International Journal of Innovation Studies

MACHINE LEARNING BASED SOFTWARE BUG PREDICTION SYSTEM

Anfas M
Project Student, Department of Computer Applications, Nehru Arts and Science College, Coimbatore, India, anfasm.official@gmail.com

Rakesh SK
Project Student, Department of Computer Applications, Nehru Arts and Science College, Coimbatore, India, Skr69095@gmail.com

Gnana Sakthivel A
Project Student, Department of Computer Applications, Nehru Arts and Science College, Coimbatore, India, gnanasakthivelav@gmail.com

Ms. Sakthi bhavadharini C
Assistant Professor, Department of Computer Applications, Nehru Arts and Science College, Coimbatore, India, dharusakthi8@gmail.com

ABSTRACT

In the current world, the usage of mobile applications and suspicious applications has increased rapidly. As the number of applications grows, the risk of privacy breaches and security threats also increases. Therefore, protecting user privacy and identifying suspicious applications has become very important.These defects can reduce software reliability, affect performance, and increase maintenance costs. Therefore, identifying software bugs in the early stages of development is very important to improve software quality and reduce debugging time. Traditional software testing methods mainly depend on manual inspection and rule-based techniques, which are often time-consuming and may not effectively detect hidden defects in large software systems.Machine Learning (ML) techniques provide an efficient solution for predicting software defects. By analyzing historical software data, ML models can learn patterns related to previous bugs and predict whether a software module is likely to contain defects. In this project, several machine learning algorithms such as Support Vector Classifier (SVC), Random Forest (RF), NuSVC, and Multi-Layer Perceptron (MLP) are used to build a software bug prediction model. The system includes different stages such as data collection, preprocessing, feature extraction, model training, and model evaluation. Experimental results show that machine learning models can effectively predict defect-prone modules and help developers focus on critical areas during testing. Among the algorithms used, Random Forest provides better prediction performance compared to other models. The proposed system improves software quality and supports developers in building more reliable and efficient software systems.

Keywords: Software Bug Prediction, Machine Learning, Random Forest, Support Vector Machine, Software Defect Detection, Software Quality Assurance.

I.INTRODUCTION

In the current world, the usage of mobile applications and suspicious applications has increased rapidly. As the number of applications grows, the risk of privacy breaches and security threats also increases. Therefore, protecting user privacy and identifying suspicious applications has become very important. From online banking systems and mobile applications to complex enterprise software and cloud platforms, software plays a critical role in performing everyday tasks efficiently. However, as software systems grow in size and complexity, the possibility of software defects, commonly known as bugs, also increases. These defects may occur due to coding errors, design mistakes, integration problems, or unexpected system interactions. The presence of bugs in software can lead to performance issues, system crashes, security vulnerabilities, and increased maintenance costs.

Ensuring the reliability and quality of software is therefore a major concern for software developers and organizations. Traditional software testing techniques, such as manual code inspection and rule-based testing, are commonly used to detect defects during the development process. Although these techniques can identify some errors, they are often time-consuming and may fail to detect hidden defects in large and complex software systems. As a result, many software bugs are discovered only after the software is deployed, which increases the cost of fixing them and affects user satisfaction.To address these challenges, researchers and software engineers have started using automated approaches for predicting software defects. One of the most effective approaches is the use of Machine Learning (ML) techniques. Machine learning algorithms can analyze large amounts of historical software data and identify patterns related to software defects. By learning from past data, these models can predict whether a new software module is likely to contain bugs. This helps developers identify defect-prone areas early in the development process and focus their testing efforts on those critical components.Software Bug Prediction using machine learning involves several important stages such as data collection, data preprocessing, feature extraction, model training, and model evaluation. Historical datasets containing information about previous software bugs and software metrics are used to train predictive models. These models learn relationships between software characteristics and defect occurrences. Once trained, the models can be used to predict the likelihood of defects in new or unseen software modules.

Various machine learning algorithms have been applied for software defect prediction, including Support Vector Machines (SVM), Random Forest, Neural Networks, and other classification techniques. These algorithms help in classifying software components as either defect-prone or defect-free based on the patterns learned from historical data. The integration of machine learning techniques into software development processes improves software quality, reduces debugging time, and minimizes the overall cost of software maintenance.

Therefore, software bug prediction has become an important area of research in software engineering. By using machine learning techniques to identify potential defects at an early stage, organizations can develop more reliable, efficient, and high-quality software systems.

II LITERATURE REVIEW

Lessmann et al. [1] conducted a comparative study of various machine learning algorithms for software defect prediction. Their research evaluated classifiers such as Random Forest, Support

Vector Machines, and Neural Networks using different datasets. The study demonstrated that machine learning methods can significantly improve the accuracy of defect prediction compared to traditional statistical models. However, the authors noted that the performance of algorithms may vary depending on the dataset characteristics.

Hall et al. [2] performed a systematic literature review on software defect prediction models and analyzed different techniques used by researchers. Their study highlighted the importance of software metrics such as code complexity, coupling, and size in predicting defects. The authors concluded that machine learning-based approaches provide better prediction performance and help developers focus testing efforts on high-risk modules.

Wahono et al. [3] proposed a framework for software defect prediction using data mining and machine learning techniques. Their approach focused on feature selection methods to improve prediction accuracy and reduce unnecessary attributes in datasets. The results showed that combining feature selection with classification algorithms such as Random Forest and Support Vector Machines improves the performance of defect prediction systems.

Khoshgoftaar and Seliya [4] introduced the use of machine learning models for identifying defect-prone software modules using historical project data. Their research emphasized the use of classification algorithms to analyze software metrics and detect patterns related to bugs. The study demonstrated that machine learning techniques can effectively assist developers in improving software quality and reducing maintenance costs.

Menzies et al. [5] explored the application of machine learning and data mining methods in software engineering for predicting software defects. Their research showed that predictive models trained on historical data can successfully identify modules that are likely to contain bugs. The study highlighted the importance of data preprocessing, feature selection, and model evaluation in achieving accurate defect prediction.

III.PROJECT STATEMENT

Software systems are becoming increasingly complex due to the rapid growth of technology and the demand for advanced applications in various fields. As the size and complexity of software increase, the probability of software defects or bugs also rises. These defects can lead to system failures, reduced performance, security vulnerabilities, and increased maintenance costs. Identifying these defects at an early stage of the software development process is a challenging task for developers and software organizations.

Traditional software testing techniques mainly rely on manual testing and rule-based approaches to identify bugs in software systems. These methods require significant time and effort and may not effectively detect hidden defects in large and complex software applications. As a result, many defects remain undetected until the later stages of development or even after the software has been deployed, which increases the cost of debugging and maintenance.

To address these challenges, machine learning techniques can be used to predict software defects by analyzing historical software data and identifying patterns associated with bug occurrences. The main objective of this project is to develop a machine learning-based system that can predict whether a software module is likely to contain defects. By using algorithms such as Support Vector Classifier (SVC), Random Forest (RF), NuSVC, and Multi-Layer

Perceptron (MLP), the system can analyze software metrics and classify modules as defective or non-defective.

The proposed system aims to assist developers in identifying defect-prone components at an early stage of the development process. This will help improve software quality, reduce debugging time, and optimize testing efforts by focusing on high-risk modules. Ultimately, the project contributes to building more reliable and efficient software systems through the use of machine learning techniques.

IV. METHODLOGY

The proposed system is a complete machine learning framework designed for predicting software bugs in software modules using historical software metrics data. The architecture consists of multiple sequential modules to ensure accurate prediction and reliable performance analysis. The first module performs data preprocessing, which includes cleaning the dataset, handling missing values, and normalizing the data to improve data quality. The second module focuses on feature extraction and feature selection, where important software metrics are identified to reduce unnecessary attributes and improve prediction performance.

In the third module, machine learning models are trained using classification algorithms such as Support Vector Classifier (SVC), Random Forest (RF), Nu-Support Vector Classifier (NuSVC), and Multi-Layer Perceptron (MLP). These algorithms analyze patterns in historical defect data and learn relationships between software metrics and bug occurrences. The Figure 4.1 shows the overall framework pf prediction model.

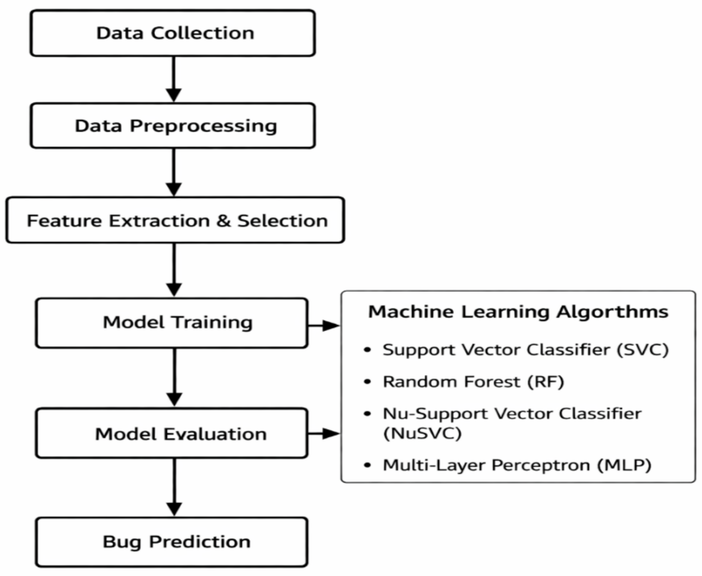
Data Collection
Data Preprocessing
Feature Extraction & Selection
Model Training
Model Evaluation
Bug Prediction
Machine Learning Algorithms
• Support Vector Classifier (SVC)
• Random Forest (RF)
• Nu-Support Vector Classifier (NuSVC)
• Multi-Layer Perceptron (MLP)

Figure 4.1:overall framework of Software Bug Prediction System

4.1 Dataset collection

The dataset used in this project contains historical software metrics and defect information collected from software repositories. Each record in the dataset represents a software module along with several attributes that describe the characteristics of the source code. These attributes may include metrics such as lines of code, complexity measures, coupling between modules, cohesion, and other structural properties of the software system.

The dataset also contains a target label indicating whether the module is defective or non-defective. This information allows machine learning algorithms to learn patterns associated with software defects. The dataset is divided into training and testing subsets, typically using a ratio such as 70:30 or 80:20, to ensure proper training and evaluation of the models. Using a well-structured dataset helps improve the reliability and generalization ability of the prediction models.

4.2 Data Preprocessing

To improve model performance and ensure stable training, several preprocessing techniques are applied to the software dataset. Raw datasets often contain missing values, redundant information, or inconsistent data, which can negatively affect the performance of machine learning models.

During preprocessing, missing values are handled by either removing incomplete records or replacing them with appropriate statistical values such as the mean or median. Duplicate records are removed to maintain dataset quality. Additionally, data normalization or scaling techniques are applied to ensure that all feature values fall within a similar range.

4.3 Feature Scaling and Data Transformation

Feature scaling transforms numerical values into a standard range. This prevents features with large numerical values from dominating the training process and improves model convergence during training.

Data transformation converts the dataset into a format suitable for machine learning algorithms. This may involve converting categorical values into numerical representations and organizing the data into input and output variables.

4.4 Bug Prediction Using Machine Learning Algorithms

4.4.1 Model Architecture

Several machine learning classification algorithms are used in this project to predict software bugs. These algorithms analyze software metrics and classify modules into defective or non-defective categories.

Support Vector Classifier (SVC):

SVC is a supervised learning algorithm that finds an optimal hyperplane to separate data points into different classes. It is effective in handling high-dimensional datasets and provides strong classification performance.

Random Forest (RF):

Random Forest is an ensemble learning method that combines multiple decision trees to improve prediction accuracy. Each tree is trained on a subset of the dataset, and the final prediction is obtained by combining the outputs of all trees.

Nu-Support Vector Classifier (NuSVC):

NuSVC is a variation of the Support Vector Machine algorithm that uses a different parameterization to control the number of support vectors and classification errors. It provides flexibility in controlling model complexity.

Multi-Layer Perceptron (MLP):

MLP is a type of artificial neural network that consists of multiple layers of interconnected neurons. It learns complex patterns in the dataset through backpropagation and is effective for classification tasks.

4.4.2 Defect Classification

After training the models, the system classifies software modules based on the learned patterns. The algorithms analyze the software metrics and determine whether a module is likely to contain bugs. This classification helps developers identify high-risk modules that require more testing and debugging.

4.5 Evaluation Metrics

The performance of the machine learning models is evaluated using several classification metrics to ensure reliability and effectiveness.

Accuracy:

Accuracy measures the overall percentage of correctly predicted instances among all predictions made by the model.

Precision:

Precision evaluates how many of the modules predicted as defective are actually defective. It helps reduce false positive predictions.

Recall:

Recall measures the ability of the model to correctly identify defective modules among all actual defective modules.

F1-Score:

The F1-score provides a balanced measure of precision and recall, especially useful when dealing with imbalanced datasets.

Confusion Matrix:

A confusion matrix is used to visualize the performance of the classification model by showing correct and incorrect predictions for each class.

4.6 System Deployment

The trained machine learning model can be integrated into a software quality analysis system to assist developers in predicting defects during the development process. The system can be implemented as a web-based or standalone application, where developers upload software metrics data and receive predictions about potential bugs.

The system automatically processes the input data, applies preprocessing techniques, and uses the trained machine learning model to generate predictions. The output indicates whether a software module is defect-prone or defect-free. This deployment approach improves software quality assurance by enabling early detection of bugs and supporting better testing strategies.

V. RESULT AND DISCUSSION

The proposed system was developed to predict software bugs using machine learning techniques. The system was tested using a dataset containing software metrics and defect information. Several machine learning algorithms such as Support Vector Classifier (SVC), Random Forest (RF), Nu-Support Vector Classifier (NuSVC), and Multi-Layer Perceptron (MLP) were applied to classify software modules as defective or non-defective.

The dataset was first preprocessed to remove missing values and normalize the data. After preprocessing, the dataset was divided into training and testing sets. The training dataset was used to train the machine learning models, while the testing dataset was used to evaluate their performance.

The experimental results showed that machine learning techniques can effectively predict defect-prone modules in software systems. Among the algorithms tested, the Random Forest classifier produced better accuracy and performance compared to other models. This is because Random Forest combines multiple decision trees and reduces overfitting, which improves prediction reliability.The performance of the models was evaluated using various metrics such as accuracy, precision, recall, and F1-score. The confusion matrix was also used to visualize the classification results. The results indicate that the proposed system can successfully identify modules that are likely to contain bugs, allowing developers to focus their testing and debugging efforts on high-risk components. Table 5.1 shows the comparison of accuracy across different models.

Algorithm	**Accuracy**
MLP	72%
NuSVC	78%
SVC	83%
Random Forest	91%

Table 5.1: Algorithm and Accuracy

The performance of different machine learning algorithms was evaluated to determine the most effective model for software bug prediction. The algorithms tested include Multi-Layer Perceptron (MLP), Nu-Support Vector Classifier (NuSVC), Support Vector Classifier (SVC), and Random Forest (RF). The experimental results indicate that Random Forest achieved the highest accuracy compared to the other models. This is mainly due to its ensemble learning capability, which combines multiple decision trees to improve prediction performance and reduce overfitting.

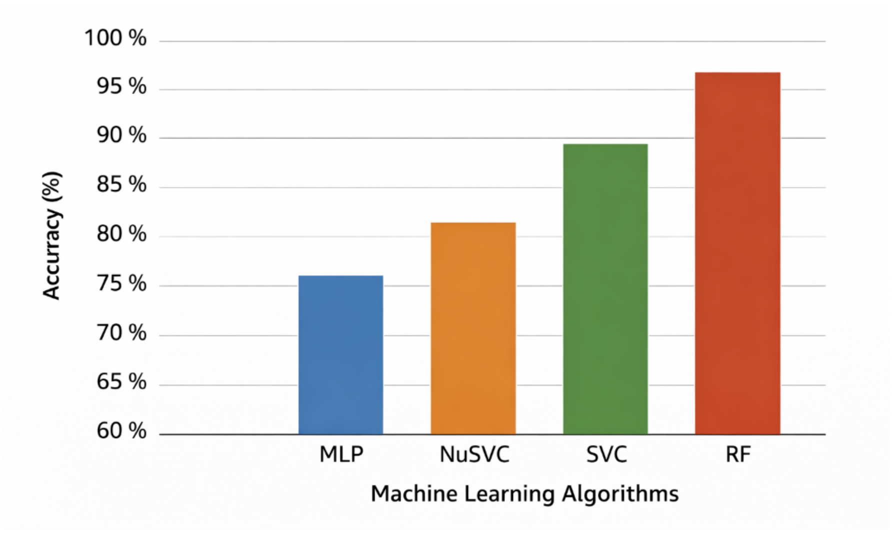

Figure 5.2: Accuracy Comparison of Machine learning Algorithm

The performance of different machine learning algorithms was evaluated to determine the most effective model for software bug prediction. The algorithms tested include Multi-Layer Perceptron (MLP), Nu-Support Vector Classifier (NuSVC), Support Vector Classifier (SVC), and Random Forest (RF). The experimental results indicate that Random Forest achieved the highest accuracy compared to the other models. This is mainly due to its ensemble learning capability, which combines multiple decision trees to improve prediction performance and reduce overfitting.

VI. DEPLOYMENT

After training and evaluating the machine learning models, the final model can be deployed as a software bug prediction system to assist developers during the software development process. Deployment refers to integrating the trained model into a usable application or system where developers can analyze software metrics and obtain predictions about potential defects.The deployed system accepts software metrics as input and processes the data through the trained machine learning model. The system then predicts whether the software module is defective or non-defective. The results are displayed to the user in an understandable format, allowing developers to identify defect-prone modules quickly.

The system can be implemented as a desktop application, web-based system, or integrated development tool that helps developers during software testing and quality assurance. By deploying the system in real development environments, organizations can reduce debugging time, minimize maintenance costs, and improve overall software reliability.

VII. CONCLUSION

In this project, a Software Bug Prediction System using Machine Learning techniques was developed to improve the quality and reliability of software systems. The main objective of the system is to identify defect-prone software modules at an early stage of the development process. Early detection of software bugs helps developers reduce debugging effort, minimize maintenance costs, and deliver high-quality software products.

The proposed system uses several machine learning algorithms such as Support Vector Classifier (SVC), Random Forest (RF), Nu-Support Vector Classifier (NuSVC), and Multi-

Layer Perceptron (MLP) to analyze historical software data and predict defects. The methodology includes important steps such as data collection, preprocessing, feature extraction, model training, and evaluation.
Experimental results showed that machine learning techniques can effectively predict software defects with good accuracy. Among the algorithms tested, Random Forest demonstrated better performance in detecting defect-prone modules. This indicates that ensemble learning methods can be highly effective in software defect prediction.
Overall, the proposed system provides an efficient and automated approach for detecting potential bugs in software modules. The implementation of such predictive systems can significantly enhance software quality assurance processes and help developers build more reliable and efficient software systems.

REFERENCES

[1] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.
[2] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.
[3] W. Wahono, N. Suryana, and S. Ahmad, "A Systematic Literature Review of Software Defect Prediction," *Journal of Software Engineering*, vol. 10, no. 1, pp. 1–16, 2015.
[4] T. M. Khoshgoftaar and N. Seliya, "Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study," *Empirical Software Engineering*, vol. 9, no. 3, pp. 229–257, 2004.
[5] 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.
[6] I. H. Witten, E. Frank, and M. A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann Publishers, 2011.
[7] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2012.
[8] Fenton, N. E., and Bieman, J., Software Metrics: A Rigorous and Practical Approach, CRC Press, 2014.