

An Intelligent Machine Learning Framework for Predicting Software Defects Using Repository Mining

Deep Shaileshbhai Patel
(2504308)
Research Methods and
Professional Development
(ACFA7004)
Research Proposal
Prof. Dr. Seena Joseph

Research Context & Background

Modern software systems are increasingly complex and continuously evolving due to agile development and CI/CD practices.

Despite automation tools, software defects remain a major cause of:

- ▶ Project delays
- ▶ Increased maintenance cost
- ▶ Reduced reliability

Studies show defect fixing cost increases significantly in later lifecycle stages [1].

Current Developments (2019-2024)

Recent research trends include:

- ▶ Ensemble learning models (Random Forest, XGBoost) [2]
- ▶ Deep learning approaches
- ▶ Cross-project defect prediction [3]
- ▶ Explainable AI integration [4]
- ▶ Advanced imbalance handling techniques

Research Problem & Literature Gap

Although machine learning has improved prediction performance, key limitations remain:

- ▶ **Class imbalance** leads to biased models [5].
- ▶ High-performing models lack interpretability [4].
- ▶ Cross-project generalisation remains inconsistent [3].

Existing studies often address these issues separately.

There is limited research integrating:

- ▶ Robust imbalance handling
- ▶ Comparative ensemble benchmarking
- ▶ Explainable AI analysis

within a single experimentally validated framework.

Problem Statement

Current defect prediction approaches lack an integrated, explainable, and statistically validated framework capable of improving reliability and industrial adoption.

Research Aim

To design, implement, and rigorously evaluate an intelligent and explainable machine learning framework for predicting software defects using repository mining techniques.

Objectives

- ▶ Conduct a targeted review of contemporary defect prediction techniques (Month 1).
- ▶ Extract and preprocess repository-based metrics from public datasets (Month 2).
- ▶ Implement imbalance handling techniques such as SMOTE (Month 3).
- ▶ Develop and benchmark at least four ML models (Month 3-4).
- ▶ Integrate SHAP-based explainability mechanisms (Month 4).
- ▶ Evaluate model performance using cross-validation and statistical testing (Month 5).

Literature Review

Key Findings from Literature

- ▶ Ensemble methods outperform single classifiers [2].
- ▶ Imbalanced datasets distort evaluation metrics [5].
- ▶ Explainability improves model trust in industrial contexts [4].
- ▶ Cross-project learning remains unstable [3].

Critical Analysis

- ▶ Most prior research:
- ▶ Focuses only on improving accuracy.
- ▶ Uses limited validation techniques.
- ▶ Neglects interpretability integration.

Therefore, a unified experimental framework is justified.

Methodology

Research Design

Quantitative experimental research with comparative benchmarking.
Performance-based validation requires numerical measurement and statistical testing.

Data Collection

Public datasets:

- ▶ PROMISE dataset
- ▶ NASA MDP dataset
- ▶ Apache open-source repositories

These datasets are widely used in empirical software engineering research [2].

Evaluation Metrics

- Accuracy
- Precision
- Recall
- F1-score
- ROC-AUC
- Confusion Matrix
- Wilcoxon signed-rank test.

Experimental Workflow

- ▶ Data cleaning
- ▶ Feature engineering
- ▶ Imbalance treatment (SMOTE)
- ▶ Model development:
 - ▶ Random Forest
 - ▶ XGBoost
 - ▶ Logistic Regression
 - ▶ Neural Network
- ▶ 10-fold cross-validation
- ▶ SHAP explainability analysis

Project Planning & Resource

Phase	Duration
Literature Review	Month 1
Data Preparation	Month 2
Model Development	Month 3-4
Evaluation	Month 5
Writing & Revision	Month 6

Resources

Software:

- ▶ Python
- ▶ Scikit-learn
- ▶ XGBoost
- ▶ SHAP
- ▶ Pandas

Hardware:

- ▶ High-performance laptop / Google Colab GPU

Project Impact & Limitations

Academic Impact

- ▶ Advances empirical software engineering research.
- ▶ Contributes to explainable AI in defect prediction.

Industrial Impact

- ▶ Enables proactive defect detection.
- ▶ Reduces technical debt.
- ▶ Supports QA teams.

Limitations

- ▶ Dataset bias
- ▶ Limited generalisation
- ▶ Computational constraints
- ▶ Dependency on historical bug labeling accuracy

References

- [1] T. Menzies, J. Greenwald and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," vol. 33, no. 1, pp.2-13, 2007, doi: 10.1109/TSE.2007.256941.
- [2] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," vol. 34, no. 4, pp.485-496, 2008, doi: 10.1109/TSE.2008.35.
- [3] J. Nam, S. J. Pan and S. Kim, "Transfer defect learning", *2013 35th International Conference on Software Engineering (ICSE)*, pp. 382-391. [Online].doi: 10.1109/ICSE.2013.6606584.
- [4] S. M. Lundberg and S. Lee, "A unified approach to interpreting model predictions," vol. 30, 2017,
- [5] A. Nasiri, et al., "A Survey on Handling Imbalance Data on Software Defect Prediction", *2024 8th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, pp. 226-231. [Online].doi: 10.1109/ICITISEE63424.2024.10730271.

Thank YOU