Code Quality Analysis Using Machine Learning

Kawthar Muhammad
Department of Computer Engineering, SSIT,
Skyline University of Nigeria (SUN)
kautharmuhammad965@gmail.com

Abstract

Ensuring high-quality software code is a critical aspect of software development, as poor code quality can lead to security vulnerabilities, maintainability challenges, and increased technical debt. Traditional methods of code quality analysis, such as manual code reviews and rule-based static analysis tools, often fail to scale effectively, especially in large and complex codebases. With the rapid advancements in artificial intelligence and machine learning, automated code quality analysis has emerged as a promising solution to enhance software reliability and maintainability. This paper explores the application of machine learning algorithms in automated code quality analysis, focusing on how these techniques can improve defect detection, code smell identification, and maintainability assessment. We discuss various supervised and unsupervised learning approaches, including deep learning models, decision trees, and ensemble methods, which have demonstrated significant potential in learning patterns from historical code repositories. By leveraging large datasets of annotated code samples, these models can predict quality issues with higher accuracy than traditional static analysis techniques.

Keywords—machine learning, code quality, defect detection, static analysis, maintainability, automation.

1. Introduction

Software development has become increasingly complex, with millions of lines of code powering modern applications. As software grows in scale, ensuring code quality becomes a significant challenge. Poor code quality can result in technical debt, increased maintenance costs, security vulnerabilities, and decreased performance. Traditional methods for assessing code quality, such as manual code reviews and rule-based static analysis tools, are often inefficient, time-consuming, and prone to human error. Moreover, as software teams adopt agile methodologies and continuous integration/continuous deployment (CI/CD) pipelines, there is a growing need for faster and more reliable methods of detecting code defects and enforcing coding standards.

Machine learning (ML) offers a powerful alternative to conventional code analysis techniques by enabling automated, intelligent assessment of code quality. Instead of relying solely on predefined rules, ML models can learn patterns from large datasets of annotated code, improving their ability to detect defects, identify code smells, and assess maintainability. However, despite its potential, the adoption of machine learning in code quality analysis is still evolving, with challenges related to model accuracy, interpretability, and real-world applicability.

2. Research Questions

- How can machine learning algorithms be effectively used to automate code quality analysis in modern software development?
- What types of machine learning techniques are most effective for detecting code quality issues?
- How does automated machine learning-based code analysis compare to traditional rule-based static analysis tools in terms of accuracy and efficiency?
- What are the key challenges in applying machine learning to automated code quality analysis, and how can they be addressed?
- How can automated machine learning-driven tools be integrated into CI/CD pipelines to provide real-time code quality feedback?
- Can machine learning models be trained to identify not only syntactic but also semantic issues in code quality assessment?

3. Literature Review

Automated code quality analysis aims to improve software quality by identifying defects, enhancing maintainability, and ensuring adherence to best practices. Traditional static and dynamic analysis tools rely on predefined rules and heuristics, which can lead to limitations in adaptability and generalization. Machine learning (ML)-based approaches

offer a data-driven alternative, capable of learning patterns from code repositories and improving over time.

Machine learning has increasingly been applied to various aspects of code quality, offering adaptability and insight beyond traditional methods. This review synthesizes foundational ideas and recent advances in ML for automated code quality analysis.

The reviewed study focuses on applying ML to static code analysis. Unlike dynamic analysis, which relies on runtime behavior, static analysis examines code without execution, providing early fault detection. ML integration allows tools to generalize from examples of faulty and correct code.

Several studies highlight ML's contributions:
- Bayesian models for error ranking (Kremenek et al.).
- Similarity-based approaches in plagiarism detection (e.g., SID system).
- Large Language Models (LLMs) used in automated review tools (Cihan et al.).

4. Methodology

This study employs a structured approach to predict software defects using machine learning models, leveraging the Metrics Data Program (MDP) dataset from Kaggle. The methodology includes data collection, preprocessing, feature selection, model implementation, and evaluation.

Data Collection: The MDP dataset includes software metrics and defect annotations. Key metrics include cyclomatic complexity, lines of code (LOC), and Halstead metrics.

Data Preprocessing:
- Cleaning: Address missing values and rectify inconsistencies.
- Normalization: Scale features to standardize value ranges.

Feature Selection: The Chi-Square test is used to select features strongly associated with defects, ensuring efficient model training.

Model Implementation: Three models were implemented:
- Random Forest (RF): Effective in high-dimensional data and highlights feature importance.
- Long Short-Term Memory (LSTM): Captures temporal patterns in sequential data.
- Gradient Boosting Machines (GBM): Builds sequential models for improved prediction accuracy.

Python-based frameworks such as TensorFlow and Scikit-learn were used for model development.

Model Evaluation: Performance was assessed using:
- Precision, Recall, F1 Score, Accuracy.
The dataset was split into 70% training and 30% testing.

5. References

1. [1] D. Silva, I. Nunes, and R. Terra, “Investigating code quality tools in the context of software engineering education,” Computer Applications in Engineering Education, vol. 25, no. 2, pp. 230–241, 2017.
2. [2] H. Keuning, B. Heeren, and J. Jeuring, “Code quality issues in student programs,” in Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp. 110–115, 2017.
3. [3] S. Jin, Z. Li, B. Chen, B. Zhu, and Y. Xia, “Software Code Quality Measurement: Implications from Metric Distributions,” in 2023 IEEE 23rd Int. Conf. on Software Quality, Reliability, and Security (QRS), pp. 488–496.
4. [4] A. Řechtáčková, R. Pelánek, and T. Effenberger, “Catalog of Code Quality Defects in Introductory Programming,” in Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, pp. 59–65.
5. [5] H. M. Chen, B. A. Nguyen, Y. X. Yan, and C. R. Dow, “Analysis of learning behavior in an automated programming assessment environment: A code quality perspective,” IEEE Access, vol. 8, pp. 167341–167354, 2020.
6. [6] U. Cihan et al., “Automated Code Review In Practice,” arXiv preprint arXiv:2412.18531, 2024.
7. [7] S. Axelsson, D. Baca, and R. Feldt, “Detecting defects with an interactive code review tool based on visualisation and machine learning,” in Proc. 21st Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), 2009.