Software Defect Prediction Using Machine Learning to Support Test Automation

Abstract
Software testing is a critical phase of the software development lifecycle, yet identifying defect-prone modules remains challenging due to increasing system complexity. Traditional test automation focuses on executing test cases but lacks intelligence in prioritizing high-risk components. This paper investigates the application of machine learning techniques for software defect prediction using static code metrics. A Logistic Regression model was trained and evaluated on a publicly available software defect dataset. Experimental results demonstrate that addressing class imbalance significantly improves the detection of defective modules, particularly recall, which is crucial for effective testing. The findings highlight the potential of integrating AI-driven defect prediction into test automation to support risk-based testing and improve overall testing efficiency.

1. Introduction

Software testing plays a crucial role in ensuring the quality and reliability of software systems. As modern software applications continue to grow in size and complexity, identifying defect-prone components has become increasingly challenging. Large codebases often contain hundreds or thousands of modules, making exhaustive testing difficult within limited time and resources. As a result, testing teams must make informed decisions about where to focus their testing efforts.

Test automation has significantly improved testing efficiency by enabling faster and repeatable execution of test cases. However, traditional test automation frameworks primarily focus on test execution and do not provide intelligence regarding which parts of the system are more likely to contain defects. This limitation can lead to equal testing effort being spent on both low-risk and high-risk modules, potentially delaying the discovery of critical defects.

Recent advancements in artificial intelligence and machine learning have opened new opportunities to enhance software testing practices. Machine learning models can analyze historical software data and identify patterns associated with defect-prone modules. By leveraging static code metrics such as size, complexity, and structural properties, machine learning techniques can assist testers in predicting potential defects before execution of test cases.

In this paper, we investigate the application of machine learning techniques for software defect prediction to support test automation. Using a publicly available software defect dataset, we develop and evaluate a Logistic Regression-based classification model. The study also addresses the challenge of class imbalance commonly found in defect datasets and demonstrates how imbalance-aware learning can significantly improve defect detection performance. The proposed approach aims to support risk-based testing by enabling better test prioritization and improving overall testing effectiveness.

2. Related Work

Software defect prediction has been widely studied as a means of improving software quality and testing efficiency. Previous research has explored the use of machine learning techniques such as Decision Trees, Logistic Regression, Support Vector Machines, and Neural Networks to identify defect-prone software modules using static code metrics. Many studies report promising results, demonstrating that software metrics related to size and complexity are useful indicators of defect occurrence.

Several researchers have highlighted challenges associated with defect prediction, particularly the issue of class imbalance, where defective modules represent a small portion

of the dataset. Imbalance-aware learning strategies and cost-sensitive models have been proposed to address this limitation and improve defect detection performance. While existing work establishes the feasibility of machine learning-based defect prediction, this study focuses on demonstrating how such techniques can practically support test automation by prioritizing high-risk modules and aligning evaluation metrics with real-world testing objectives.

3. Dataset Description

This study uses a publicly available software defect dataset obtained from the PROMISE repository. Specifically, the NASA CM1 dataset was selected due to its widespread use in software defect prediction research and its suitability for binary classification tasks. The dataset contains metrics collected from real software modules developed for a NASA project.

The CM1 dataset consists of 498 software modules, each described using 21 static code metrics. These metrics capture various characteristics of the source code, including size, complexity, and structural properties. Examples of such metrics include lines of code, cyclomatic complexity, and Halstead measures.

Each software module is labeled with a binary target variable indicating whether the module is defective or non-defective. This labeling enables supervised machine learning techniques to be applied for defect prediction. The dataset reflects a class imbalance, with defective modules representing a smaller proportion of the total instances, which is a common characteristic of real-world software defect data.

4. Methodology

The methodology followed in this study consists of data preprocessing, model training, and evaluation using supervised machine learning techniques. The overall workflow was designed to simulate a realistic defect prediction scenario in software testing.

Initially, the dataset was preprocessed to ensure suitability for machine learning. Irrelevant attributes such as module identifiers were removed, as they do not contribute to defect prediction. The target variable indicating defect presence was converted into a numerical format to enable binary classification. All remaining attributes were treated as input features representing static code metrics.

The preprocessed dataset was then divided into training and testing subsets using an 80–20 split. The training set was used to learn model parameters, while the testing set was reserved for unbiased evaluation on unseen data. A fixed random seed was applied to ensure reproducibility of the experimental results.

Before model training, feature scaling was performed using standardization to normalize the range of numerical attributes. This step was necessary to improve model convergence and ensure that features with larger numerical values did not dominate the learning process.

Logistic Regression was selected as the baseline classification model due to its simplicity, interpretability, and widespread use in defect prediction research. To address the issue of class imbalance present in the dataset, a balanced class weighting strategy was applied during model training. This approach assigns higher importance to minority class instances, enabling the model to better identify defect-prone modules.

The trained model was then evaluated on the test dataset using multiple performance metrics, including accuracy, precision, recall, and F1-score. This evaluation strategy ensured that the model's effectiveness was assessed not only in terms of overall correctness but also in its ability to detect defective modules, which is critical for software testing applications.

5. Experimental Results

The performance of the proposed defect prediction approach was evaluated using multiple classification metrics, including accuracy, precision, recall, and F1-score. These metrics were selected to provide a comprehensive assessment of model effectiveness, particularly in the presence of class imbalance.

As a baseline, a Logistic Regression model was trained without any imbalance handling. This baseline model achieved an overall accuracy of 87% on the test dataset. However, further analysis revealed that the model failed to correctly identify any defective modules, resulting in zero recall for the defective class. This outcome highlights the limitation of relying solely on accuracy in imbalanced defect prediction scenarios.

To address this issue, a class-balanced Logistic Regression model was trained by assigning higher importance to minority class instances. The balanced model achieved an overall accuracy of 81%. Although this represents a slight reduction in accuracy compared to the baseline, the model demonstrated a significant improvement in defect detection capability. Specifically, the recall for defective modules increased to 83%, indicating that the majority of defect-prone modules were correctly identified.

Table 1 presents a comparison of the baseline and balanced Logistic Regression models. The results demonstrate that incorporating imbalance-aware learning substantially improves the model's ability to detect defects, which is more critical for software testing applications than achieving high overall accuracy.

Model	Accuracy	Recall (Defect)	Defect(Prevision)
Logistic Regression(Baseline)	0.87	0.00	0.00
Logistic Regression(Balanced)	0.81	0.83	0.37

6. Discussion

The experimental results highlight important insights regarding the application of machine learning for software defect prediction. Although the baseline Logistic Regression model achieved relatively high accuracy, it failed to detect defective modules due to the imbalanced nature of the dataset. This finding demonstrates that accuracy alone is not a reliable metric for evaluating defect prediction models, particularly in software testing contexts where missing defects can have serious consequences.

By incorporating class imbalance handling, the balanced Logistic Regression model significantly improved recall for defective modules. This improvement indicates that the model became more effective at identifying high-risk components, even though it resulted in a modest reduction in overall accuracy. From a software testing perspective, this trade-off is acceptable and often desirable, as prioritizing defect detection reduces the likelihood of critical issues escaping into later stages of development.

The results also emphasize the importance of aligning machine learning evaluation criteria with real-world testing objectives. In test automation, false positives may lead to additional testing effort, but false negatives represent missed defects that can impact software quality and reliability. Therefore, models that prioritize recall for defect-prone modules provide greater practical value in supporting risk-based testing strategies.

Overall, the findings suggest that machine learning-based defect prediction can effectively complement traditional test automation by providing intelligence for test prioritization. However, careful consideration of data characteristics and evaluation metrics is essential to ensure that such models deliver meaningful benefits in real-world testing environments.

7. Conclusion and Future Work

This paper explored the application of machine learning techniques for software defect prediction to support test automation. Using static code metrics from a publicly available dataset, a Logistic Regression-based classification approach was developed and evaluated. The study demonstrated that while baseline models may achieve high accuracy, they can fail to detect defect-prone modules due to class imbalance.

By applying imbalance-aware learning, the proposed approach significantly improved recall for defective modules, which is a critical requirement in software testing. The results indicate that machine learning-based defect prediction can provide valuable insights for risk-based testing by helping testers prioritize high-risk components and allocate testing effort more effectively.

Future work may extend this study by evaluating additional machine learning models, such as ensemble-based approaches, and by incorporating multiple datasets to improve generalizability. Further research could also explore integrating defect prediction models into continuous integration and continuous testing pipelines, enabling real-time risk assessment during software development. Overall, this work highlights the potential of combining AI with test automation to enhance software quality and testing efficiency.

References

[1] T. Menzies, J. Greenwald, and A. Frank, “Data Mining Static Code Attributes to Learn Defect Predictors,” IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 2–13, 2007.

[2] N. E. Fenton and M. Neil, “A Critique of Software Defect Prediction Models,” IEEE Transactions on Software Engineering, vol. 25, no. 5, pp. 675–689, 1999.

[3] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, “The Misuse of the NASA Metrics Data Program Data Sets for Automated Software Defect Prediction,” IEEE International Workshop on Software Metrics, 2011.

[4] J. Nam and S. Kim, “Heterogeneous Defect Prediction,” Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 508–519, 2015.

[5] PROMISE Repository, “NASA Software Defect Data Sets,” Available: http://promise.site.uottawa.ca/SERepository