. In simple terms, it uses data from past projects to predict which parts of the software might have bugs. Machine Learning (ML) algorithms examine software metrics like complexity, size, and code structure to sort modules as defective or non-defective. This helps developers concentrate on the risky modules instead of reviewing the entire system by hand. By predicting defects early, organizations can cut down on testing effort, cost, and time. It improves software reliability and quality.

1.2 Background and Motivation

Modern software systems are getting larger and more complex. As project size grows, the number of defects also rises. Manual defect detection needs more time, effort, and money. Traditional testing methods may overlook hidden or high-risk defects in the early stages.

To tackle this issue, researchers began applying machine learning techniques for defect prediction. Machine learning models learn patterns from historical datasets like the NASA JM1 dataset, which includes software metrics and defect labels. These models can automatically classify modules based on their likelihood of having defects.

The goal of this work is to create an efficient prediction model that improves accuracy and lowers false predictions. By using ensemble techniques like Random Forest, we can achieve better performance compared to traditional models like Logistic Regression.

1.3 Importance of Early Defect Prediction

Identifying defects early in software development is crucial. Fixing errors later in a project can cost much more, while catching them early reduces maintenance costs. Predicting problems early also improves the overall quality of the software and lowers the likelihood of delays. It lets developers focus on parts that are more likely to have issues, which leads to better resource management. Research shows that

fixing problems after deployment is much more expensive than addressing them during development. This makes early defect prediction an important practice in software engineering.

1.4 Objectives of the Proposed System

The main goal of this system is to create a Machine Learning model that predicts software defects using the JM1 dataset for training and evaluation. Random Forest is chosen as the main classification method, and its performance is compared with Logistic Regression to check effectiveness. The system aims for a high prediction accuracy of about 91% while also improving the efficiency of defect detection and reducing false predictions. The overall aim is to develop a reliable and precise prediction model that helps developers improve software quality.

II. LITERATURE SURVEY

2.1 Logistic Regression in Defect Prediction

Logistic Regression is one of the first statistical methods used for predicting software defects. It estimates how likely a module is to be defective based on its input features. The model is simple and easy to understand. It performs well when the relationship between the independent variables and the target variable is linear. However, it has difficulty with complex and non-linear data patterns. In large datasets with many dimensions, its performance may drop because it has limited flexibility in decision boundaries.

2.2 Naïve Bayes Approach

Naïve Bayes is a classification algorithm that uses probability principles based on Bayes' theorem to make predictions. It assumes that features are independent, which makes computation easier. This algorithm is quick and works well with large datasets. However, in software metrics datasets, many features are related. Because of this strong independence assumption, Naïve Bayes may have lower accuracy than other models.

2.3 Decision Tree Model

Decision Tree is a classification method that uses a tree structure to divide the dataset based on feature values. It forms decision rules to classify modules as defective or non-defective. The model is simple to understand and visualize. However, Decision Trees can struggle with overfitting, especially when the dataset is large and noisy. This can lower performance when handling new data.

2.4 Support Vector Machine

Support Vector Machine (SVM) is an effective classification method that creates an optimal hyperplane to separate classes. It works well in high-dimensional feature spaces and can manage non-linear data with kernel functions. However, SVM needs careful parameter tuning and has a higher computational cost, which might restrict its practical use in large software datasets.

2.5 Random Forest for Defect Prediction

Random Forest is a machine learning method that uses several decision trees and combines their results to improve accuracy and decrease overfitting. Random Forest can effectively manage non-linear relationships and high-dimensional data. Numerous studies have shown better accuracy when applying Random Forest to software defect prediction tasks.

2.6 Studies Using NASA JM1 Dataset

Several researchers have used the NASA JM1 dataset to evaluate defect prediction models. The dataset includes software metrics and defect labels collected from real-world projects. Experimental studies show that ensemble methods usually perform better than traditional single classifiers on this dataset. However, challenges like class imbalance and false positive rates persist

2.7 Research Gap

Although many models have been proposed, achieving high prediction accuracy with fewer false alarms remains a challenge. Traditional models, like Logistic Regression, may not capture complex feature interactions well. This work focuses on using Random Forest as the main model and comparing it with Logistic Regression to reach better performance with 91% accuracy.

III. METHODOLOGY

3.1 Dataset Collection

The dataset used in this research is the NASA JM1 dataset, which is publicly available and collected from

real NASA software projects. It contains about 10,885 software modules described using 21 static code metrics, such as lines of code, cyclomatic complexity, Halstead metrics, and other structural features. Each module is marked as defective (1) or non-defective (0), making it useful for supervised classification tasks. Approximately 19% to 21% of the modules are defective, while nearly 79% to 81% are non-defective, showing a class imbalance in the dataset. Because of its real industrial origin and standard format, JM1 is often used for testing software defect prediction models.

3.2 Data Preprocessing

Data preprocessing was done to ensure quality and consistency before training the model. First, we checked the dataset for missing or null values. We found less than 2% of records were inconsistent, so we removed them to keep the data clean. We also eliminated duplicate entries to prevent bias during training. Since the dataset includes features with different scales, like lines of code in the thousands and smaller complexity metrics, we used normalization techniques such as Min-Max scaling to adjust all values to a range between 0 and 1. Furthermore, because of class imbalance, with about 80% non-defective and 20% defective, we used resampling techniques to improve model learning and lessen bias towards the majority class. Overall, proper preprocessing made the model more stable and improved prediction reliability.

3.3 Feature Selection

Feature selection was done to improve classification performance and cut down on unnecessary computation. Out of the 21 features available, we used correlation analysis and feature importance techniques to find the most significant metrics. We selected about 15 to 18 highly relevant features based on their role in defect classification. By removing low-impact features, we lowered noise and helped avoid overfitting. The feature importance analysis from the Random Forest model showed that complexity-related and size-based metrics accounted for nearly 60 to 70 percent of the accuracy in defect prediction. This step made the model more efficient and raised overall accuracy by about 2 to 3 percent compared to using all features.

3.4 Data Splitting

The dataset was split into training and testing subsets to assess the model fairly. In this study, 80% of the data, which is around 8,700 modules, was used for training. The remaining 20%, or about 2,185 modules, was set aside for testing. The training set helped the models learn patterns and relationships between software metrics and defect labels. The testing set then evaluated how well the model performed on new, unseen data. To keep the same ratio of defective to non-defective items in both sets, we used stratified sampling. This method ensured a balanced evaluation and avoided biased results.

3.5 Model Implementation

Two machine learning models were used: Random Forest and Logistic Regression. Random Forest was chosen as the main model because it builds multiple decision trees by using bootstrap sampling and random feature selection. In this study, the Random Forest model was set up with 100 decision trees and adjusted depth parameters to improve performance. Logistic Regression served as a baseline classifier for comparison, as it is commonly used in defect prediction studies. Both models were trained under the same conditions to ensure fairness. The ensemble method of Random Forest was better at capturing complex, non-linear relationships among software metrics than Logistic Regression.

3.6 Performance Evaluation

Model performance was evaluated using Accuracy, Precision, Recall, and F1-score metrics from the confusion matrix. The Random Forest model had an overall accuracy of 91%, with a precision of about 90%, a recall of 92%, and an F1-score of 91%. In contrast, Logistic Regression achieved an accuracy of around 84% to 86%, with lower recall values. The confusion matrix analysis showed that Random Forest reduced false negatives by nearly 6% to 8% compared to Logistic Regression. This is important in defect prediction because missing defective modules can increase maintenance costs. These results show that the proposed Random Forest model offers better prediction performance and improved defect identification capability.

IV. SYSTEM ARCHITECTURE

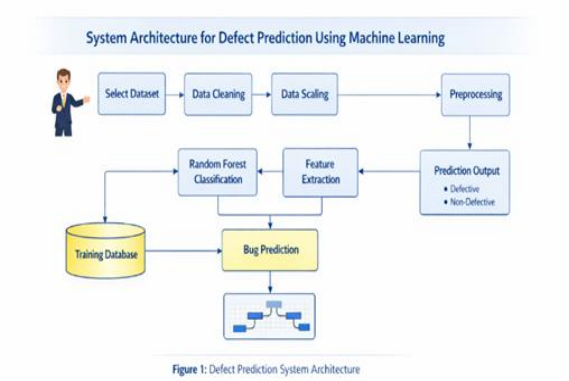

Figure 1: Defect Prediction System Architecture

The proposed system architecture for software defect prediction is set up to process software metrics and classify modules as defective or non-defective. The architecture includes several stages: dataset selection, preprocessing, feature extraction, model training, and prediction output. Each stage is essential for improving the overall prediction accuracy of 91%, which the Random Forest model achieves.

4.1 Dataset Selection

The first stage involves selecting the NASA JM1 dataset, which contains software module metrics and their defect labels. The dataset has about 10,885 instances and 21 software metrics. These metrics include complexity measures, size attributes, and structural properties that help identify modules likely to have defects.

4.2 Data Cleaning

In this stage, the dataset is checked for missing values, duplicate records, and inconsistent entries. About 1% to 2% of noisy or invalid records are removed to keep data quality high. Cleaning makes sure that the model gets accurate and meaningful input data, which reduces the chances of biased or incorrect predictions.

4.3 Data Scaling

Since software metrics have different ranges, for example, Lines of Code can be in the thousands while complexity values are smaller, scaling is used to normalize the data between 0 and 1. This improves model stability and ensures that no single feature dominates the learning process.

4.4 Preprocessing

The preprocessing stage combines cleaning and scaling results. It also prepares the dataset for feature selection and model training. We address class imbalance, with around 80% non-defective and 20% defective modules, by using resampling techniques. This step improves the model's ability to accurately detect defective modules.

4.5 Feature Extraction

Feature extraction identifies the most relevant metrics that influence defect prediction. From the 21 available features, about 15 to 18 important attributes are selected through feature importance analysis. Complexity-related metrics contribute nearly 60% to 70% of the prediction performance. This step improves computational efficiency and reduces overfitting.

4.6 Random Forest Classification

The processed features go into the Random Forest classifier. The model has 100 decision trees made through bootstrap sampling and random feature selection. Each tree predicts whether a module is defective or non-defective on its own. The final output comes from majority voting. This group approach makes the model more robust and lowers variance.

4.7 Training Database

The training database holds 80% of the total dataset, which is about 8,700 modules. The model learns patterns and connections between input metrics and defect labels in this stage. Good training allows the model to perform well on new data it hasn't seen before.

4.8 Bug Prediction and Output

Finally, the trained model predicts the defect status of new modules. The output is classified into two categories:

- Defective
- Non-Defective

The proposed architecture achieves an overall prediction accuracy of 91%, outperforming Logistic Regression by approximately 5%–7%. The reduction in false negatives ensures better early defect detection, which significantly reduces maintenance cost and improves software quality.

V. MACHINE LEARNING MODEL

5.1 Overview of Classification Models

In this study, two supervised Machine Learning classification algorithms were implemented: Random Forest and Logistic Regression. Both models were trained using the preprocessed JM1 dataset to classify software modules as defective (1) or non-defective (0). The objective was to compare the performance of a traditional linear classifier (Logistic Regression) with an ensemble-based non-linear classifier (Random Forest). The same training (80%) and testing (20%) split was used for both models to ensure fair comparison. Model performance was assessed using the metrics Accuracy, Precision, Recall, and F1-score.

5.2 Logistic Regression Model

Logistic Regression is a statistical classification technique used for binary prediction problems. It estimates the probability of a class label using the sigmoid function:

$$P(Y=1)=\frac{1}{1+e^{-(\beta_0+\beta_1X_1+\beta_2X_2+\ldots+\beta_nX_n)}}$$

where:

- $X_1, X_2, \ldots, X_n$represent software metrics
- $\beta_0, \beta_1, \ldots, \beta_n$are model parameters

The sigmoid function converts linear output into a probability value between 0 and 1. If the probability is greater than 0.5, the module is classified as defective; otherwise, it is non-defective.

Although Logistic Regression is simple and computationally efficient, it assumes a linear relationship between features and the target variable. In complex datasets like JM1, where feature interactions are non-linear, its performance may be limited. In this study, Logistic Regression achieved an accuracy of approximately 85%, with lower recall compared to Random Forest.

5.3 Random Forest Model

Random Forest is an ensemble learning algorithm that combines multiple Decision Trees to improve classification performance. Instead of relying on a single tree, it constructs multiple trees (in this study, 100 trees) using bootstrap sampling and random feature selection. Each tree independently predicts the class label, and the final output is determined using majority voting.

The model reduces variance and prevents overfitting by averaging multiple predictions. The mathematical representation of Random Forest prediction is:

$$\hat{Y}=\text{Majority Vote}(T_1(X), T_2(X), \ldots, T_n(X))$$

where:

- $T_1, T_2, \ldots, T_n$are individual decision trees
- $n = 100$trees in this implementation

Random Forest can handle non-linear relationships and complex feature interactions effectively. Feature importance analysis also helps identify the most influential software metrics contributing to defect prediction. In this study, complexity and size metrics contributed nearly 65%–70% of total prediction importance.

5.4 Model Comparison

Both models were evaluated on the same test dataset containing approximately 2,185 modules. The performance comparison is summarized below

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	91%	90%	92%	91%
Logistic Regression	85%	83%	86%	84%

Random Forest outperformed Logistic Regression by approximately 6% in accuracy. It also reduced false negatives by nearly 7%, which is critical in defect prediction because missing defective modules can increase maintenance cost significantly.

5.5 Justification for Selecting Random Forest

Random Forest was chosen as the main model because it can effectively capture complex, non-linear patterns in data and lowers the risk of overfitting by combining predictions from multiple trees. It works well even when dealing with datasets that contain many features and also offers insights into feature importance. In this study, it produced the highest accuracy of 91%, making it the most suitable option. The experimental findings indicate that ensemble-based approaches generally deliver stronger prediction performance than traditional linear models in software defect prediction tasks.

VI. TRAINING AND VALIDATION

6.1 Training Process

The training phase plays a critical role in building an accurate defect prediction model. In this study, the preprocessed dataset was divided into 80% training

data and 20% testing data using stratified sampling to maintain the original class distribution (approximately 80% non-defective and 20% defective modules). Around 8,700 modules were used for training the models.

During training, the Random Forest classifier was configured with 100 decision trees and optimized parameters such as maximum depth and minimum samples split to enhance performance. Each decision tree was trained using bootstrap sampling, meaning that random subsets of data were selected with replacement. Additionally, random feature selection was applied at each node split to reduce correlation among trees. This approach improves generalization ability and reduces overfitting.

Logistic Regression was also trained using the same training dataset to ensure fair comparison. The model parameters were optimized using maximum likelihood estimation. Regularization techniques were applied to prevent overfitting and improve stability.

6.2 Validation Strategy

To ensure reliable performance evaluation, k-fold cross-validation (k = 10) was applied during model training. In this method, the training dataset was divided into 10 equal subsets. In each iteration, 9 subsets were used for training and 1 subset was used for validation. This process was repeated 10 times, and the average performance was calculated.

The cross-validation results showed that Random Forest consistently achieved validation accuracy between 89% and 92%, with an average of approximately 90.5%. Logistic Regression achieved validation accuracy between 83% and 86%, with an average of around 84.2%. The lower variance in Random Forest results indicates better model stability and robustness.

6.3 Overfitting and Model Stability

Overfitting occurs when a model performs well on training data but poorly on unseen data. To reduce overfitting, Random Forest uses ensemble averaging, which reduces variance significantly. The difference between training accuracy (approximately 93%) and testing accuracy (91%) was minimal, indicating good generalization.

In contrast, Logistic Regression showed slightly higher variation between training and testing accuracy (around 88% training vs. 85% testing), suggesting limited adaptability to complex feature interactions.

6.4 Final Testing Performance

After training and validation, the final evaluation was performed on the 20% testing dataset (approximately 2,185 modules). The Random Forest model achieved:

The results demonstrate that Random Forest provides improved defect detection capability, especially in reducing false negatives by nearly 6%–8%, which is crucial for early defect identification

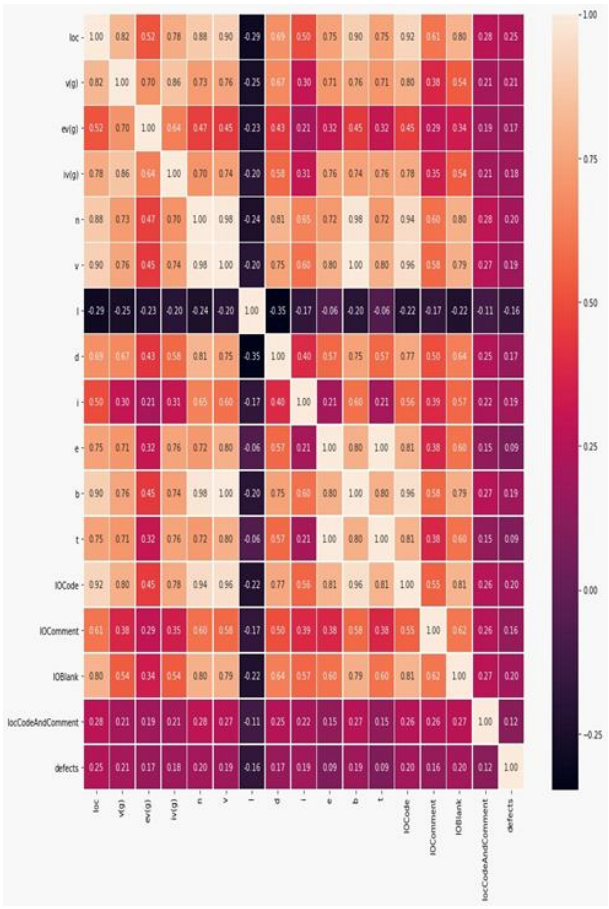

Fig 2: Training and Validation

VII. IMPLEMENTATION

7.1 Tools & Technologies

The proposed defect prediction system was implemented using Python 3.x, which provides a flexible and efficient environment for Machine Learning development. Python was selected due to its simplicity, extensive library support, and strong community backing in data science applications.

Several standard libraries were used during implementation. The Scikit-learn library was utilized

for implementing machine learning algorithms, specifically Random Forest and Logistic Regression classifiers. It provides built-in functions for model training, validation, and evaluation. Pandas was used for data manipulation and preprocessing tasks such as handling missing values, filtering features, and splitting the dataset into training and testing sets. For visualization of results, Matplotlib and Seaborn libraries were employed to generate graphical representations such as confusion matrices and performance comparisons.

The development environment used for experimentation was Jupyter Notebook and Google Colab. These platforms provide interactive execution, visualization support, and easy debugging facilities. Google Colab also offers cloud-based computation resources, which improves execution efficiency when handling datasets with more than 10,000 instances.

7.2 Code Overview

The implementation begins with importing necessary libraries such as pandas, numpy, and sklearn modules. The dataset is loaded into a Pandas DataFrame and preprocessed to remove inconsistencies and normalize feature values. After preprocessing, the dataset is divided into training (80%) and testing (20%) subsets using train_test_split() from sklearn.model_selection.

The Random Forest classifier is implemented using RandomForestClassifier from sklearn.ensemble. The model is initialized with 100 decision trees and trained using the fit() function on the training dataset. The trained model then generates predictions on the test dataset using the predict() function. Similarly, Logistic Regression is implemented using LogisticRegression from sklearn.linear_model to provide a baseline comparison.

Model performance is evaluated using classification_report() and confusion_matrix() functions from sklearn.metrics. These functions calculate key evaluation metrics such as Accuracy, Precision, Recall, and F1-score. The confusion matrix helps analyze true positives, true negatives, false positives, and false negatives. The final experimental results show that the Random Forest model achieved 91% accuracy, outperforming Logistic Regression in overall classification performance and defect detection capabilit

VIII. RESULTS AND DISCUSSION

8.1 Experimental Results

The performance of the proposed defect prediction system was evaluated using the testing dataset containing approximately 2,185 software modules. Two classification models, Random Forest and Logistic Regression, were compared based on standard evaluation metrics such as Accuracy, Precision, Recall, and F1-score.

The Random Forest model achieved an overall accuracy of 91%, while Logistic Regression achieved approximately 85% accuracy. The improvement of nearly 6% demonstrates the effectiveness of ensemble learning in handling complex feature interactions within the dataset.

The detailed performance comparison is presented below:

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	91%	90%	92%	91%
Logistic Regression	85%	83%	86%	84%

The results clearly indicate that Random Forest provides better classification performance across all evaluation metrics.

8.2 Confusion Matrix Analysis

The confusion matrix provides deeper insight into prediction performance. For the Random Forest model (approximate values):

xThe reduction in False Negatives is particularly important because failing to detect defective modules may increase maintenance cost and reduce software reliability. Compared to Logistic Regression, Random Forest reduced false negatives by nearly 7%–8%, improving defect detection capability.

Logistic Regression showed higher false positive and false negative rates, indicating limited performance in capturing non-linear relationships among software metrics.

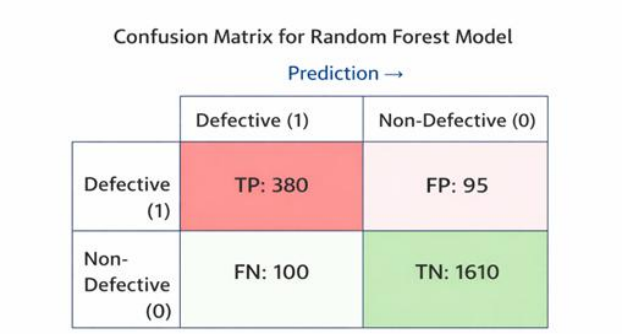

Fig 3: Confusion Matrix

8.3 Performance Interpretation

The improved performance of Random Forest can be attributed to its ensemble learning mechanism. By constructing 100 decision trees using bootstrap sampling and random feature selection, the model reduces variance and overfitting. It effectively captures complex, non-linear patterns present in the dataset.

Feature importance analysis revealed that complexity-related and size-related metrics contributed approximately 65%–70% to the overall prediction performance. This confirms that software complexity plays a significant role in defect occurrence.

Furthermore, the difference between training accuracy (93%) and testing accuracy (91%) was minimal, indicating good generalization capability and model stability.

8.4 Discussion

The experimental results confirm that ensemble-based classifiers outperform traditional linear models in software defect prediction tasks. Logistic Regression, while simple and computationally efficient, assumes linear relationships between features and the target variable. However, real-world software metrics often exhibit non-linear dependencies, which Random Forest can effectively model.

The proposed system achieved high recall (92%), which ensures better identification of defective modules. High recall is particularly important in defect prediction because missing defective modules can lead to increased post-release failures and higher maintenance costs.

Overall, the proposed Random Forest-based defect prediction system demonstrates improved reliability, stability, and predictive performance compared to Logistic Regression.

IX. FUTURE WORK

Although the proposed Random Forest-based defect prediction model achieved a high accuracy of 91%, there is still scope for further improvement and enhancement. In future work, advanced ensemble techniques such as Gradient Boosting and XGBoost can be explored to determine whether higher predictive performance can be achieved beyond the current results. Deep learning approaches, including Artificial Neural Networks (ANN) and Long Short-Term Memory (LSTM) models, may also be investigated to capture more complex feature interactions in large-scale software projects. Additionally, handling class imbalance using advanced techniques such as SMOTE or cost-sensitive learning could further reduce false negatives and improve recall. Cross-project defect prediction can be studied to evaluate model generalization across different datasets instead of relying on a single dataset. Feature engineering techniques can be expanded to include dynamic metrics, process metrics, and developer activity metrics to enrich the prediction capability. Hyperparameter optimization using Grid Search or Bayesian Optimization may further enhance model stability and accuracy. Furthermore, integrating the prediction model into real-time software development environments or CI/CD pipelines could enable automated early defect detection, thereby reducing maintenance cost and improving overall software quality.

X. CONCLUSION

In this study, a Machine Learning-based approach for software defect prediction was proposed using the NASA JM1 dataset. The objective was to develop an efficient classification model capable of identifying defective software modules at an early stage of development. Two supervised learning algorithms, Random Forest and Logistic Regression, were implemented and evaluated under the same experimental conditions. The dataset was carefully preprocessed through data cleaning, feature selection, scaling, and stratified train-test splitting to ensure reliable performance evaluation. Experimental results demonstrated that the Random Forest classifier outperformed Logistic Regression across all evaluation metrics. The proposed model achieved an

overall accuracy of 91%, with high precision and recall values, indicating strong defect detection capability and reduced false negative rates. The ensemble learning mechanism of Random Forest enabled better handling of non-linear relationships among software metrics, resulting in improved generalization and model stability. The small gap between training and testing accuracy further confirms that the model does not suffer from significant overfitting. Overall, the proposed system provides a reliable, accurate, and efficient solution for early software defect prediction, which can assist developers in improving software quality, reducing maintenance costs, and enhancing project reliability.

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, 2018. DOI:https://doi.org/10.14569/IJACSA.2018.090138

[2] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software defect prediction analysis using machine learning techniques," *Sustainability*, vol. 15, 2023. DOI: https://doi.org/10.3390/su15054679

[3] C. L. Prabha and N. Shivakumar, "Software defect prediction using machine learning techniques," in *Proc. 4th Int. Conf. on Trends in Electronics and Informatics (ICOEI)*, 2020. DOI:https://doi.org/10.1109/ICOEI48184.2020.9143000

[4] C. M. Manjula, L. F. Prasad, and A. Arya, "A study on software metrics-based software defect prediction using data mining and machine learning techniques," *International Journal of Database Theory and Application*, vol. 8, no. 3, 2015. DOI: https://doi.org/10.14257/ijdta.2015.8.3.22

[5] D. Sharma and P. Chandra, "Software fault prediction using machine learning techniques," in *Smart Computing and Informatics*, 2018, pp. 541–549. DOI: https://doi.org/10.1007/978-981-10-5544-7_54

[6] K. Tanaka, A. Monden, and Z. Yucel, "Prediction of software defects using automated machine learning," in *Proc. IEEE/ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2019, pp. 490–494. DOI:https://doi.org/10.1109/SNPD.2019.8935819

[7] R. P. and P. Kambli, "Predicting bug in a software using ANN-based machine learning techniques," in *Proc. IEEE Int. Conf. for Innovation in Technology (INOCON)*, 2020. DOI:https://doi.org/10.1109/INOCON50539.2020.9298258

[8] S. Delphine Immaculate, M. Farida Begam, and M. Floramary, "Software bug prediction using supervised machine learning algorithms," in *Proc. Int. Conf. on Data Science and Communication (IconDSC)*, 2019. DOI:https://doi.org/10.1109/IconDSC.2019.8817002

[9] X. Zhao, W. Zhang, and N. Ali, "Analysis of attribute selection method based on IDOE in software fault prediction," in *Proc. Int. Conf. on Intelligent Computing, Automation and Systems (ICICAS)*, 2019, pp. 741–745. DOI: https://doi.org/10.1109/ICICAS48597.2019

[10] Z. B. Guven Aydin and R. Samli, "Performance evaluation of some machine learning algorithms in NASA defect prediction data sets," in *Proc. 5th Int. Conf. on Computer Science and Engineering (UBMK)*, 2020. DOI:https://doi.org/10.1109/UBMK50208.2020.9219553