A Hybrid Machine Learning Framework Using Random Forest and XGBoost for Software Bug Prediction

[1]P.C. SANDHYA, [2]DR.I.NASRULLA
[1]*PG student, Vemu Institute of Technology, P. Kothakota*
[2]*Professor, Vemu Institute of Technology, P. Kothakota*

Abstract: **Code smells, indicating poor design or implementation choices, can harm software maintainability and increase bug-proneness. This study explores the significance of code smell metrics in prediction models for detecting bug-prone code modules. By incorporating smell-based metrics, we aim to enhance bug prediction accuracy. Using 14 open-source projects from the PROMISE repository, all written in Java, we trained models with metrics like F1-score, accuracy, precision, and recall. Classifiers like Naïve Bayes, Random Forest (RF), Support Vector Machine (SVM), Logistic Regression, and k-Nearest Neighbor were applied. RF and SVM outperformed the other methods, delivering higher accuracy both within versions and across projects, proving their effectiveness in predicting buggy components.**

Keywords: **Code smell, source code, smell-aware, bugs classification.**

INTRODUCTION

Software systems play a critical role in today's digital world, driving nearly every aspect of daily life. From economics, transportation, and healthcare to communication and entertainment, software applications are integral. Given this reliance, ensuring software functions accurately and remains as bug-free as possible is crucial. A software bug, which is a flaw in the code leading to incorrect results, can significantly impact software quality and user experience. Detecting and fixing bugs is a time-consuming but essential task in software development. While it is impossible to create entirely bug-free software, predicting and addressing bugs early in the development process can improve performance, quality, and user satisfaction. Automated bug detection, using binary classification, helps identify buggy modules before release, enhancing the overall software development life cycle. This proactive approach is vital to delivering robust, reliable applications.

LITERATURE SURVEY

1. Antipattern Metrics for Bug Prediction (Taba et al., 2019)

Taba et al. developed one of the earliest bug classification models incorporating antipattern metrics. Antipatterns are recurring design flaws that negatively impact software quality. By analyzing PROMISE repository datasets, their study demonstrated that antipattern detection could significantly enhance bug prediction accuracy. However, their approach was limited to structural metrics and did not account for process-related factors affecting software bugs.

2. Program Slicing Metrics for Bug Detection (Pan et al., 2014)

Pan et al. introduced program slicing metrics to predict software bugs in Apache HTTP and Latex2rtf projects. Program slicing involves analyzing dependencies and control flows within software to identify critical bug-prone regions. Their study found that slicing metrics provide a more detailed understanding of program behavior compared to traditional complexity measures. However, the technique faced scalability challenges when applied to large-scale software systems.

3. Code Coverage and Bug Classification (Varshneya, 2018)

Varshneya's research focused on using code coverage as a predictive feature for bug classification. The study analyzed software bug reports and found that code coverage metrics alone were insufficient for accurate bug prediction. While code coverage helps in identifying untested areas, it does not account for the quality of executed code. This limitation prompted further research into integrating additional metrics, such as code smells.

4. Design Flaws and Antipatterns (Khomh et al., 2012)

Khomh et al. conducted a study demonstrating that classes exhibiting design flaws (antipatterns) were more likely to contain future bugs. Their research emphasized the importance of identifying code smells early in the software development process.

Using a dataset based on code smell metrics, they showed that certain antipatterns, such as large classes and cyclic dependencies, were strong predictors of software defects. However, implementing their model in dynamic environments posed significant challenges.

5. Severity Index for Code Smells (Palomba et al., 2016)

Palomba et al. introduced the severity index for code smells, which quantifies the impact of design flaws on software maintainability. Their study on Java open-source projects demonstrated that severity indexing could refine bug classification accuracy. By combining source code metrics with smell severity, their approach improved the distinction between major and minor defects. However, the computation of severity indices required extensive processing, limiting its real-time applicability.

PROBLEM STATEMENT

Code smells negatively impact software maintainability and bug-proneness. Existing bug prediction models lack sufficient incorporation of these metrics. This project seeks to enhance the accuracy of bug prediction by integrating code smell metrics into predictive models, particularly in open-source Java projects.

PROPOSED METHOD

In propose paper author employing machine learning algorithms to combat against buggy software's. Propose work utilizing source code metrics and smell code metrics from 14 different open source projects such as ANT, XALAN, XERCES and many more.

Source code metrics includes bugs related to Inheritance coupling, coding complexity, public methods and many more. Smell code metrics refers to 'Number of anti-patterns which includes number of bugs in source code', complexity metrics, recurrence length and cumulative pair wise difference. Complete details about this metrics can be read in TABLE2 in base paper.

All existing algorithms were utilizing only coding metrics to detect bugs but propose paper utilizing both Source Code Metrics and Smell code metrics. Both metrics will be combined and then split into various number of TRAIN AND TEST split and then trained with different ML algorithms like Random Forest, SVM, KNN, Logistic Regression and Naïve Bayes. Each algorithm performance is evaluated in terms of accuracy, precision, recall, confusion matrix, AUC-ROC graph and FSCORE. Among all algorithms Random Forest and SVM is giving best accuracy.

To get best results author employing CORRELATION based Features selection algorithm which will identify and remove highly co-related or similar values and then select unique features values. Dataset having imbalance issue where BUGGY class contains only 5000 records and NON_BUGGY class contains 10000 records so to balance both classes with equal number of records author has used SMOTE algorithm to generate synthetic instances of BUGGY instances.

METHODOLOGY

The research community has presented many bugs prediction [1], [2], [3], [4] and classification [5], [6] models based on various indicators to recognize more error-prone modules in software applications. Few of them [7] have enhanced accuracy and evaluation metrics as compared to others. However, only a few authors [8] did bug classification but their model is not smell-aware. This study used different approaches to do smell-aware bug classification through ML algorithms. Furthermore, we will do the result analysis of the algorithms with each other and compare their accuracy using dissimilar source code and smell-based metrics.

We propose five ML models: LR, RF, SVM, NB, and k-NN, to detect and classify smell-aware bugs. Our objective in these proposed models is to achieve high accuracy. Multiple stages have been conducted to address the challenges in Machine Learning, resulting in significant success in achieving the highest possible accuracy for smell-aware bug classification. However, we aim to investigate the reasons behind the lower accuracy of the ML models and compare the results and performance of LR, SVM, RF, k-NN, and NB. Consequently, we will analyze which machine learning approach is best for smell-aware bug detection and classification. For our study, we proceeded with the datasetfrom Jureczko et al. [43], which is accessible from the PROMISE repository [44]. This dataset comprises a rich collection of 44 releases from 14 projects, each with 20 code metrics. Additionally, the occurrence of bugs in each release is readily available. It is worth noting that the dataset includes systems of various sizes and scopes, allowing us to enhance the validity of our investigation [45]. Furthermore, we considered the

findings of Mende et al. [46], who discovered that models trained on limited datasets can yield unreliable performance estimations.

A. INPUT

The first step in our methodology is the input stage, where we gather data from different open-source projects containing bugs. These projects are obtained from the PROMISE bug repository and serve as the training data for our model. The details of the software projects dataset used in this study can be found in 3.3. A comprehensive description of the dataset is provided, including specific information about each project. For further reference, please consult Table 1, which presents the specific details of the dataset used in our study

B. PROPOSED MACHINE LEARNING TECHNIQUES

In this study, we developed a smell-aware bug prediction model using various machine learning approaches. Our chosen learning style is supervised learning, which meanswe focus on algorithms that support this type of learning. The prediction outputs of our model are classified into two types: classification and regression. Classification involves linking input variables to discrete output values, while regression predictive analysis maps input factors to continuous output variables. In the case of our bug prediction model, the output type is binary, meaning we categorize a source code segment as either buggy or non-buggy. Consequently, we will only explore methods that support binary classification, as this paper specifically focuses on the binary classification of bugs. For our investigation, we selected five commonly used classifiers in bug prediction research: LR classifier, RF classifier, SVM classifier, k-NN, and NB classifier. These classifiers will be utilized in our study to develop and evaluate the performance of the smell-aware bug prediction model.

RESULTS

CONCLUSION

Software defects are called bugs in a software development process - unanticipated deeds and actions figured out by quality control engineers during application testing and are preserved as software bugs. Bugs have high effects on software quality. The process of bug fixing is exceptionally steady and time-consuming. Therefore, it is crucial to detect bugs automatically. The primary goal of this investigation is to create a smell-aware bug prediction model by using code smell as a nominee metric. To be smell-aware, we added an intensity index to the dataset. The results showed that using the intensity index as a predictor for bug prediction improves the accuracy of the bug prediction model. Furthermore, the data show that the severity index is more significant than any other quality metric in predicting the bug-proneness of the smelly classes. The findings suggest that using the severity index as a reliable indicator of buggy modules improves the effectiveness of structurally based baseline models for bug prediction. Furthermore, they also emphasize the significance of the intensity of code smells in the process metrics-based prediction approaches.'' We provided empirical evidence in this study that code smell-based metrics are quite useful in bug prediction. Using several source code metrics and code smell-based metrics proposed in the literature, we constructed a bug prediction model. To create the model, we employed k-NN, NB, RF, SVM, and LR algorithms. Multiple versions of fourteen different open-source projects were used to train the bug prediction model. We experimented with how our bug prediction model behaved within the version, within the project, and across the projects.

REFERENCES

[1] G. M. Ubayawardana and D. D. Karunaratna, ‘‘Bug prediction model using code smells,’’ in Proc. 18th Int. Conf. Adv. ICT for Emerg. Regions (ICTer), Sep. 2018, pp. 70–77.

[2] N. K. Nagwani and P. Singh, ‘‘Bug mining model based on eventcomponent similarity to discover similar and duplicate GUI bugs,’’ in Proc. IEEE Int. Advance Comput. Conf., Mar. 2009, pp. 1388–1392.

[3] N. K. Nagwani and S. Verma, ‘‘Predictive data mining model for software bug estimation using average weighted similarity,’’ in Proc. IEEE 2nd

Int. Advance Comput. Conf. (IACC), Feb. 2010, pp. 373–378.

[4] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, ‘‘Fuzzy setbased automatic bug triaging: NIER track,’’ in Proc. 33rd Int. Conf. Softw. Eng. (ICSE), May 2011, pp. 884–887.

[5] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, ‘‘Improving automated bug triaging with specialized topic model,’’ IEEE Trans. Softw. Eng., vol. 43, no. 3, pp. 272–297, Mar. 2017.

[6] R. Varshneya, ‘‘There’s no such thing as a bug-free app,’’ Entrepreneur, vol. 22, Oct. 2015.

[7] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, ‘‘Software bug prediction using machine learning approach,’’ Int. J. Res. Appl. Sci. Eng. Technol., vol. 11, no. 12, pp. 1401–1404, Dec. 2023.