Methodology summary
Code Quality Assessment

Overview

This report summarizes five key research papers related to AI-driven code quality assessment. Each paper's methodology is analyzed to extract four core aspects:

- Features trained on- Data sources- Models used- Model targets (predictions)

These insights will help in designing our own approach for assessing and improving code quality using machine learning.

Paper 1: Research Topics and Practices in Code Smell (IEEE, 2024)

- Features: Software metrics such as class size, coupling, cohesion, complexity, and lines of code used to detect code smells.
- Data Sources: Multiple open-source software repositories analyzed across 69 empirical studies.
- Models Used: Various machine-learning and deep-learning algorithms (Decision Tree J48, Random Forest, Naive Bayes, MLP, CNN, Genetic Programming, Whale Algorithm); also heuristic, rule-based, and static-analysis tools like SonarQube, JDeodorant, aDoctor, and SMAD.
- Target: Detect and classify code smells (e.g., God Class, Data Class, Feature Envy, Long Method) to assess and improve code quality.
- Link: https://ieeexplore.ieee.org/abstract/document/10322720

Paper 2: Software quality prediction using machine learning (ScienceDirect, 2022)

- Features trained on: Software quality metrics — process metrics, project metrics, and product metrics (covering reliability, functionality, efficiency, maintainability, portability, and usability).

- Data sources: Previously published studies and software project datasets from 2010–2021.
- Models used: Various machine learning algorithms for software quality prediction (specific models compared and analyzed across reviewed papers).
- Model target: Prediction of software quality attributes (e.g., reliability, maintainability, efficiency).
- Link: https://www.sciencedirect.com/science/article/pii/S2214785322014936

Paper 3: Evolution of Software Testing Strategies and Trends (IEEE, 2022)

- Features trained on: Textual data (titles, abstracts, author keywords) from software testing publications, preprocessed into word tokens, n-grams, and document-term matrices.
- Data sources: 14,684 software testing articles (1980–2019) extracted from SCOPUS-indexed journals and conferences in the software engineering field.
- Models used: Latent Dirichlet Allocation (LDA) with Gibbs sampling implemented using Python’s tmtoolkit package.
- Model target: Discovery of semantic topics and trends in software testing research over time (42 identified topics).
- Link: https://ieeexplore.ieee.org/abstract/document/9910177

Paper 4: A Federated Learning Based Approach for Code-Smell Detection (IEEE, 2024)

- Features trained on: Code-level metrics related to God Class code smells (e.g., class size, coupling, cohesion, complexity typical static code metrics).
- Data sources:
    - Codebases from multiple organizations participating in a federated learning (FL) setup.
    - Each organization keeps its source code private while training locally on its own data.
- Models used:

    - Federated Learning (FL) framework that aggregates locally trained ML models without sharing raw code data.
    - Compared against centralized ML models trained using traditional methods.
- Model targets (what is predicted): Detection of God Class code smells i.e., identifying classes in software systems that violate good design principles.
- Link: https://ieeexplore.ieee.org/abstract/document/10477413

Paper 5: Code Smell Detection Using Classification Approaches

- Features: Code metrics on classes, methods, parameters, coupling, and cohesion.
- Data Sources: Four datasets — God-Class, Data-Class, Feature-Envy, and Long-Method.
- Models Used: Random Forest, SVM, Naive Bayes, KNN, Logistic Regression (10-fold cross-validation).
- Target: Detect and classify code smells (smelly vs. non-smelly)
- Link: https://link.springer.com/chapter/10.1007/978-981-19-0901-6_25

Conclusion

The reviewed papers collectively provide a comprehensive understanding of how machine learning can be applied to assess and enhance code quality. They demonstrate that software metrics, such as coupling, cohesion, and complexity, can be effectively used as predictive features for identifying code smells and other quality issues. Various machine-learning and deep-learning algorithms — including Random Forest, SVM, Naive Bayes, and Federated Learning — have shown promising results in automating code quality evaluation across different datasets and contexts. These studies form a solid foundation for developing the **Code Quality Assessor**, a tool capable of scraping repositories, analyzing code metrics, and using trained models to detect defects and suggest improvements, ultimately supporting cleaner, more maintainable, and efficient software systems.