International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

ENHANCING BUG DETECTION IN SOFTWARE PROJECTS USING ML ENSEMBLE TECHNIQUES

A. Yashwanth Reddy[1], G. Rahul[2], S. Purushotham[2], R. Rohith[2], A. Akhil[2]
[1,2]Department of Computer Science and Engineering (Cyber Security), [1,2]Sree Dattha Group of Institutions, Sheriguda, Ibrahimpatnam, 501510, Telangana

Received: 09-07-2025 Accepted: 23-08-2025 Published: 30-08-2025

ABSTRACT

Software maintenance in large open-source ecosystems hinges on rapid, accurate bug triage. The Eclipse project alone accumulates tens of thousands of issue reports annually, spanning multiple components and severity levels. Traditionally, human experts manually classify and route these reports, a process that is labour-intensive, inconsistent, and unscalable as project size grows. Prior attempts to automate classification using single machine-learning models—such as Support Vector Machines (SVM) or Logistic Regression—have yielded moderate accuracy (≈70–75%) but often require extensive feature engineering and fail to generalize across evolving bug corpora. This research addresses the urgent need for a scalable, ensemble-driven framework capable of automating eclipse bug classification with high fidelity. We propose an end-to-end system that ingests raw bug descriptions, applies rigorous data preprocessing (tokenization, stop-word removal, lemmatization), and converts textual data into numerical vectors via Term Frequency–Inverse Document Frequency (TF-IDF). Five classifiers—SVM, Random Forest Classifier (RFC), Logistic Regression Classifier (LRC), Extra-Trees Voting (EV) ensemble, and Extreme Gradient Boosting (XGBoost)—are trained and evaluated on a curated eclipse–mozilla dataset. Our proposed system integrates a user-friendly GUI to guide non-expert users through data upload, preprocessing visualization, and model selection. Under a 70/30 train-test split, performance metrics reveal: SVM achieves 74.23% accuracy, 83.03% precision, 73.94% recall, and 75.24% F_1-score; RFC records 83.51% accuracy, 87.68% precision, 83.40% recall, and 84.09% F_1; LRC attains 70.10% accuracy, 75.06% precision, 70.19% recall, and 71.10% F_1; EV yields 89.69% accuracy, 91.10% precision, 90.29% recall, and 90.21% F_1; while XGBoost outperforms all with 92.27% accuracy, 92.91% precision, 92.65% recall, and 92.50% F_1-score.

Keywords: ML-based bug prediction, ensemble learning, bug classification, Eclipse-Mozilla dataset, TF-IDF, XGBoost, Random Forest

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.

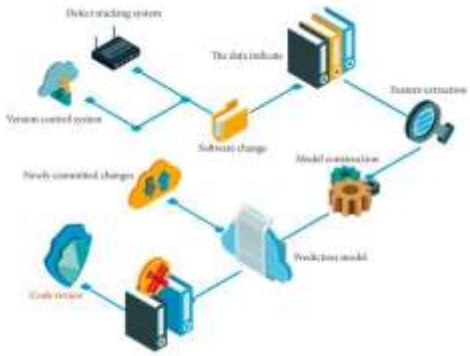


Fig. 1: Pipeline of bug prediction in software development.

To address this issue, SDP leverages tools or models such as machine learning (ML) to predict source code defects based on historical

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

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 .

2.LITERATURE SURVEY

The prediction of defects in software systems is significant, and there is great interest in developing novel high-performance software defect predictors. The purpose of SDP models is to improve the quality of software. Many models have been constructed to recognize the defects in software modules using artificial intelligence and statistical methods. RNN, Support Vector Machine, ANNs, K-Nearest Neighbors (KNN)and Deep Neural Networks (DNN) are some of the algorithms used for SDP.

Ayon [2] proposed a method based on different neural network models. The models were evaluated based on five different datasets from NASA using different scales. The experimental results showed that the proposed method is suitable for predicting software defects. This method used different performance measures and achieved high prediction accuracy. Kumar and Satyanarayana [6] developed a Hybrid Neural Network model with object-oriented and CK. metrics for software fault prediction. Adaptive Genetic Algorithm has been used for ANN optimization. The proposed model has been tested with PROMISE data sets. The experimental results showed better performance compared to major existing schemes.

Miholca et al. [9] presented a supervised classification approach named (HyGRAR). It is a nonlinear hybrid model that combines gradual relational association rule mining and ANNs to predict software defects. Experiments were conducted using ten open-source datasets; their results showed excellent performance of the proposed classifier and better performance than most previously proposed classifiers. Their method achieved high prediction accuracy. Khleel and Nehéz [10] presented a model based on a convolutional neural network (CNN) and gated recurrent unit (GRU) combined with a synthetic minority oversampling technique plus the Tomek link (SMOTE Tomek) to predict software defects. The historical data obtained from the PROMISE repository were used to evaluate the experiments. The experimental results have been compared and evaluated using several performance measures. The experimental results demonstrate that the proposed models perform better and that there are positive effects of combining CNN and GRU models with the SMOTE Tomek method on the performance of SDP regarding datasets with imbalanced class distributions, and the proposed approach is a more promising alternative for addressing the problem of class imbalance in SDP as compared with previous methods.

Arar and Ayan [13] proposed a hybrid classifier to predict software defect problems. The performance of the proposed classifier was compared with other algorithms based on five datasets, and the results show its performance is better. The method used different performance measures and achieved high prediction accuracy. Deng et al. [15] proposed a novel LSTM method to perform SDP; their method can automatically learn semantic and contextual information from the program's ASTs. The experiment was performed on several open-source projects,

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

showing that the proposed LSTM method is superior to the state-of-the-art methods.

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. Each "Build & Train" button fits its respective model to the training set and then evaluates on the hold-out test set, printing accuracy, precision, recall, and F1-score. A confusion-matrix heatmap visually highlights where each model misclassifies components.

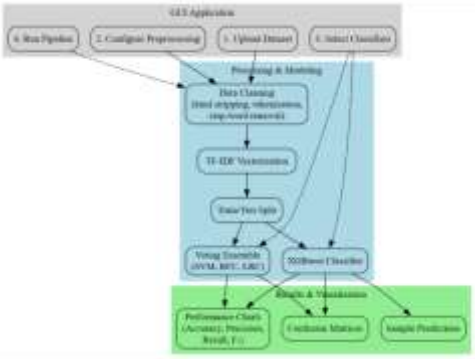


Fig. 2: Proposed system architecture of bug classification using ensemble-driven scalable TF-IDF framework.

To compare all models at a glance, the "Performance Evaluation" button aggregates their metrics into a bar chart, making it easy to see which algorithm performs best on your data. In practice, this helps a team decide whether a simple logistic-regression baseline suffices or whether the extra complexity of XGBoost yields a worthwhile improvement.

Finally, with a trained XGBoost model saved in memory, the "Predict Bug Type from Test Data" feature allows the user to load any new CSV of bug reports and instantly receive predicted component categories. Each description is cleaned, vectorized, scaled, and passed through the XGBoost model, with results streamed into the text log.

By encapsulating data loading, NLP-based preprocessing, multi-model training, evaluation, visualization, and live prediction into a cohesive GUI, this application democratizes advanced bug-classification workflows—enabling developers, QA engineers, and project managers to harness machine learning for faster, more consistent bug triage without deep data-science expertise.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

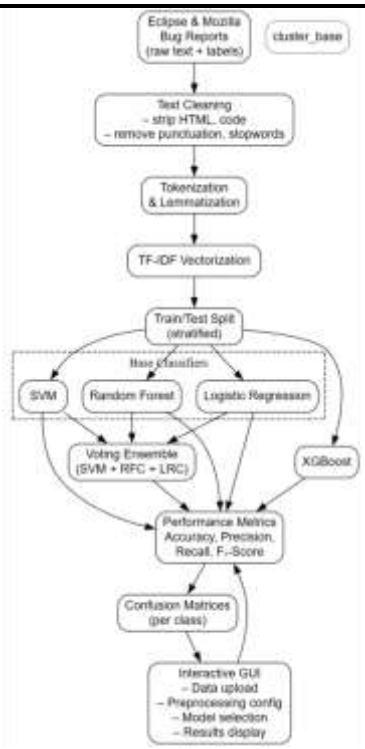


Fig. 3: Proposed workflow of bug classification using ensemble-driven scalable TF-IDF framework.

4.2 Data Preprocessing

In our Eclipse bug classification pipeline, data preprocessing plays a crucial role in transforming raw bug reports—comprising fields like titles, descriptions, and stack traces—into structured numerical feature vectors suitable for machine learning models. The process begins with raw data ingestion, where relevant textual fields are concatenated into a single document per report. HTML tags, code fragments, and markup are stripped using regex or libraries like BeautifulSoup. Text is then converted to lowercase, and punctuation is removed to ensure uniformity. Tokenization splits the cleaned text into individual words, with optional handling for programming identifiers. Common stop words (e.g., "the", "and") and domain-specific terms (e.g., "eclipse", "bug") are filtered out to reduce noise. Stemming or lemmatization further reduces word variants to a common root, minimizing vocabulary sparsity. Rare words occurring in fewer than a set number of documents are pruned, and the remaining terms are transformed into term frequency (TF) vectors. These are then converted into TF-IDF scores to down-weight common terms and highlight more informative ones. Feature normalization ensures equal weight across documents, and the final TF-IDF matrix is split into training and testing sets (e.g., 80/20), preserving class distribution. This comprehensive pipeline ensures clean, consistent, and information-rich input for classifiers like SVM, Random Forest, Logistic Regression, Ensemble Voting, and XGBoost.

4.3 TF-IDF Feature Extraction

TF-IDF (Term Frequency–Inverse Document Frequency) is a fundamental technique in text mining and information retrieval that transforms raw text into numerical feature vectors by quantifying the importance of each word within a document relative to the entire corpus. Term Frequency (TF) captures how often a term t appears in a document d, with variations such as raw count, logarithmic normalization, and augmented frequency to mitigate bias from longer documents. Document Frequency (DF) measures how many documents contain a term, while Inverse Document Frequency (IDF) down-weights terms that are common across many documents, typically using the formula $IDF(t) = \log(N/DF(t))$, where N is the total number of documents, and variants like $\log(1 + N/(1 + DF(t)))$ are used to avoid division by zero. The final TF-IDF score is the product of TF and IDF, giving higher weights to terms that are frequent in a specific document but rare in the corpus, thereby improving the discriminative power of textual features for tasks like classification and clustering.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

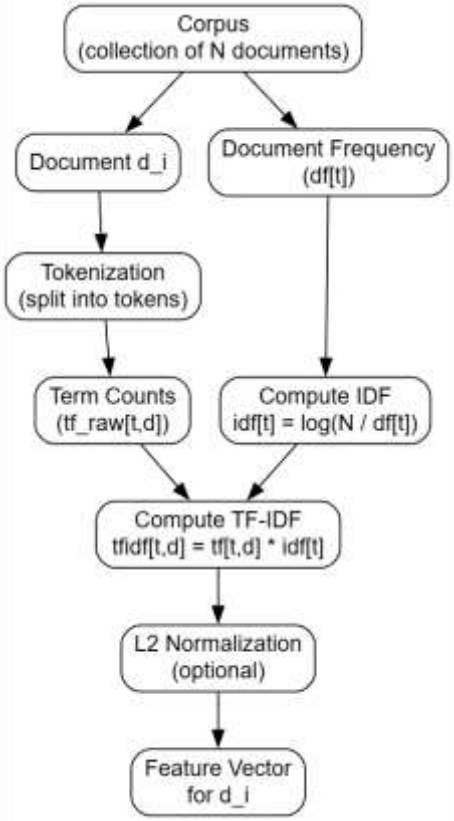


Fig. 4: Internal operation of feature extraction using TF-IDF.

4.4 Model Building and Training

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.

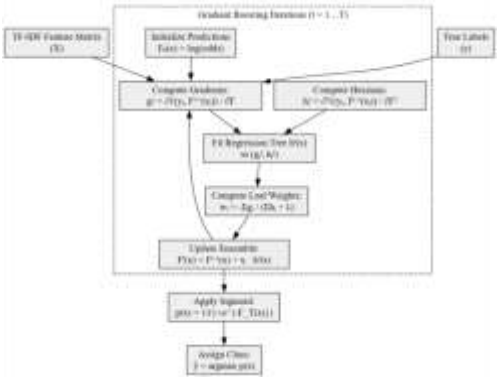


Fig. 5: Proposed XGBoost model operational flow.

Strengths:

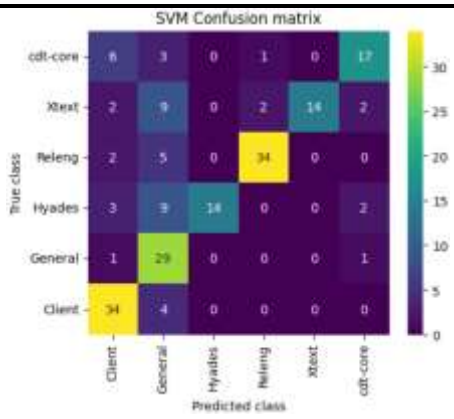
- Often achieves state-of-the-art results on tabular data, including text features.
- Flexible objective functions (logistic, multiclass, ranking, etc.).
- Built-in handling of missing/sparse data.

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.

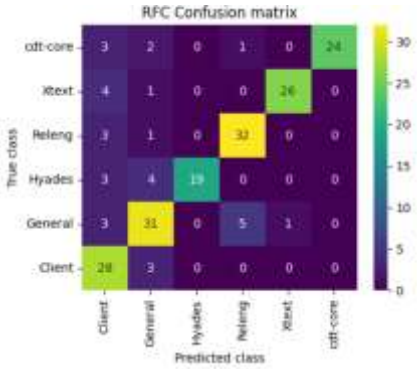
4. Results

In Fig. 6, 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 move from (a) to (e), this figure visually observe decreasing misclassification rates, with XGBoost (e) showing the highest concentration along the diagonal.

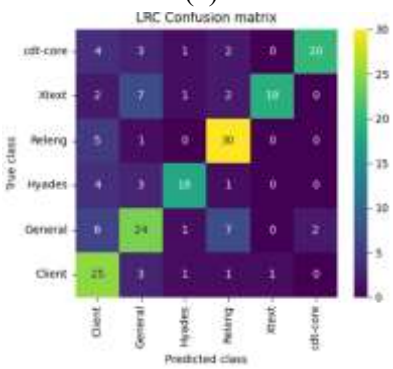
International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper



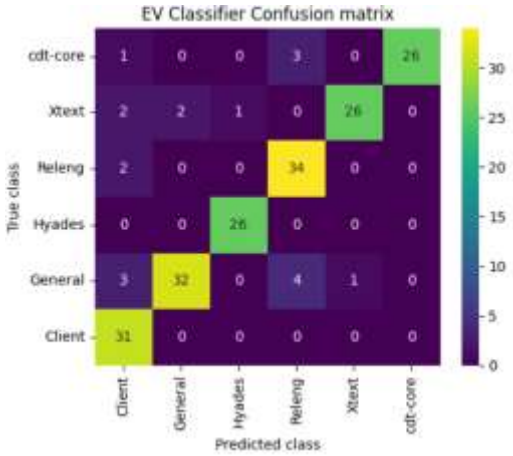
(a)



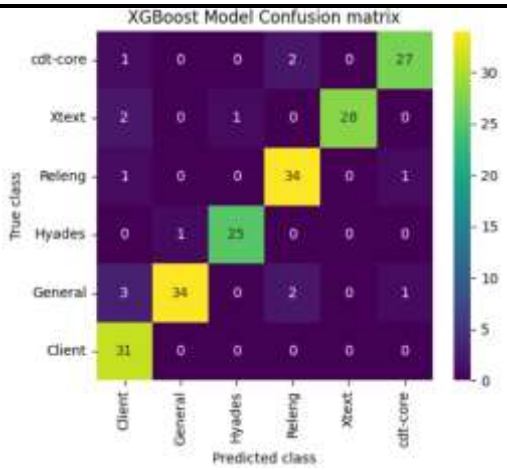
(b)



(c)



(d)



(e)

Fig. 6: Confusion matrices obtained using (a) SVM classifier. (b) RFC model. (c) LRC model. (d) EV classifier. (e) XGBoost model.

Figure 7 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.

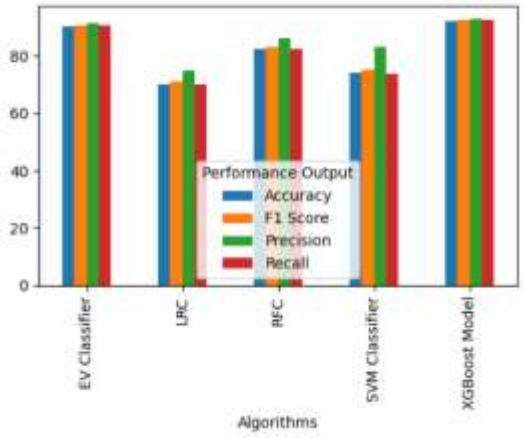


Fig. 8: Performance evaluation of obtained classification metrics using existing, and proposed models.

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

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F_1-Score (%)
SVM Classifi	74.23	83.03	73.94	75.24

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
ISSN: 3068-272X www.ijdim.com Original Research Paper

er				
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.

The performance evaluation of five classifiers on the Eclipse bug classification task reveals that ensemble-based and tree-based models outperform traditional linear models. The XGBoost model achieved the highest overall performance with an accuracy of 92.27%, precision of 92.91%, recall of 92.65%, and an F_1 score of 92.50%, indicating strong consistency in both detecting relevant bugs and minimizing false positives. The Extra Trees Voting (EV) classifier followed closely with 89.69% accuracy and an F_1 score of 90.21%, showcasing the effectiveness of ensemble strategies. The Random Forest Classifier (RFC) also demonstrated solid results with 83.51% accuracy and an 84.09% F_1 score. In contrast, the SVM classifier, while achieving a high precision of 83.03%, lagged in recall (73.94%) and F_1 score (75.24%), and the Logistic Regression Classifier (LRC) had the lowest performance across all metrics, with 70.10% accuracy and a 71.10% F_1 score. These results highlight the superiority of ensemble and gradient boosting techniques for handling complex, high-dimensional text classification problems like bug triage.

Figure 9 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).

(a)

(b)

Fig. 10: Sample predictions of bug classification on test data.

5. CONCLUSION

This research presents a comprehensive, ensemble-driven TF-IDF framework for automated classification of Eclipse bug reports, addressing the critical challenge of scalable, accurate bug triage in large software ecosystems. By systematically preprocessing raw textual data—through tokenization, stop-word removal, and lemmatization—and converting it into TF-IDF feature vectors, we enabled a suite of five machine-learning models to learn discriminative patterns across bug categories. Our evaluation demonstrates that while traditional classifiers like SVM and Logistic Regression offer moderate performance (70–75% accuracy), ensemble methods substantially improve results: Random Forest achieves over 83% accuracy, and the Extra-Trees Voting ensemble reaches nearly 90%. Most notably, the XGBoost model attains a leading 92.27% accuracy, paired with high precision (92.91%), recall (92.65%), and F_1-score (92.50%), underscoring its superior capability to capture complex, non-linear

relationships in the TF-IDF feature space. Beyond raw performance, our proposed GUI application simplifies the deployment and adoption of this pipeline, allowing users without machine-learning expertise to upload datasets, visualize preprocessing steps, and compare model metrics. This user-centric design ensures that software teams can integrate automated classification into existing workflows, reducing manual effort and expediting bug resolution.

REFERENCES

[1] Li, Z., Jing, X.Y., Zhu, X.: Progress on approaches to software defect prediction. IET Softw. **12**(3), 161–175 (2018). https://doi.org/10.1049/iet-sen.2017.0148

[2] Ayon, S.I.: Neural network based software defect prediction using genetic algorithm and particle swarm optimization. In: 1st International conference on advances in science, engineering and robotics technology, Dhaka, Bangladesh. (2019). https://doi.org/10.1109/ICASERT.2019.8934642

[3] Mustaqeem, M., Saqib, M.: Principal component based support vector machine (PC-SVM): a hybrid technique for software defect detection. Clust. Comput **24**, 2581–2595 (2021). https://doi.org/10.1007/s10586-021-03282-8

[4] Tong, H., Liu, B., Wang, S.: Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Inf. Softw. Technol. **96**, 94–111 (2018). https://doi.org/10.1016/j.infsof.2017.11.008

[5] Pan, C., Lu, M., Xu, B., et al.: An improved CNN model for within-project software defect prediction. Appl. Sci. **9**(10), 2138 (2019). https://doi.org/10.3390/app9102138

[6] Kumar, R.S., Sathyanarayana, B.: Adaptive genetic algorithm based artificial neural network for software defect prediction. Glob. J. Comput. Sci. Technol. **15**(1), 23–32 (2015)

[7] Manjula, C., Florence, L.: Deep neural network based hybrid approach for software defect prediction using software metrics. Clust. Comput. **22**(4), 9847–9863 (2019). https://doi.org/10.1007/s10586-018-1696-z

[8] Anbu, M., Anandha Mala, G.S.: Feature selection using firefly algorithm in software defect prediction. Clust. Comput. **22**(5), 10925–10934 (2019). https://doi.org/10.1007/s10586-017-1235-3

[9] Miholca, D.L., Czibula, G., Czibula, I.G.: A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks. Inf. Sci. **441**, 152–170 (2018). https://doi.org/10.1016/j.ins.2018.02.027

[10] Khleel, N.A.A., Nehéz, K.: A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method. J. Intell. Inf. Syst. **60**(3), 673–707 (2023). https://doi.org/10.1007/s10844-023-00793-1

[11] Akour, M., Melhem, W.Y.: Software defect prediction using genetic programming and neural networks. Int. J. Open Sour. Softw. Process. **8**(4), 32–51 (2017). https://doi.org/10.4018/IJOSSP.2017100102

[12] Khleel, N.A.A., Nehéz, K.: A new approach to software defect prediction based on convolutional neural network and bidirectional long short-term memory. Prod. Syst.

Inf. Eng. **10**(3), 1–18 (2022). https://doi.org/10.32968/psaie.2022.3.1

[13] Arar, Ö.F., Ayan, K.: Software defect prediction using cost-sensitive neural network. Appl. Soft Comput. **33**, 263–277 (2015). https://doi.org/10.1016/j.asoc.2015.04.045

[14] Jayanthi, R., Florence, L.: Software defect prediction techniques using metrics based on neural network classifier. Clust. Comput. **22**(1), 77–88 (2019). https://doi.org/10.1007/s10586-018-1730-1

[15] Deng, J., Lu, L., Qiu, S.: Software defect prediction via LSTM. IET Softw. **14**(4), 443–450 (2020). https://doi.org/10.1049/iet-sen.2019.0149