International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 04 | April - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Software Bug Prediction Using Machine Learning

Mr. S. Anil Kumar[1], Bhimavarapu Siva Prasanna[2],

Arigela Grace Mani[3] , Gorantla Deepika[4], Bhavanam Venkata Ajay Kumar Reddy[5]

[1]Associate Professor, Department of Computer Science and Engineering, Tirumala Engineering College
[2,3,4,5]Student, Department of Computer Science and Engineering, Tirumala Engineering College

---

Abstract - Software bug prediction is a crucial aspect of software development and maintenance as it directly impacts the overall success of the software. By identifying potential bugs early on, software quality, reliability, and efficiency can be improved while also helping to reduce costs. Creating a reliable bug prediction model is a complex task, with various techniques proposed in the literature. In this paper, a bug prediction model utilizing machine learning algorithms is introduced. Three supervised machine learning algorithms have been incorporated to forecast future software faults using historical data. The evaluation process revealed that Machine Learning algorithms, specifically Naïve Bayes (NB), Decision Tree (DT), and Artificial Neural Networks (ANNs), can be effectively utilized with a high level of accuracy. Additionally, a comparison measure was implemented to assess the proposed prediction model against other methods. The findings indicated that the Machine Learning approach outperformed other techniques in terms of performance.

Key Words: Bug Prediction, Reliability, Efficiency, Quality, Machine Learning, Performance Metrics

1.INTRODUCTION

Software bugs have a significant impact on the reliability, quality, and cost of maintaining software. Despite careful development, it is difficult to create bug-free software due to hidden bugs. Developing a software bug prediction model to identify faulty modules early on is a major challenge in the field of software engineering. Predicting software bugs is crucial in software development as it allows for identifying problematic modules before deployment, leading to higher user satisfaction and improved software performance. However, building a reliable bug prediction model is a challenging task, and various techniques have been suggested in the research literature. In this study, a software bug prediction model based on machine learning (ML) is presented. By analyzing historical data, three supervised ML algorithms—Naive Bayes (NB), Decision Trees (DT), and Convolutional Neural Networks (CNN)—were used to forecast future software faults.

Many methods have been suggested for addressing the issue of software bug prediction (SBP). The most commonly used methods are those based on machine learning (ML). ML techniques are widely employed in SBP for forecasting problematic modules by analyzing past fault records, key metrics, and various software computational methods.

In this research, we explored the effectiveness of three supervised machine learning classifiers in the context of SBP. Specifically, we looked at the Naïve Bayes (NB) classifier, the Decision Tree (DT) classifier, and the Artificial Neural Networks (ANNs) classifier. These classifiers were tested on three distinct datasets from previous works [1] and [2]. Furthermore, we conducted a comparison between these classifiers, analyzing their performance based on various evaluation metrics, including accuracy, precision, recall, F-measures, and ROC curves.

2. LITERATURE SURVEY

Numerous studies have been conducted on predicting software bugs using machine learning methods. For instance, a study cited introduced a linear Auto-Regression (AR) technique for forecasting faulty modules. This study relied on past software fault data to anticipate future issues. The research also assessed and contrasted the AR model with the known power model (POWM), utilizing the root Mean Square Error (RMSE) metric. Furthermore, three datasets were employed in the evaluation, yielding promising outcomes.

In recent research papers, the effectiveness of different machine learning techniques for predicting faults has been examined. Sharma and Chandra also summarized key findings from previous studies on each machine learning method and current developments in using machine learning for predicting software bugs. This study serves as a foundational starting point for future research in software bug prediction.

R. Malhotra conducted a thorough systematic review of software bug prediction techniques, utilizing Machine Learning (ML). The study covered research from 1991 to 2013, examined ML methods for bug prediction models, evaluated their effectiveness, compared ML with statistical methods, compared various ML techniques, and highlighted the strengths and weaknesses of ML approaches. The paper sets a standard for easy comparison among different bug prediction methods. The research conducted a detailed analysis comparing various bug prediction methods, introducing a new approach, and assessing its effectiveness through a thorough comparison with other approaches using a standardized benchmark. D. L. Gupta and K. Saxena [7] created a model for predicting software bugs in an object-oriented context, combining relevant defect datasets from the Promise Software Engineering Repository.

The model was evaluated based on accuracy metrics, with results showing an average accuracy of 76.27% for the proposed approach.

Additionally, research was conducted and explored using different machine learning methods, which highlighted their role in predicting software defects. These studies supported developers in utilizing valuable software metrics and selecting appropriate data mining techniques to improve software quality. Specifically, the study identified key metrics for effective defect prediction, including response for class (ROC), line of code (LOC), and lack of coding quality (LOCQ).

However, many other studies have delved into various machine learning techniques and datasets. Previous research has primarily concentrated on the optimization of SBP metrics for efficiency, while other studies have put forth alternative methods for predicting software bugs instead of relying solely on ML techniques.

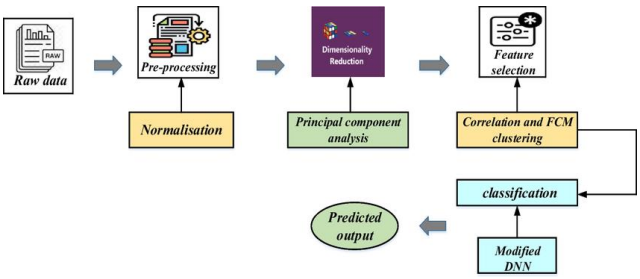
3. PROPOSED SYSTEM

The aim of the model is to predict software bugs using machine learning algorithms. By analyzing past data, the model can anticipate future software issues, leading to enhancements in software quality, reliability, efficiency, and cost reduction. The model incorporates Naïve Bayes, Decision Tree, Random Forest, and Convolutional Neural Networks as supervised machine learning classifiers. Evaluation results indicate that the system achieves high accuracy rates and is effective in forecasting software faults.

The new software bug prediction system that uses machine learning algorithms offers many benefits. It can help improve software quality by identifying and fixing bugs before deployment, leading to a higher quality software product that meets user needs. Discovering and resolving software bugs is a time-consuming and costly process. By anticipating bugs and dealing with them at an early stage in the development process, the suggested system can aid in cutting down the overall expenses of software development and maintenance. Thanks to the suggested system, software developers can detect and solve bugs sooner in the development cycle, leading to a quicker development process and reducing time to market.

4. METHODOLOGY

Software bug prediction involves using a variety of machine learning techniques, including Support Vector Classification (SVC), Random Forest (RF), Nu-Support Vector Classification (NuSVC), and Multi-Layer Perceptron (MLP). The first step is to gather three testing/debugging datasets from reliable sources to cover a range of software fault scenarios. Data pre-processing includes normalizing numerical features and addressing class imbalances to improve model training. Techniques are used to select important software metrics that help improve the accuracy of predictive models. Three machine learning classifiers (SVC, RF, and NuSVC) are then trained on processed and balanced datasets with historical fault data and essential metrics. The MLP, a neural network method, adds complexity to the predictive modeling process. The use of ensemble learning, particularly stacking, aims to blend the unique abilities of different classifiers to create a more powerful bug prediction model. This model is then used to forecast future software issues, and a detailed analysis is conducted to measure the performance of each classifier, emphasizing the influence of ensemble learning.



- **Data Collection and Data Preprocessing**
  Gathering data is an important next step in the process, where we identify and collect relevant information sources related to software development and bug tracking. We gather historical data on software development activities, such as code changes, bug reports, and resolutions, to train our machine learning model. It is crucial to ensure the quality of the data by addressing issues like missing values and outliers at this stage. After that, we preprocess the collected data to get it ready for model training. This includes cleaning the data by handling missing values and outliers, as well as transforming the data through scaling, normalization, and encoding categorical variables.

- **Feature Engineering and Model Selection**
  In this phase, the AI identifies key elements from the gathered data. It can also generate fresh attributes based on its knowledge in a particular field to highlight distinct aspects of software creation and code structure. Choosing the right model is crucial, as it involves picking machine learning techniques like logistic regression, decision trees, random forests, SVM, and neural networks based on the issue and dataset. Various metrics are employed to evaluate different algorithms and select the most efficient one for further training.

- **Model Training and Model Evaluation**
  After preparing the data, the chosen machine learning model is trained. Hyperparameters are adjusted using techniques like cross-validation to avoid overfitting. The model's effectiveness is verified by assessing its performance on the training dataset. Next, the model is evaluated using a different validation dataset to test its generalization abilities. If needed, the model's parameters are fine-tuned to enhance its performance.

- **Model Deployment**
  After the model is trained and assessed, it is put into use to predict bugs in a live setting. The model is connected to current software development tools and processes, and its progress is monitored continuously. Regular updates and upkeep are carried out to maintain the model's effectiveness.

5. CONCLUSION

In order to find a bug, developers rely not only on the information provided in the bug report but also on their knowledge of the software project. We have developed a learning-to-rank technique that simulates how developers locate bugs. This model identifies important connections between a bug report and the source code files by taking into account domain knowledge, such as API specifications, code structure, and issue tracking data. Testing on six Java projects revealed that our method successfully identifies the relevant files in the top 10 suggestions for more than 70 percent of bug reports in Eclipse Platform and Tomcat. In addition, the ranking model we suggest is better than three other advanced methods. Through feature evaluation tests using greedy backward feature elimination, we found that all features are helpful. By analyzing the runtime alongside the feature evaluation results, we can choose a specific group of features to strike a balance between system precision and runtime complexity. Three AI models were utilized in the evaluation process: NB, DT, and ANNs. The evaluation was conducted using three real testing and debugging datasets. The experimental results were measured based on accuracy, precision, recall, F-measure, and RMSE. The findings indicate that machine learning techniques are effective in predicting future software bugs. The comparison revealed that the DT classifier outperformed the other models. Additionally, the experimental results demonstrated that the ML approach yielded better performance for the prediction model compared to linear AR and POWM models. In future work, other ML techniques will be explored to conduct a thorough comparison among them. In the future, we plan to incorporate additional machine learning methods and conduct a thorough comparison between them. Additionally, including a wider range of software metrics during the learning phase could improve the accuracy of our prediction model.

6. FUTURE ENHANCEMENTS

In our upcoming research, we will make use of more forms of expertise specific to the field, including the stack traces provided along with bug reports and the revision history of files. We will also incorporate characteristics that have been utilized in previous defect prediction systems. Additionally, we aim to employ ranking SVM with nonlinear kernels and extensively test this methodology on projects written in different programming languages.

7. ACKNOWLEDGEMENT

We are incredibly thankful to Mr. S. Anil Kumar for his exceptional guidance and support during our project. His expertise and motivation were essential to our success, and we are sincerely appreciative of his consistent commitment and mentorship. Additionally, we extend our gratitude to the faculty in the Computer Science and Engineering Department at Tirumala Engineering College for providing us with the opportunity to participate in this research project, which has been a valuable educational journey for us.

8. REFERENCES

1. Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan, Fatima Alsarayrah “Software Bug Prediction using Machine Learning Approach” (IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 9, No. 2, 2018
2. A. Sheta and D. Rine, “Modeling Incremental Faults of Software Testing Process Using AR Models ”, the Proceedings of 4th International Multi-Conferences on Computer Science and Information Technology (CSIT 2006), Amman, Jordan. Vol. 3. 2006.
3. M. M. Rosli, N. H. I. Teo, N. S. M. Yusop and N. S. Moham, "The Design of a Software Fault Prone Application Using Evolutionary Algorithm," IEEE Conference on Open Systems, 2011.
4. G. Antoniol and Y.-G. Gueheneuc,“Feature identification: A novel approach and a case study,” in Proc. 21st IEEE Int. Conf. Softw. Maintenance, Washington, DC, USA, 2005, pp. 357–366.
5. A. Bacchelli and C. Bird, “Expectations, outcomes, and challenges of modern code review,” in Proc. Int. Conf. Softw. Eng., Piscataway, NJ, USA, 2013, pp. 712–721.T.
6. J. Biggerstaff, B. G. Mitbander, and D. Webster, “The concept assignment problem in program understanding,” in Proc. 15th Int. Conf. Softw. Eng., Los Alamitos, CA, USA, 1993, pp. 482–498.
7. D. Binkley and D. Lawrie, “Learning to rank improves IR in SE,” in Proc. IEEE Int. Conf. Softw. Maintenance Evol., Washington, DC, USA, 2014, pp. 441–445.
8. Malhotra, Ruchika. "A systematic review of machine learning techniques for software fault prediction." Applied Soft Computing 27 (2015): 504-518.
9. https://www.researchgate.net/figure/Architecture-of-proposed-software-bug-prediction-system_fig2_378338847