An Empirical Replication and Extension of Model Comparison Study on Predicting Co-Occurring Test and Source Code Refactoring

Zhouyiyang Yang
Department of ECE
University of Alberta
Edmonton, Canada
zhouyiya@ualberta.ca

***Abstract*—Refactoring is a crucial practice for maintaining software quality. While often applied to source code, test code requires refactoring to avoid quality decay and ensure efficacy. Recent research by Nagy and Abdalkareem revealed that refactoring of test and source code co-occurs in a significant portion (17.9%) of refactoring commits, and that a Random Forest classifier can predict this co-occurrence with moderate accuracy. This proposal outlines a plan to replicate the foundational findings of this prior work and extend it by conducting a comparative analysis of machine learning models. The project will first reproduce the original study's baseline (Random Forest) using the provided dataset. Subsequently, it will evaluate the performance of alternative models, such as Gradient Boosting Machines (XGBoost), and employ explainable AI techniques (SHAP) to gain deeper insights into the most influential features. The goal is to validate the existing results, identify a potentially more accurate predictive model, and enhance the interpretability of the predictions, contributing to a better understanding of the factors that trigger co-occurring refactorings.**

***Index Terms*—Software Refactoring, Test Code, Machine Learning, Empirical Study, Replication Study**

I. Introduction

Refactoring is a well-established practice to improve software internal quality without altering its external behavior. While extensively studied in the context of source code, refactoring of test code has received comparatively less attention. Test code is vital for software reliability, and its quality degradation through test smells can significantly impact its effectiveness. Nagy and Abdalkareem's recent study [1] investigated the co-occurrence of refactorings in both test and source code within the same commit. Their work established two key findings: 1) co-occurring refactoring commits are twice as common as test-only refactoring commits, and 2) a classifier built on features from source code refactorings can predict this co-occurrence with promising accuracy (AUC up to 0.92).

However, their study primarily relied on a Random Forest model. The field of machine learning offers a plethora of algorithms, each with different inductive biases and strengths. A natural and valuable extension is to explore whether other state-of-the-art models can achieve superior performance on this prediction task.

II. Background and Related Work

This project builds directly upon the work of Nagy and Abdalkareem [1]. Their study mined 77 Java projects from the SmartSHARK dataset [2], using RefactoringMiner [3] to identify refactoring operations and classify commits into three categories: source-only, test-only, and co-occurring. They extracted features related to refactoring size, developer experience, and code elements from the source code changes in a commit. Using these features, they trained a Random Forest classifier to predict whether a source code refactoring commit should also involve test code refactoring.

Other related work includes studies on test smells [4], [5] and their impact, as well as the broader literature on predicting software refactorings [6]. The distinct contribution of this project lies in its focus on the predictive modeling aspect of the test-source co-refactoring phenomenon, specifically through a comparative model analysis and interpretation.

III. Problem Statement and Research Gap

The specific challenge being investigated is the accurate prediction of co-occurring test and source code refactorings. Understanding this relationship can help build better developer tools that proactively suggest test refactoring opportunities when related source code is modified, thus improving overall codebase quality.

The limitation of the existing approach [1] is that it explores only one family of models (Random Forest) and provides limited model interpretability. This leaves open questions: (1) Could other models perform better? (2) What are the most robust features across different models? (3) Can we better understand the reasoning behind the predictions? Addressing these gaps will provide a more comprehensive understanding of the predictive landscape for this task and offer more interpretable results.

IV. Planned Approach

A. Overview

This project will be conducted in two main phases:

1) **Replication Phase**: Faithfully reproduce the key results from [1] (RQ1 and the Random Forest baseline for RQ2) to establish a validated baseline for comparison.
2) **Extension Phase**: Train and evaluate additional one or two (if time permits) machine learning models on the prediction task. Perform a comparative analysis of their performance and use explainable AI techniques to interpret the predictions and identify the most important features.

B. Step-by-Step Breakdown

- **Data Acquisition & Preprocessing**: Download the dataset from Zenodo [7]. Load it into a Pandas DataFrame. Perform necessary preprocessing (e.g., handling missing values, encoding categorical variables if any).
- **Replication of RQ1**: Analyze the dataset to calculate the distribution of commit types (source-only, test-only, co-occurring) and the most frequent test refactoring types, verifying the original study's findings.
- **Replication of RQ2 Baseline**: Implement a Random Forest classifier using `scikit-learn`. Precisely follow the experimental setup (10-fold cross-validation, SMOTE for handling class imbalance). Record performance metrics (AUC, F1-score, Precision, Recall).
- **Comparative Model Analysis**: Train and evaluate at least two other models. Strong candidates include:
  - **XGBoost**: A powerful gradient boosting framework known for its high performance on tabular data.
  - **Support Vector Machine (SVM)(if time permits)**: A classic model effective in high-dimensional spaces.
- **Model Interpretation**: Apply SHAP (SHapley Additive exPlanations) [8] to the best-performing model(s) to generate global and local explanations. This will identify the features that most strongly drive predictions.(if time permits)
- **Analysis & Reporting**: Statistically compare the performance of all models. Discuss the results, the most important features identified, and the implications of the findings.

C. Tools and Technologies

- **Python**: Primary programming language.
- **Libraries**: Pandas, NumPy, Scikit-learn, XGBoost, SHAP, Matplotlib/Seaborn.
- **Environment**: VS Code or Google Colab (if necessary).

These tools are industry standards for data science and machine learning, ensuring robust and reproducible results.

D. Potential Challenges and Mitigation

- **Challenge**: Reproducing the exact baseline results due to potential randomness in algorithms or SMOTE.
- **Mitigation**: Set random seeds for all algorithms and sampling procedures to ensure reproducibility.
- **Challenge**: Computational cost of training multiple models and running SHAP explanations.
- **Mitigation**: Use a subset of data for initial experimentation. Utilize Google Colab if necessary.

V. Planned Evaluation

A. Experimental Evaluation

The evaluation will be based on the standard metrics for binary classification: **Area Under the ROC Curve (AUC)**, **F1-Score**, **Precision**, and **Recall**. The models will be evaluated using 10-fold cross-validation.

B. Evaluation Metrics

Success will be measured by: 1. Successful replication of the original study's key statistics and baseline model performance. 2. The performance of the new models relative to the Random Forest baseline. A model will be considered superior if it shows a statistically significant improvement in AUC and F1-score. 3. The quality of insights derived from the model interpretation (SHAP analysis), providing a clear ranking of feature importance.

VI. Project Timeline

- **Week 4**: Data acquisition, environment setup.
- **Week 5-7**: Replication of the RQ1 and Random Forest baseline (RQ2).
- **Week 7-9**: Training, evaluation, and comparison of XGBoost (if time permits, I will add SVM).
- **Week 10**: SHAP analysis and interpretation of the best model.
- **Week 11**: Final analysis, report writing, and presentation preparation.

Acknowledgment

This proposal is based on the work and dataset provided by Nagy and Abdalkareem [1].

References

[1] N. A. Nagy and R. Abdalkareem, "On the co-occurrence of refactoring of test and source code," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022.

[2] A. Trautsch, F. Trautsch, and S. Herbold, "Msr mining challenge: The smartshark repository data," https://smartshark.github.io/, 2021.

[3] N. Tsantalis, A. Ketkar, and D. Dig, "Refactoringminer 2.0," *IEEE Transactions on Software Engineering*, 2020.

[4] M. Tufano, F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk, "An empirical investigation into the nature of test smells," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, p. 4–15.

[5] A. Peruma, K. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, and F. Palomba, "On the distribution of test smells in open source android applications: An exploratory study," in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, 2019, p. 192–202.

[6] E. A. AlOmar, M. W. Mkaouer, and A. Ouni, "Toward the automatic classification of self-admitted refactoring," *Journal of Systems and Software*, vol. 171, p. 110821, 2021.

[7] N. A. Nagy and R. Abdalkareem, "Refactoring co-occurrence replication scopus + data," 2022.

[8] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in Neural Information Processing Systems*, vol. 30, 2017.