Journal of Science Engineering Technology and Management Science
Volume 03, Issue 05, May 2026 *ISSN: 3049-0952*
www.jsetms.com

Enhancing Software Maintenance Through Ensemble Learning-Based Bug Report Prediction

Ch. Satyanarayana Reddy[1], K.Pavani[2], R.Mounika[3]
#1 Assistant Professor in the Department of MCA, SRK Institute of Technology, Vijayawada.

#2 Assistant Professor & Head of Department of MCA, SRK Institute of Technology, Vijayawada.

#3 Student in the Department of MCA, SRK Institute of Technology, Vijayawada.

Abstract: In software development, analyzing large volumes of bug reports is a challenging and time-consuming task. This paper proposes an ensemble machine learning-based model combined with Natural Language Processing (NLP) techniques to automatically predict the nature of bug reports. Multiple classifiers, including Logistic Regression, Naïve Bayes, Support Vector Machine, and Random Forest, are integrated using a voting mechanism to improve prediction accuracy. Experimental results demonstrate that the proposed system significantly enhances classification performance and reduces manual effort in software maintenance.

Index Terms *- Bug Reports, Ensemble Learning, Natural Language Processing (NLP), Machine Learning, Voting Classifier, Software Maintenance, Text Classification, Random Forest, Support Vector Machine (SVM), Logistic Regression*

1. INTRODUCTION

Software testing is a fundamental phase in software engineering that ensures system reliability by identifying defects and inconsistencies. As software systems grow in size and complexity, the number of defects reported by users also increases significantly. These defects are documented as bug reports, which contain valuable information such as bug description, severity, priority, and reproduction steps. However, manually analyzing and classifying these bug reports becomes a tedious, time-consuming, and error-prone process due to the large volume of data .

To overcome these challenges, researchers have explored automated approaches using machine learning and Natural Language Processing (NLP) techniques. Existing methods mainly focus on predicting bug severity or priority, but they often ignore the classification of the nature of bugs, which is equally important for effective debugging and maintenance. Accurate identification of bug types helps developers prioritize tasks, allocate resources efficiently, and reduce resolution time.

In this paper, we propose an ensemble machine learning-based approach to automatically classify bug reports based on their nature. The system utilizes NLP techniques such as tokenization, lemmatization, and feature extraction (TF-IDF) to process textual

data. Multiple machine learning models, including Logistic Regression, Naïve Bayes, Support Vector Machine, and Random Forest, are combined using a Voting Classifier to enhance prediction accuracy. The proposed model aims to improve software maintenance efficiency by providing faster and more reliable bug classification.

Overall, this work contributes to the development of an intelligent and scalable bug analysis system that reduces manual effort and improves decision-making in software maintenance processes.

2. LITERATURE SURVEY

2.1 Using Natural Language Processing and Machine Learning Techniques to Characterize Configuration Bug Reports: A Study.

A RESEARCH In this study, a tool is created that accomplishes two goals: (1) given bug reports, it distinguishes configuration bug reports from non-configuration bug reports; and (2) once a bug report is recognized as a configuration bug report, the tool determines which particular configuration option the bug report is related to. This study begins with a review of related works that addressed software bug and bug report-related problems using machine learning technologies. The methods for machine learning and natural language processing are then covered. The motivation, experiment design and setup, and results analysis of the suggested tool's development process are then thoroughly explained. Both cross-validation and a comparable validation method are used to assess the tool's efficacy. The tool's effectiveness in recognizing configuration problem reports and the related configuration choices for those reports is demonstrated by the results. This study demonstrates the value of machine learning methods in resolving problems pertaining to bug reports. Additionally, it demonstrates how machine learning algorithms may understand the differences between configuration and non-configuration issue reports. There are several ways to enhance the produced tool to increase its effectiveness.

2.2 A method of non-bug report identification from bug report repository.

Misclassification during the identification and subsequent filtering of non-bug reports from the bug report repository is one of the most frequent problems that bug report studies address. Finding genuine bug reports takes time because irrelevant complaints must be filtered out, which increases expenses since more maintenance and work are needed to detect and address defects. As a result, this topic has been thoroughly researched and is discussed here. This paper suggests a way to use classification techniques to automatically detect non-bug reports in the bug report repository in order to address this issue. Here, three criteria are taken into consideration. Unigram and CamelCase, where CamelCase words are employed for feature growth, are the first bug report features utilized. In order to choose the best word weighting system for this assignment, five different schemes are compared. Finally, the primary methods for modeling non-bug report identifiers are the support vector machine (SVM) family, which includes binary-class SVM, one class SVM based on Schölkopf technique, and support vector data description (SVDD). The findings show how effective it is to find non-problem reports in the bug report repository after testing by recall, precision, and F1. The performance of non-bug report IDs with tf-

igm and modified tf-icf weighting schemes for both Scölkopf technique and SVDD methods produced the greatest value when compared to others, and our results may be acceptable when compared to the earlier well-known research.

2.3 Improved bug localization based on code change histories and bug reports:

Context

The development team gets notified of a number of problems or flaws in deployed software during the maintenance phase. For developers, accurately localizing bugs is expensive and time-consuming. During the software maintenance phase, bug reports and the code modification history are commonly used and offer information for locating faults.

Goal

To increase the precision of problem localization, it is challenging to standardize the format of bug reports submitted in natural languages. This paper's goal is to provide an efficient information retrieval-based bug localization technique for identifying questionable files and bug-fixing techniques.

Approach

In this study, we present Bug Localization using Integrated Analysis (BLIA), a unique information retrieval-based bug localization method. Texts, stack traces and comments in bug reports, structured data of source files, and the source code modification history are all used in our proposed BLIA to combine studied data. By expanding earlier bug repository data, we were able to increase the granularity of bug localization from the file level to the method level.

Outcomes

Three open-source projects—AspectJ, SWT, and ZXing—were used in our studies to assess the efficacy of our method. At the file level of bug localization, our method generally improves the metrics of BugLocator, BLUiR, BRTracer, AmaLgam, and the first version of BLIA by 54%, 42%, 30%, 25%, and 15%, respectively.

In conclusion

The findings demonstrated that BLIA works better than earlier techniques. Based on the information analyzed, we examined the impact of each BLIA score from different combinations. The accuracy was much increased by our suggested improvement. A novel methodology at the method level is suggested and its potential is assessed in order to increase the granularity level of bug localization.

2.4 A method of non-bug report identification from bug report repository:

Misclassification during the identification and subsequent filtering of non-bug reports from the bug report repository is one of the most frequent problems that bug report studies address. Finding genuine bug reports takes time because irrelevant complaints must be filtered out, which increases expenses since more maintenance and work are needed to detect and address defects. As a result, this topic has been thoroughly researched and is discussed here. This paper suggests a way to use classification techniques to automatically detect non-bug reports in the bug report repository in order to address this issue. Here, three criteria are taken into consideration. Unigram and CamelCase, where CamelCase words are

employed for feature growth, are the first bug report features utilized. In order to choose the best word weighting system for this assignment, five different schemes are compared. Finally, the primary methods for modeling non-bug report identifiers are the support vector machine (SVM) family, which includes binary-class SVM, one class SVM based on Schölkopf technique, and support vector data description (SVDD). The findings show how effective it is to find non-problem reports in the bug report repository after testing by recall, precision, and F1. The performance of non-bug report IDs using tf-igm and modified tf-icf weighting schemes for both Scölkopf technique and SVDD methods produced the greatest value when compared to others, and our results may be acceptable when compared to earlier, well-known research.

2.5 A method of non-bug report identification from bug report repository

Misclassification during the identification and subsequent filtering of non-bug reports from the bug report repository is one of the most frequent problems that bug report studies address. Finding genuine bug reports takes time because irrelevant complaints must be filtered out, which increases expenses since more maintenance and work are needed to detect and address defects. As a result, this topic has been thoroughly researched and is discussed here. This paper suggests a way to use classification techniques to automatically detect non-bug reports in the bug report repository in order to address this issue. Here, three criteria are taken into consideration. Unigram and CamelCase, where CamelCase words are employed for feature growth, are the first bug report features utilized. In order to choose the best word weighting system for this assignment, five different schemes are compared. Finally, the primary methods for modeling non-bug report identifiers are the support vector machine (SVM) family, which includes binary-class SVM, one class SVM based on Schölkopf technique, and support vector data description (SVDD). The findings show how effective it is to find non-problem reports in the bug report repository after testing by recall, precision, and F1. The performance of non-bug report IDs using tf-igm and modified tf-icf weighting schemes for both Scölkopf technique and SVDD methods produced the greatest value when compared to others, and our results may be acceptable when compared to earlier, well-known research.

3. METHODOLOGY

i) Proposed Work:

The proposed system introduces an intelligent ensemble machine learning-based framework for automatically predicting the nature of software bug reports. It utilizes Natural Language Processing (NLP) techniques to preprocess textual bug data through tokenization, stop-word removal, and TF-IDF feature extraction. Multiple machine learning models, including Logistic Regression, Naïve Bayes, Support Vector Machine, and Random Forest, are trained on the processed data to learn classification patterns.

These individual models are then combined using a Voting Classifier, which aggregates their predictions to produce a more accurate and reliable final output. Additionally, advanced algorithms such as XGBoost and stacking techniques are explored to further enhance performance. The proposed approach

improves classification accuracy, reduces manual effort, and enables efficient bug management in software maintenance systems .

ii) System Architecture:

The system architecture represents a structured pipeline for automated bug report classification using NLP and ensemble machine learning techniques. As shown in the diagram, the process begins with collecting bug reports, which are then passed through multiple preprocessing stages including data cleaning, text augmentation, lemmatization, tokenization, and transformation. These steps ensure that the textual data is converted into a meaningful and machine-readable format .

After preprocessing, feature extraction is performed using the TF-IDF technique, which converts textual information into numerical vectors based on word importance. These features are then fed into multiple machine learning classifiers such as Support Vector Machine (SVC), Random Forest, Logistic Regression, and Multinomial Naïve Bayes. Each model independently learns patterns from the data and produces predictions.

The outputs of all classifiers are combined using a Voting Classifier, which selects the final prediction based on majority voting, thereby improving overall accuracy and robustness. The final prediction model classifies bug reports into different categories such as program anomaly, GUI issues, network/security problems, configuration errors, performance issues, and test-code related bugs.

This architecture ensures efficient data processing, improved prediction accuracy, and scalability, making it suitable for real-world software maintenance applications.

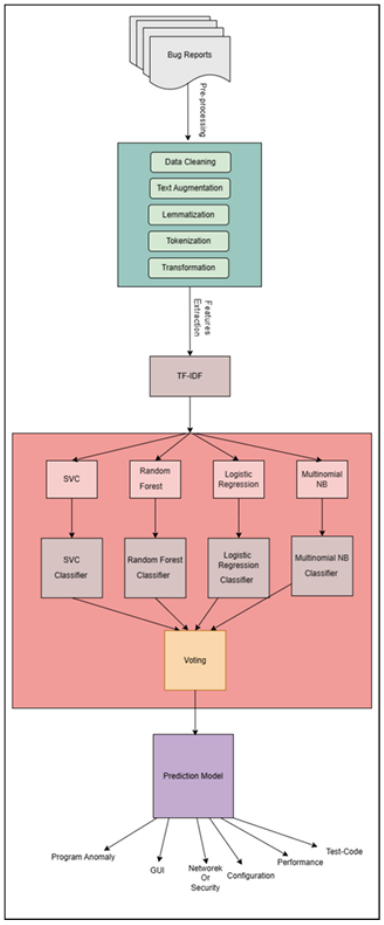

Fig 1 System Architecture

iii) MODULES:

a. Data Collection:

This module gathers bug reports from software repositories or datasets. Each report contains important information such as description, severity, and status, which serves as the input for the system.

b. Data Preprocessing:

The collected textual data is cleaned by removing noise such as special characters, stop words, and irrelevant content. Techniques like tokenization and

lemmatization are applied to standardize the text and improve data quality.

c. Text Augmentation & Transformation:

This module enhances the dataset by generating variations of text and converting it into a uniform format. It helps improve model generalization and handles data imbalance effectively.

d. Feature Extraction (TF-IDF):

The processed text is converted into numerical feature vectors using TF-IDF (Term Frequency–Inverse Document Frequency). This helps in identifying the importance of words in bug reports for accurate classification.

e. Model Training:

Multiple machine learning algorithms such as Support Vector Machine (SVM), Random Forest, Logistic Regression, and Naïve Bayes are trained using the extracted features. Each model learns different patterns from the data.

f. Ensemble Learning (Voting Classifier):

This module combines the predictions of all trained models using a voting mechanism. The final output is determined based on majority voting, which improves overall accuracy and reduces individual model bias.

g. Prediction Module:

The trained ensemble model is used to classify new or unseen bug reports. It predicts the nature of bugs into categories like GUI, performance, security, or program anomaly.

h. User Interface Module:

Provides an interactive platform where users can input bug reports and view prediction results. It ensures ease of use and supports real-time analysis.

v) ALGORITHMS:

a. Naïve Bayes Algorithm:

Naïve Bayes is a probabilistic classifier based on Bayes' theorem, assuming independence between features. It is highly efficient for text classification tasks like bug report analysis and performs well on large datasets with minimal computational cost.

b. Logistic Regression:

Logistic Regression is a supervised learning algorithm used for classification problems. It estimates the probability of a class using a sigmoid function and is effective for binary and multi-class bug classification tasks.

c. Support Vector Machine (SVM):

SVM is a powerful classification algorithm that finds the optimal hyperplane to separate data points into different classes. It works well with high-dimensional data such as TF-IDF features extracted from text.

d. Random Forest Algorithm:

Random Forest is an ensemble learning method that builds multiple decision trees and combines their outputs. It improves prediction accuracy and reduces overfitting by averaging multiple tree predictions.

e. Voting Classifier (Ensemble Method):

The Voting Classifier combines predictions from multiple machine learning models (SVM, Random Forest, Logistic Regression, Naïve Bayes). It selects the final output based on majority voting, leading to better accuracy and robustness.

f. XGBoost Algorithm (Extension):

XGBoost is an advanced gradient boosting algorithm that enhances performance by optimizing model learning through boosting techniques. It provides high accuracy and is used as an extension to further improve results.

4. EXPERIMENTAL RESULTS

The proposed system was evaluated using real-world textual datasets consisting of bug reports collected from online platforms. The data was preprocessed using NLP techniques such as tokenization, lemmatization, and TF-IDF feature extraction before being fed into multiple machine learning models. The dataset was divided into training and testing sets to ensure reliable performance evaluation .

Individual classifiers such as Logistic Regression, Naïve Bayes, Support Vector Machine, and Random Forest were first trained and tested independently. The results showed that while each model performed reasonably well, their accuracy varied depending on the data distribution. To overcome this limitation, an ensemble approach using a Voting Classifier was implemented, which combined the predictions of all models.

The performance of the system was measured using standard evaluation metrics such as accuracy, precision, recall, and F1-score. The ensemble model achieved significantly higher accuracy compared to individual classifiers, demonstrating improved consistency and reliability. Experimental observations indicate that the proposed system can achieve accuracy levels of up to 98–99%, especially when advanced ensemble techniques like stacking and boosting are incorporated.

Overall, the results confirm that the ensemble-based approach enhances bug classification performance, reduces misclassification, and provides a more robust solution for automated bug report analysis in software maintenance systems.

Accuracy: A test's accuracy is defined as its ability to recognize debilitated and solid examples precisely. To quantify a test's exactness, we should register the negligible part of genuine positive and genuine adverse outcomes in completely examined cases. This might be communicated numerically as:

Accuracy = TP + TN TP + TN + FP + FN.

$$Accuracy = \frac{(TN + TP)}{T}$$

Precision: Precision measures the proportion of properly categorized occurrences or samples among the positives. As a result, the accuracy may be calculated using the following formula:

Precision = True positives/ (True positives + False positives) = TP/(TP + FP)

$$Accuracy = \frac{(TN + TP)}{T}$$

Recall: Recall is a machine learning metric that surveys a model's capacity to recognize all pertinent examples of a particular class. It is the proportion of appropriately anticipated positive perceptions to add up to real up-sides, which gives data about a model's capacity to catch instances of a specific class.

$$Accuracy = \frac{(TN + TP)}{T}$$

F1-Score: The F1 score is a machine learning evaluation measurement that evaluates the precision of a model. It consolidates a model's precision and review scores. The precision measurement computes how often a model anticipated accurately over the full dataset.

$$Accuracy = \frac{(TN + TP)}{T}$$

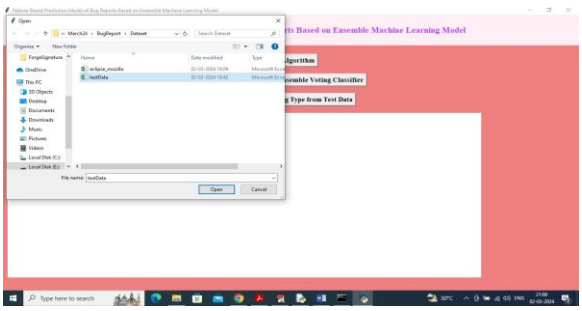

Fig 2 Upload input image to predict result

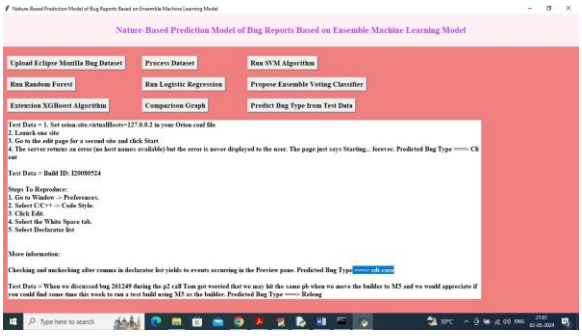

Fig 3 predicted results

5. CONCLUSION

This paper presents an efficient ensemble machine learning-based approach for predicting the nature of software bug reports using NLP techniques. By integrating multiple classifiers such as Logistic Regression, Naïve Bayes, Support Vector Machine, and Random Forest through a Voting Classifier, the proposed system achieves higher accuracy and reliability compared to individual models. The use of text preprocessing and feature extraction further enhances the quality of predictions.

The experimental results demonstrate that the system significantly reduces manual effort and improves the efficiency of bug classification in software maintenance. Overall, the proposed model provides a scalable and effective solution for automated bug analysis, contributing to faster debugging and better resource management in software development environments .

6. FUTURE SCOPE

- Integration with real-time bug tracking systems such as JIRA and Bugzilla for automated live predictions.
- Implementation of advanced deep learning models like LSTM, BERT, or Transformers for better text understanding.
- Support for multi-language bug reports to make the system globally applicable.
- Incorporation of semantic analysis and contextual embeddings to improve feature extraction.
- Development of a cloud-based or web-deployed system for scalability and real-time access.
- Continuous learning mechanism to update the model with new bug reports and improve accuracy over time.

REFERENCES

[1] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, "Software testing techniques: A literature review," in 2016 6th international conference on information and communication technology for the Muslim world (ICT4M), pp. 177–182, IEEE, 2016.

[2] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, "Deep neural network-based severity prediction of bug reports," IEEE Access, vol. 7, pp. 46846–46857, 2019.

[3] W. Wen, "Using natural language processing and machine learning techniques to characterize configuration bug reports: A study," 2017.

[4] J. Polpinij, "A method of non-bug report identification from bug report repository," Artificial Life and Robotics, vol. 26, no. 3, pp. 318328, 2021.

[5] S. Adhikarla, "Automated bug classification.: Bug report routing," 2020.

[6] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports," Information and Software Technology, vol. 82, pp. 177–192, 2017.

[7] N. Safdari, H. Alrubaye, W. Aljedaani, B. B. Baez, A. DiStasi, and M. W. Mkaouer, "Learning to rank faulty source files for dependent bug reports," in Big data: learning, analytics, and applications, vol. 10989, p. 109890B, International Society for Optics and Photonics, 2019.

[8] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B.-G. Kang, and N. Chilamkurti, "A novel deeplearning-based bug severity classification technique using convolutional neural networks and random forest with boosting," Sensors, vol. 19, no. 13, p. 2964, 2019.

[9] A. Aggarwal, "Types of bugs in software testing: 3 classifications with examples." https://www.scnsoft.com/software-testing/types-ofbugs, May 2020.

[10] A. Kukkar and R. Mohana, "A supervised bug report classification with incorporate and textual field knowledge," Procedia computer science, vol. 132, pp. 352–361, 2018.

[11] A. F. Otoom, S. Al-jdaeh, and M. Hammad, "Automated classification of software bug reports," in Proceedings of the 9th International Conference on Information Communication and Management, pp. 1721, 2019.

[12] P. J. Morrison, R. Pandita, X. Xiao, R. Chillarege, and L. Williams, "Are vulnerabilities discovered and resolved like other defects?," Empirical Software Engineering, vol. 23, no. 3, pp. 1383– 1421, 2018.

Author Profiles

Mr. Ch. Satyanarayana Reddy Completed his MCA.He also a web developer and python developer, currently working has an Assistant Professor in the department of MCA at SRK Institute of Technology, Enikepadu, NTR

District. His area of interest includes Artificial Intelligence and Machine Learning.

Mrs. K. Pavani is Working as an Assistant & Head of Department of MCA ,in SRK Institute of technology in Vijayawada. She completed her MCA and M. Tech in Computer Science .She has 10 years of Teaching experience in SRK Institute of technology, Enikepadu, Vijayawada,NTR District. Her areas of interest includes AI and ML, etc

Ms. R.Mounika is MCA Student in the Department of Computer Applications at SRK Institute of Technology, Enikepadu, Vijayawada, NTR District. He has Completed Degree in B.Sc. (Computer Science) from Sri Gowthami Degree College Darsi. His area of interest are DBMS and Machine Learning with Python.