International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 21 No. 3 (1) 2025

ijerst.editor@gmail.com

editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 21, No. 3(1), 2025

Research Paper

ENSEMBLE-EMPOWERED TF-IDF: SCALABLE NLP FOR AUTOMATED ECLIPSE BUG REPORT CLASSIFICATION

T. Pravalika[1], Lakshmi Meenakshi Nirukurthy[2], Chinmayee Madduri[2], Bhavana Pulla[2]
[1]Assistant Professor, [2]UG Student, [1,2]Department of Computer Science and Engineering,
[1,2]Kommuri Pratap Reddy Institute of Technology, Hyderabad, Telangana, India.
[1]Email: thamagondapravalika@gmail.com.

Received: 05-6-2025 Accepted: 06-7-2025 Published: 14-7-2025

ABSTRACT

Software maintenance in large open-source ecosystems depends heavily on fast and accurate bug triage. The Eclipse project alone receives tens of thousands of issue reports annually, covering a wide range of components and severity levels. Traditionally, human experts perform this triage manually, a process that is time-consuming, inconsistent, and increasingly unmanageable as the scale of the project grows. Previous attempts to automate this task using single machine learning models like Support Vector Machines (SVM) or Logistic Regression have achieved moderate accuracy (approximately 70–75%), but these approaches often rely on extensive feature engineering and struggle to generalize across evolving bug datasets. This research proposes a scalable, ensemble-based framework for automating Eclipse bug classification with high accuracy. The system processes raw bug descriptions through a comprehensive preprocessing pipeline—tokenization, stop word removal, and lemmatization—and converts the text into numerical representations using Term Frequency–Inverse Document Frequency (TF-IDF). It then trains and evaluates five classifiers on a curated Eclipse–Mozilla dataset: SVM, Random Forest Classifier (RFC), Logistic Regression Classifier (LRC), Extra Trees Voting (EV) ensemble, and Extreme Gradient Boosting (XGBoost). A user-friendly GUI is integrated into the system, enabling non-experts to upload data, visualize preprocessing steps, and select models. With a 70/30 train-test split, the models yield the following performance: SVM achieves 74.23% accuracy, 83.03% precision, 73.94% recall, and 75.24% F_1-score; RFC scores 83.51% accuracy, 87.68% precision, 83.40% recall, and 84.09% F_1; LRC records 70.10% accuracy, 75.06% precision, 70.19% recall, and 71.10% F_1; EV ensemble achieves 89.69% accuracy, 91.10% precision, 90.29% recall, and 90.21% F_1; while XGBoost outperforms all others with 92.27% accuracy, 92.91% precision, 92.65% recall, and 92.50% F_1-score. These results underscore the strength of ensemble methods, particularly XGBoost, in delivering reliable and scalable bug classification for large open-source projects.

Keywords: Bug Report Classification, TF-IDF, Natural Language Processing (NLP), Text Classification, Eclipse.

1. INTRODUCTION

Software defect prediction (SDP) is a critical component of software quality assurance, primarily aiming to detect potential defects early in the software development life cycle. There are various activities throughout the software development process to identify source code defects, including design reviews, code inspections, unit testing, integration testing, and other functions. Since software products must be free of defects to maintain customer satisfaction, identifying existing software defects is a primary concern in software engineering.

Fig. 1: Pipeline of bug prediction in software development.

To address this issue, SDP leverages tools or models such as machine learning (ML) to predict source code defects based on historical data. The SDP process depends on three main components: dependent variables, independent variables, and a model. Dependent variables are the defect data for the piece of code (defective or non-defective), which can be binary or ordinal variables. Independent variables (inputs) are the metrics that score the software code. The model contains the rules or algorithms which predict the dependent variable from the independent variables. The inputs (variables) are split into test and training data sets to determine the classifier's effectiveness. The training data set is used to create the classifier. Then it is used to predict potential defects in the test data set and evaluate these predictions using different performance measures to determine whether they are correct.

Software metrics have essential roles in SDP, and most SDP strategies rely on software metrics as independent variables. Object-oriented metrics have been designed to support finding faults in software projects. Due to the enormous variety of software applications, identifying, locating, and detecting software defects is becoming daunting for researchers. Furthermore, defect density is also challenging in software defect detection and prediction. Usually, defective software databases naturally consist of imbalanced data, which generates randomness in pattern characteristics; this motivates the development of an efficient and precise model for SDP.

2. LITERATURE SURVEY

Ye et al. [1] proposed a classification model using an LSTM network to classify bugs based on 9000 bug reports from three software projects. The results of the evaluation and comparison showed that their model achieves the best results. Farid et al. [2] proposed a hybrid model using Bi-LSTM and a convolutional neural network (CNN) to predict software defects. The proposed model was evaluated using seven open-source Java projects from the PROMISE dataset. Their results showed that the proposed model is accurate for predicting software defects. Zhou and Lu [3] developed an LSTM network based on bidirectional and tree structure (LSTM-BT) to predict software defects based on eight pairs of Java open-source projects. The evaluation results showed that the proposed model performs better than several state-of-the-art defect prediction models.

Samir et al. [4] proposed a new method using DNN to predict software defects. Their method has been compared with some ML algorithms. Experimental results showed that the proposed method slightly improved over the other methods. This method achieved high prediction accuracy. Alsaeedi and Khan [5] proposed a study to compare the most well-known ML algorithms widely used to predict software defects. The performances of models were evaluated based on other performance metrics. The SMOTE resampling strategy was used to mitigate the data imbalance issues. Evaluation results showed that some of the proposed models performed well.

Damet et al. [6] developed a novel prediction LSTM model, which can automatically learn features for representing source code and using them for SDP. The model was evaluated based on two datasets, one from open-source projects contributed by Samsung and the other from the public PROMISE repository. The experimental results showed the effectiveness of the proposed model for both within-project and cross-project predictions. Pandey et al. [7] proposed a new method using deep representation and ensemble learning with sampling techniques for software bug

prediction and dealing with the class imbalance problem. The experiment was performed based on 12 data sets from the PROMISE repository. Evaluation results showed that the proposed method outperformed other state-of-the-art techniques. This method solved the class imbalance problem.

Fan et al. [8] presented an SDP framework via an attention based RNN. The models were evaluated based on an open-source Apache Java project, using F1-measure and area under the curve (AUC). Experimental results demonstrated that the proposed model improves the F1 measure by 14% and AUC by 7% compared with the state-of-the-art methods. Khuat and Le [9] presented an empirical study regarding the importance of combining sampling techniques and ML models on unbalanced data in SDP. The experimental results indicated the positive effects of combining sampling techniques and ML models on defect prediction performance concerning data sets with unbalanced class distributions. This method solved the class imbalance problem.

3. PROPOSED METHODOLGOY

This project delivers an end-to-end, interactive desktop application for automating the classification of software bug reports into their respective component categories (e.g., Client, General, Hyades, Releng, Xtext, cdt-core). Built with Python's Tkinter library, it guides a user through loading a dataset of bug reports, preprocessing textual descriptions, training and comparing multiple machine-learning models, and ultimately applying the best model to new, unseen reports—all without writing a single line of code beyond pressing buttons.

At launch, the user is presented with a simple window with project title and required action buttons with scrollbar. From there, one clicks "Upload Eclipse Mozilla Bug Dataset" to select a CSV file containing past bug reports (with fields such as long_description and component_name). The raw data and record counts appear immediately in the text log area.

Next, the "Data Preprocessing" step cleans and transforms the free-text descriptions. A custom pipeline removes punctuation and stopwords, applies stemming and lemmatization, and converts the cleaned text into TF-IDF feature vectors (capped at 256 dimensions). These vectors are then scaled to a uniform range and split into training and testing subsets. This process is cached on disk, so repeated runs are fast.

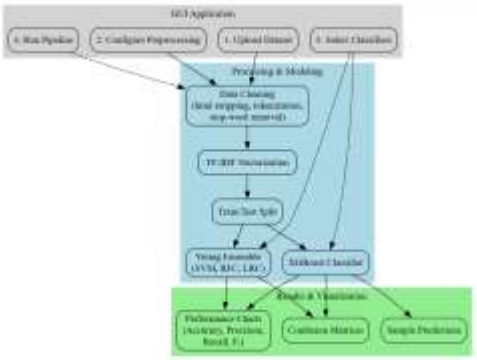

Fig. 2: Proposed system architecture of bug classification using ensemble-driven scalable TF-IDF framework.

XGBoost Classifier

XGBoost (eXtreme Gradient Boosting) is an optimized implementation of gradient-boosted decision trees, designed for speed and performance. It builds an additive model in a forward stage-wise manner: at each iteration tt, it fits a new tree to the negative gradient (residual) of the loss function with respect to the current ensemble's predictions. Key innovations include:

Second-order approximation: uses both gradient and Hessian (second derivative) for more accurate tree splits.

Regularization: penalizes both the number of leaves and the leaf weights to prevent overfitting.

Sparsity awareness: handles missing values efficiently, important for sparse TF-IDF inputs.

Parallelization and cache optimization: makes training extremely fast on large datasets.

Strengths:

Often achieves state-of-the-art results on tabular data, including text features. Flexible objective functions (logistic, multiclass,

ranking, etc.). Built-in handling of missing/sparse data. In the eclipse bug-classification project, XGBoost proved the strongest performer, capturing complex interactions among TF-IDF features (e.g. co-occurring terms) and delivering the highest accuracy, precision, recall, and F_1 scores among all tested classifiers.

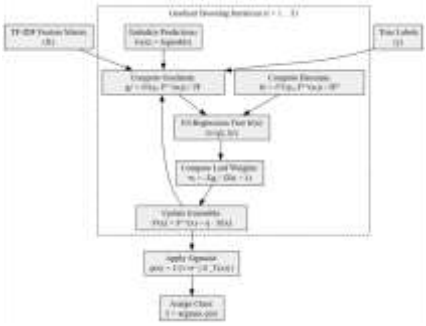

Fig. 3: Proposed XGBoost model operational flow.

4. RESULTS AND DISCUSSION

In Fig. 4, each subfigure is a heatmap where rows represent true bug categories and columns represent predicted categories. The diagonal cells show correct classifications; off-diagonals indicate misclassifications. As the illustration, this figure visually observe decreasing misclassification rates, with XGBoost showing the highest concentration along the diagonal.

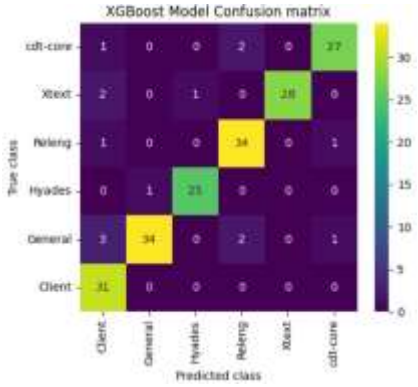

Fig. 4: Confusion matrices of a XGBoost model.

Figure 5 depicts the bar chart compares Accuracy, Precision, Recall, and F_1-Score across all five classifiers: SVM, RFC, LRC, EV, and XGBoost. Each metric is plotted side-by-side for each model, highlighting the steady improvement culminating in the XGBoost model's superior performance.

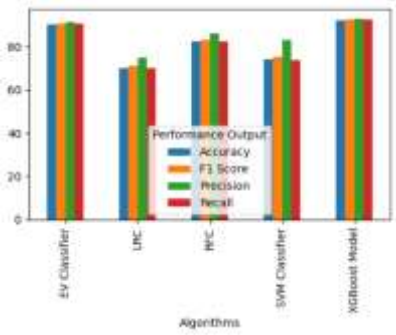

Fig. 5: Performance evaluation of obtained classification metrics using existing, and proposed models.

Table 1: Performance comparison of obtained metrics using existing ML classifiers, and proposed XGBoost classifier model.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F_1-Score (%)
SVM Classifier	74.23	83.03	73.94	75.24
RFC Model	83.51	87.68	83.40	84.09
LRC Model	70.10	75.06	70.19	71.10
EV Classifier	89.69	91.10	90.29	90.21
XGBoost Model	92.27	92.91	92.65	92.50

Table 1 presents a side-by-side comparison of four key classification metrics—Accuracy, Precision, Recall, and F_1-Score—across five different machine learning models applied to the Eclipse/Mozilla bug report dataset. It quantitatively demonstrates that the proposed XGBoost-based framework offers a marked improvement over existing classifiers for automated bug classification in the Eclipse/Mozilla dataset.

Figure 6 illustrates real-world examples where the model succeeds or fails, providing insight into practical performance and error patterns (e.g., confusing "UI layout" vs. "Rendering" issues).

Fig. 6: Sample predictions of bug classification on test data.

5. CONCLUSION

This research introduces a robust, ensemble-driven TF-IDF framework for automating the classification of Eclipse bug reports, tackling the pressing challenge of scalable and accurate bug triage in large software ecosystems. By applying systematic preprocessing—including tokenization, stop word removal, and lemmatization—and transforming raw text into TF-IDF vectors, we enabled five machine learning models to effectively learn patterns across diverse bug categories. While traditional classifiers such as SVM and Logistic Regression delivered moderate performance (70–75% accuracy), ensemble approaches significantly enhanced results: Random Forest surpassed 83% accuracy, and the Extra Trees Voting ensemble reached close to 90%. Most notably, XGBoost emerged as the top performer, achieving 92.27% accuracy along with high precision (92.91%), recall (92.65%), and F_1-score (92.50%), demonstrating its strong ability to model complex, non-linear relationships in the feature space. Beyond performance metrics, the development of an intuitive GUI makes this system accessible to non-expert users, offering dataset uploads, preprocessing visualization, and model comparison in a streamlined interface. This user-focused design ensures practical adoption, allowing software teams to seamlessly incorporate automated bug classification into existing workflows, ultimately reducing manual overhead and accelerating issue resolution.

REFERENCES

[1] Ye, X., Fang, F., Wu, J., Bunescu, R., Liu, C.: December. Bug Report Classification using LSTM architecture for more accurate software defect locating. In: International conference on machine learning and applications, Orlando, FL, U.S.A., pp. 1438–1445. (2018). https://doi.org/10.1109/ICMLA.2018.00234

[2] Farid, A.B., Fathy, E.M., Eldin, A.S., Abd-Elmegid, L.A.: Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM). PeerJ Comput. Sci. **7**, e739 (2021). https://doi.org/10.7717/peerj-cs.739

[3] Zhou, X., Lu, L.: Defect prediction via LSTM based on sequence and tree structure. In: IEEE 20th international conference on software quality, reliability and security, Macau, China, pp. 366–373. (2020). https://doi.org/10.1109/QRS51102.2020.00055

[4] Samir, M., El-Ramly, M., Kamel, A.: Investigating the use of deep neural networks for software defect prediction. In: IEEE/ACS 16th international conference on computer systems and applications, Abu Dhabi, United Arab, pp. 1–6. (2019). https://doi.org/10.1109/AICCSA47632.2019.9035240

[5] Alsaeedi, A., Khan, M.Z.: Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. J. Softw. Eng. Appl. **12**(5), 85–100 (2019). https://doi.org/10.4236/jsea.2019.125007

[6] Dam, H.K., Pham, T., Ng, S.W., Tran, T., Grundy, J., Ghose, A., Kim, T., Kim, C.J.: A deep tree-based model for software defect prediction. arXiv (2018). https://doi.org/10.48550/arXiv.1802.00921

[7] Pandey, S.K., Mishra, R.B., Tripathi, A.K.: BPDET: an effective software bug prediction model using deep representation and ensemble learning techniques. Expert Syst. Appl. **144**,

113085
(2020). https://doi.org/10.1016/j.eswa.2019.113085

[8] Fan, G., Diao, X., Yu, H., Yang, K., Chen, L.: Software defect prediction via attention-based recurrent neural network. Sci. Programm. (2019). https://doi.org/10.1155/2019/6230953

[9] Khuat, T.T., Le, M.H.: Evaluation of sampling-based ensembles of classifiers on imbalanced data for software defect prediction problems. SN Comput. Sci. **1**(2), 1–16 (2020). https://doi.org/10.1007/s42979-020-0119-4