Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 02, NCRTCS - 2026

AI-Based Software Defect Predictors: Applications and Benefits

Sayali N. Chavan
Assistant Professor
Department of Computer Science
Padmashri Vikhe Patil College,Pravaranagar

Yogita B. Kharde
Assistant Professor
Department of Computer Science
Padmashri Vikhe Patil College,Pravaranagar

Abstract: - AI-based software defect prediction models leverage machine learning to identify potentially problematic areas in code, allowing for targeted testing and improved software quality. These models analyze historical data, such as code complexity, change history, and bug reports, to predict defect-prone modules

The application of a defect prediction model in a real-life setting is difficult because it requires software metrics and defect data from past projects to predict the defect-proneness of new projects. Software defect prediction aims to reduce software testing efforts by guiding testers through the defect-prone sections of software systems. Our results show that defect predictors can be used as supportive tools during a new process implementation, predict 75% of code defects, and decrease the testing time compared with 25% of the code defects detected through more labor- intensive strategies such as code reviews and formal checklists.

Keywords: AI-driven defect prediction, machine learning in software quality assurance, software reliability, defect prediction models, explainable AI, and test automation.

INTRODUCTION:

Software defects are more costly if discovered and fixed in the later stages of the development life cycle or during production. The testing phase should be planned carefully in order to save time and effort while detecting as many defects as analyzes that information and comes up with a claim or prediction on the subject. AI-based predictors learn specific patterns concerning defect-proneness from past projects and use this information to predict the defect proneness of new projects. As more projects are observed throughout the development life cycle, more data is collected, and predictions are more accurate. In this paper, we describe our model from a machine learning perspective with measurable benefits such as the defect detection capability and the cost-benefit analysis. We present the pay-off of the model used and show how the model has been implemented in the company. Our results show that our prediction model automatically finds 75% of the defects detected in unit testing, code reviews and inspections only in a few seconds. Therefore, we conclude that the impact of process changes is limited.

THE USE OF AI TECHNOLOGY

We have used a Naïve Bayes classifier as the algorithm of our prediction model. The Bayes Theorem defines the posterior probability as proportional to the prior probability of the class p(Ci), and the likelihood of attributes, p(X\Y=Ci) (cf. Alpaydin 2004). In binary classification problems such as defect prediction, Naïve Bayes computes the posterior probability of a module being defective, or the probability of a module being defect-free, given its attributes. Then, it assigns a module to the defective class if its posterior probability is greater than a pre-defined threshold (0.5). Otherwise, the module is classified as defect-free. Pattern Recognition plays a crucial role in defect prediction and prevention in software testing, particularly through the use of AI-powered testing tools. These AI-based software testing tools leverage advanced algorithms to analyze historical data and identify recurring patterns in code changes and defect occurrences. This result is very important for our case study since it reduces the necessity of trying all classification techniques. Thus, instead of applying different algorithms, we have selected Naïve Bayes as the algorithm of our model and focused on calibration based on local data.

DESCRIPTION OF THE PREDICTION MODEL:

Performance Evaluation We use Receiver Operator Characteristics (ROC) curves to assess the discriminative performance of a binary Naïve Bayes classifier (Heeger 1998). In a ROC curve, our objective is to reach the point (1, 0) in terms of (y, x), where the y-axis represents the true positive rate and the xaxis represents the false positive rate. We have computed these performance measures to evaluate the accuracy of our model. However, similar to defect

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 02, NCRTCS - 2026

prediction research (Menzies et al. 2007; Lessmann et al. 2008), we name the true positive rate as the probability of detection rate (pd) and the false positive rate as the probability of false alarm rate (pf) in this study. The ideal classification, point (1, 0) in a ROC curve can be reached when we correctly classify all defective modules (pd=1, i.e. 100%) with no false alarms (pf=0, i.e. 0%). Finally, the prediction outcomes depending on the actual class labels of modules can be represented as a confusion matrix as shown in Table 1. The common classifier performance measures are derived from this confusion matrix (Menzies et al. 2007).

Probability of the detection rate (PD) is a measure of accuracy for correctly classifying defective modules. It corresponds to the true positive rate in machine learning and should be as close to 1 as possible:

(PD) = TP / (TP + FN) (3)

Probability of the false alarm rate (PF) is a measure of accuracy to represent the false alarms when we misclassify defect-free modules. We must avoid high PF rates in software defect prediction models since they would increase the testing effort.

(PF) = FP / (FP + TN) (4)

It is very rare to achieve the ideal case with 100% PD and 0% PF rates using a prediction model. When the model is triggered often to increase the PD rate, the PF rate would, in turn, increase. Therefore, our objective is to get as high PD rates as possible while keeping the PF rates at a minimum.

Benefits of AI-Based Software Defect Predictors:

1. Early Detection of Defects

- AI models can identify potential bugs early in the development lifecycle.
- Early fixes reduce the cost and complexity of resolving issues later in the project.

2. Increased Development Efficiency

- Developers can prioritize high-risk areas for testing and review.
- Saves time and effort by focusing resources where they're most needed.

3. Reduced Maintenance Costss

- By preventing defects from reaching production, AI reduces the long-term cost of debugging and patching software.

4. Improved Software Quality

- Consistent defect detection leads to more robust and reliable software.
- Helps in achieving higher code quality standards and better user satisfaction.

CONCLUSION

AI-based software defect predictors represent a transformative advancement in software engineering, offering the ability to detect potential bugs and vulnerabilities early in the development lifecycle. By leveraging machine learning, deep learning, and data-driven techniques, these systems can analyze historical code data and software metrics to accurately forecast defect-prone modules. This proactive approach not only improves software quality and reliability but also reduces development costs and time. However, for maximum effectiveness, such systems must be trained on high-quality, representative data and continuously updated to adapt to evolving coding practices and technologies. As AI continues to advance, its integration into defect prediction tools promises even greater precision and automation, paving the way for more resilient and efficient software development processes.

REFERENCES:

[1] Introduction to machine learning. Alpaydin, E. eds 2004.
[2] Validation, Verification and Testing of Computer Software, Adrian, R.W., Branstad, M., Cherniavsky, C. J. 1982. ACM Computing Surveys (14), 22: 159-192.
[3] 3 Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings Lessmann, S., Baesens, B., Mues, C., Pietsch, S. 2008..
[4] Using Historical In-Process and Product Metrics for Early Estimation of Software Failures. In Proceedings of the International Symposium on Software Reliability Engineering, Nagappan, N., Ball, T., Murphy, B. 2006.
[5] Ostrand, T.J., Weyuker E.J., Bell, R.M. 2005. Predicting the Location and Number of Faults in Large Software Systems.