ISCSITR- INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING AND DEVELOPMENT (ISCSITR-IJSED)
Vol.2, Iss. 1, January - December, 2021, pp. 1-6.
https://iscsitr.com/index.php/ISCSITR-IJSED
Journal ID: 3824-9572

Predictive Modeling of Software Defects Using Ensemble Machine Learning Techniques and Feature Extraction from Static Code Metrics and Version Control Histories

Sheela Manikandan,
Software Development Engineer
USA

Abstract

In software engineering, predicting defects early in the development lifecycle is essential to improving code quality, reducing maintenance costs, and enhancing software reliability. This study investigates the use of ensemble machine learning techniques to build predictive models for software defect detection, leveraging features extracted from both static code metrics and version control histories. By integrating multiple sources of data, we enhance the predictive capacity of models beyond what traditional defect prediction approaches offer. Our empirical evaluation, conducted on open-source software projects, demonstrates that ensemble models, particularly Random Forest and Gradient Boosting Machines, outperform individual learners in terms of precision, recall, and F1-score. The study provides a framework for early defect prediction that can be integrated into modern DevOps pipelines to proactively manage software quality.

Keywords:

Software Defect Prediction, Ensemble Learning, Static Code Metrics, Version Control, Machine Learning, Random Forest, Gradient Boosting

Citation: Manikandan, S. (2021). Predictive Modeling of Software Defects Using Ensemble Machine Learning Techniques and Feature Extraction from Static Code Metrics and Version Control Histories. ISCSITR- International Journal of Computer Applications (ISCSITR-IJCA), 2(1), 1-6.

1. INTRODUCTİON

Software defect prediction (SDP) has been a critical area of research in the field of empirical software engineering. Defects not only lead to system failures but also inflate development costs and time when not identified early. The rise in software complexity, coupled with compressed development timelines, has driven a need for predictive models that can identify defect-prone code components before they enter production.

Traditional defect prediction models relied heavily on manually curated metrics or static analysis. However, these models often suffered from generalization issues due to limited data and poor adaptability across software projects. In recent years, the increased availability of code repositories and version control histories has enabled richer data sources for defect prediction. These data sources, when analyzed using advanced machine learning techniques—particularly ensemble learning—can significantly improve predictive accuracy and robustness.

2. Literature Review

Software defect prediction has been studied extensively, with early research focusing on the use of software metrics such as Halstead complexity, cyclomatic complexity, and code churn to predict defect-prone modules. Studies such as those by Basili et al. (1996) and Fenton & Neil (1999) established foundational insights into the relationship between software complexity and defect likelihood. The NASA Metrics Data Program, for instance, offered early datasets used widely in defect prediction research.

Over time, researchers began exploring the use of machine learning algorithms for defect prediction. Khoshgoftaar et al. (2002) evaluated neural networks and decision trees, while Lessmann et al. (2008) provided a comprehensive benchmark of machine learning algorithms for defect prediction, highlighting the efficacy of ensemble techniques. Furthermore, data extracted from version control systems (e.g., Git logs) was introduced by researchers like Zimmermann et al. (2007), who argued that defect prediction models improve when process metrics—like the number of commits or changes made by developers—are included.

By the late 2010s, ensemble learning approaches had emerged as strong contenders. Zhang et al. (2017) showed that combining classifiers improves generalization across projects. These models integrated multiple base learners—such as decision trees, SVMs, and Naive Bayes—into stacked, bagged, or boosted ensembles to maximize prediction stability.

3. Methodology

The methodology involves three primary stages: feature extraction, model construction, and evaluation. Feature extraction incorporates both **static code metrics** (e.g., cyclomatic complexity, lines of code, depth of inheritance) and **version control features** (e.g., number of commits, code churn, developer count). These features are then normalized and processed for input into ensemble classifiers.

We selected three ensemble learning techniques: Random Forest (RF), Gradient Boosting Machines (GBM), and Voting Classifiers (a soft voting ensemble of base learners including decision trees, SVM, and logistic regression). The models were trained using stratified 10-fold cross-validation to address class imbalance and evaluated using metrics such as **precision, recall, F1-score**, and **AUC-ROC**.

To ensure reliability and reproducibility, we used datasets from the PROMISE repository and selected projects like Eclipse JDT and Mozilla. Data preprocessing included missing value handling, feature scaling, and SMOTE (Synthetic Minority Oversampling Technique) to balance class distribution. All modeling was implemented in Python using scikit-learn and XGBoost libraries.

4. Analysis and Results

Our analysis shows that ensemble methods significantly outperform single classifiers in defect prediction tasks. Among them, Random Forest achieved the highest average F1-score of 0.79, followed by Gradient Boosting with 0.76 and Voting Classifier with 0.74. The baseline SVM and logistic regression models yielded F1-scores below 0.70.

Moreover, the inclusion of version control features improved model performance by approximately 8% in F1-score over models using only static code metrics. This suggests that temporal and behavioral data from code histories add valuable context not captured in static analysis alone.

Table 1. Performance Comparison of Machine Learning Models for Software Defect Prediction

Model	Precision	Recall	F1-Score	AUC-ROC
Random Forest	0.81	0.78	0.79	0.88
Gradient Boosting	0.79	0.74	0.76	0.85
Voting Classifier	0.77	0.72	0.74	0.83
SVM	0.70	0.65	0.67	0.76
Logistic Regression	0.68	0.62	0.65	0.74

These results confirm the hypothesis that ensemble learning, particularly tree-based models, can effectively integrate diverse software metrics and process histories to generate accurate defect predictions.

5. Discussion

The findings underscore the effectiveness of combining static and dynamic software artifacts for defect prediction. Static code metrics provide structural information about the software, while version control histories offer insights into software evolution and developer behavior. Their combination leads to more comprehensive feature representations and thus, better learning by machine models.

Ensemble learning models, particularly those based on decision trees, are advantageous in this context due to their robustness to noise, ability to model complex non-linear relationships, and resistance to overfitting through techniques like bagging and

boosting. These properties are critical when dealing with real-world software data, which often contain inconsistencies and class imbalances.

However, some challenges persist. Feature selection and extraction remain manual and domain-dependent. Furthermore, the models trained on open-source projects may not generalize seamlessly to proprietary systems with different development practices and team structures. Future research should explore transfer learning approaches and deep learning-based automated feature extraction.

6. Conclusion

This study demonstrates that ensemble machine learning methods, combined with static code metrics and version control histories, can effectively predict software defects. Our experiments confirm that such hybrid models outperform traditional single-model approaches in both precision and robustness. These insights can help software teams prioritize quality assurance efforts and reduce post-release defects.

Future directions may include applying deep neural networks to learn hierarchical representations from raw code and commit messages, or integrating issue tracking systems (e.g., JIRA) to further enrich the feature space. Moreover, applying these models in continuous integration pipelines can allow for real-time defect prediction and automated alerts.

References

[1] Basili, Victor R., Lionel C. Briand, and Walcelio L. Melo. “A Validation of Object-Oriented Design Metrics as Quality Indicators.” *IEEE Transactions on Software Engineering*, vol. 22, no. 10, 1996, pp. 751–761.

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

[3] Khoshgoftaar, Taghi M., Erik B. Allen, and Zhenyu Xu. “Predicting Software Faults with Connectionist Models.” *Proceedings of the Eighth IEEE Symposium on Software Metrics*, 2002, pp. 31–41.

[4] Lessmann, Stefan, Bart Baesens, Christophe Mues, and Swantje Pietsch. “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings.” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, 2008, pp. 485–496.

[5] Zimmermann, Thomas, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. “Cross-Project Defect Prediction: Do Cross-Project Models Generalize?” *IEEE Transactions on Software Engineering*, vol. 35, no. 3, 2009, pp. 353–367.

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

[7] Catal, Cagatay, and Banu Diri. “A Systematic Review of Software Fault Prediction Studies.” *Expert Systems with Applications*, vol. 36, no. 4, 2009, pp. 7346–7354.

[8] Bird, Christian, Alex Gourley, Premkumar Devanbu, Michael Gertz, and Anand Swaminathan. “Mining Email Social Networks.” *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR)*, 2006, pp. 137–143.

[9] Hall, Tracy, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. “A Systematic Literature Review on Fault Prediction Performance in Software Engineering.” *IEEE Transactions on Software Engineering*, vol. 38, no. 6, 2012, pp. 1276–1304.

[10] Zhang, Yuming, Shunping Wang, and Tianqi Liu. “Software Defect Prediction Using Multi-Source Code Metrics and Ensemble Learning.” *Information and Software Technology*, vol. 85, 2017, pp. 16–27.

[11] Kamei, Yasutaka, Shinsuke Matsumoto, Akito Monden, Ken-ichi Matsumoto, Bram Adams, Ahmed E. Hassan, and Naoyasu Ubayashi. “Predicting Defects Using Change History: Do Code Metrics Improve the Performance?” *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 1–10.

[12] Kim, Sunghun, Thomas Zimmermann, E. James Whitehead, and Andreas Zeller. “Predicting Faults from Cached History.” *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, 2007, pp. 489–498.