Severity Classification of Code Smells

Sanya Garg - 2022UCA1826
Kairavi Kumar - 2022UCA1823
Anurag Agarwal - 2022UCA1816

Table of Contents

Introduction

- **Code Smells**: Patterns in software indicating potential issues like poor design or inefficiency, impacting maintainability and quality.
- Examples: God Class, Data Class, Long Method, Feature Envy.
- Early detection reduces technical debt and improves software quality.
- **Machine Learning (ML)** aids in automating code smell detection, enhancing efficiency.

Motivation

- Existing tools lack consistency in detecting code smells across projects.
- Rule-based methods struggle to generalize across diverse code bases.
- ML models offer objective and reliable severity classification.
- ML leverages large datasets to uncover patterns missed in manual inspection.
- Goal: Use ML to enhance severity classification and refactoring efficiency.

Problems

- **Subjectivity in Detection**
- **Inconsistent Results**
- **Limited Detection Scope**
- **Lack of Context Awareness**
- **Need for Automation**

Contributions and Novelty

- **Machine Learning Models for Code Smell Severity Classification**: Development of machine learning models to classify the severity of four prevalent code smells—God Class, Data Class, Feature Envy, and Long Method.
- **Comparison of Machine Learning Approaches**: Comparative analysis of multinomial, ordinal, and regression models to identify the most effective approach for code smell severity classification.
- **Improvement Using SMOTE Resampling**: Application of the SMOTE (Synthetic Minority Over-sampling Technique) to address class imbalance and improve model accuracy.
- **Model Interpretability via LIME Algorithm**: Use of the LIME (Local Interpretable Model-agnostic Explanations) algorithm to enhance transparency and interpretability of model predictions.

Literature Work and Research Gap

- Prior studies emphasize detection but overlook severity classification.
- ML methods (e.g., SVM, Random Forest) improve detection but face scalability and data consistency issues.
- Challenges: Data imbalance, computational complexity, and limited interpretability.
- Research Gap: Need for interpretable ML models for severity classification.

Methodology

- **Data Collection**: Data gathered from open-source repositories to detect code smells.
- **Preprocessing**:
  - Handle missing values.
  - Apply SMOTE (Synthetic Minority Over-sampling Technique) to balance class distribution.
- **Machine Learning Models**:
  - Random Forest
  - Support Vector Machine (SVM)
  - Naive Bayes
- **Evaluation Metrics**:
  - Accuracy
  - Root Mean Square Error (RMSE)
  - Spearman's Correlation

Algorithms

- **Random Forest**:
  - Best performing model for code smell severity classification.
  - Robust to overfitting and handles large datasets effectively.
  - Utilizes an ensemble of decision trees for improved accuracy.
- **Support Vector Machine (SVM)**:
  - Effective for multi-class classification of code smells.
  - Performs well with smaller datasets but not as accurate as Random Forest.
  - Sensitive to the choice of kernel and hyperparameters.
- **Naive Bayes**:
  - Simple and computationally efficient algorithm.
  - Performs poorly on complex, high-dimensional datasets.
  - Assumes feature independence, which may not always hold in real-world scenarios.

Approach

- **SMOTE (Synthetic Minority Over-sampling Technique)**:
  - Used to address class imbalance in the dataset.
  - Generates synthetic samples for underrepresented classes to balance the dataset.
- **Model Interpretability with LIME (Local Interpretable Model-agnostic Explanations)**:
  - Helps explain machine learning model predictions.
  - Increases trust in the model by providing insights into why specific code smells are classified with certain severity levels.
- **Hyperparameter Tuning**:
  - Optimizes model performance through careful selection of parameters.
  - Techniques like grid search and cross-validation are used to achieve the best model configuration.

Hardware and Software

- **Hardware**: Standard desktop/laptop with 8GB RAM and 512GB storage.
- **Software**: Python (scikit-learn, pandas, NumPy), Jupyter Notebook.
- **Prototype**: Developed as a Python-based tool for detecting and classifying code smells.

Results

Random Forest:

- **Multinomial Model:** 94.5% accuracy, effective at distinguishing categories.
- **Ordinal Model:** 93.3% accuracy, effective for severity classification.
- **Regression Model:** 79.7% accuracy, slightly lower performance in continuous predictions.

SVM:

- **Multinomial Model:** 74.8% accuracy, moderate performance.
- **Ordinal Model:** 61.8% accuracy, struggles with severity levels.
- **Regression Model:** 47.1% accuracy, weakest in continuous prediction.

Naive Bayes:

- Consistent 72.5% accuracy across all models, but lags behind Random Forest.

Comparison Table

Classifier	Multinomial Accuracy	Ordinal Accuracy	Regression Accuracy
Random Forest	94.5%	93.3%	79.7%
SVM	74.8%	61.8%	47.1%
Naive Bayes	72.5%	72.5%	72.5%

Table: Accuracy Comparison of Classifiers

Observations

- **Random Forest** achieved the highest accuracy across all models, particularly in Multinomial and Ordinal classifications.
- **SVM** showed moderate accuracy in the Multinomial model but struggled with Ordinal and Regression models.
- **Naive Bayes** demonstrated stable but lower accuracy, especially in handling ordered and continuous predictions.
- **SMOTE** successfully addressed class imbalance, improving performance in minority classes.
- **LIME** enhanced model interpretability by highlighting the key features influencing predictions.

Issues

- **Data Quality Issues**: Missing values and incomplete data affected initial model performance.
- **Complexity of Code Smells**: Some code smells remain difficult to detect due to their inherent complexity.
- **Dataset Limitations**: The current dataset is small, requiring expansion to improve generalization across different projects.

Conclusion

- This study evaluated machine learning models (Random Forest, SVM, Naive Bayes, AdaBoost, XGBoost) for classifying code smell severity across four categories.
- Random Forest outperformed other models, excelling in multinomial and ordinal tasks.
- SMOTE improved performance by addressing class imbalance, leading to higher accuracy and lower error metrics.
- Ordinal models, especially Random Forest, captured the ordered nature of severity better than multinomial models.
- Feature selection improved performance by eliminating irrelevant variables and reducing overfitting.
- Naive Bayes and AdaBoost were less effective for complex code smells like God Class.

Future Work

- Explore alternative resampling techniques (e.g., ADASYN, BorderlineSMOTE) to improve model performance.
- Investigate deep learning models, such as neural networks and transformer-based architectures, for large-scale software repositories.
- Incorporate domain-specific knowledge for targeted feature engineering to improve classification accuracy.
- Experiment with automated feature selection methods (e.g., RFE, genetic algorithms) to enhance model interpretability.
- Conduct broader performance analysis across diverse datasets, incorporating precision, recall, and F1-score metrics.
- Investigate real-time application in integrated software development environments, creating tools to assist developers in identifying code smells.

Code/Github Link

- All code and models are available on GitHub.
- Repository link:
  `https://github.com/sanyagargg/Severity-Classification-of-Code-Smells`

Code/Github Link

- Sanya Garg : Code for Severity Classification, LaTeX file
- Kairavi : Code along with LaTeX presentation

Thank You!

Questions?