Machine Learning-Based Testing Code Ranking

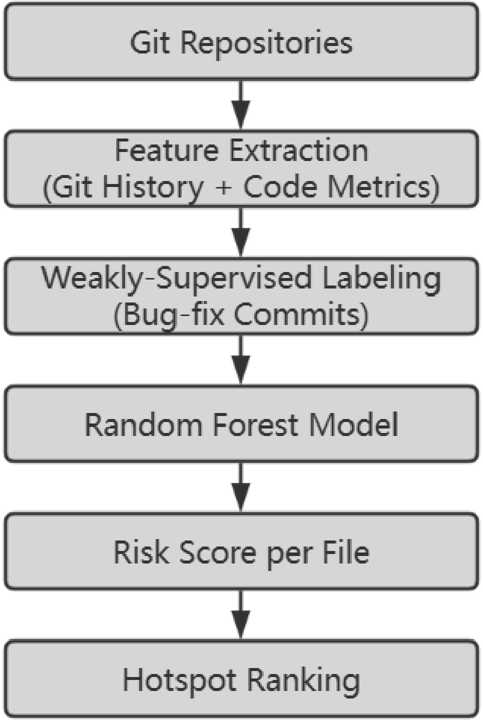
Problem & System Overview

Problem
Large code projects contain numerous files, and developers are unsure which to tackle first.

Goal
Use machine learning to determine which files need to be processed first.

What It Helps Users Do
This model helps users decide where to spend their limited testing and debugging effort by ranking files in the codebase.

This goes beyond bug prediction; it is an engineering prioritization system for testing.



Features & Labels

Features

Variable Name	Category	Meaning
commit_count	Git Evolution	Number of commits modifying the file
churn	Git Evolution	Total lines added and deleted
author_count	Git Evolution	Number of distinct file authors
cc	Code Structure	Average cyclomatic complexity[1]
mi	Code Structure	Maintainability index score[2]
loc	Code Structure	Lines of code in the file

Variable Name	Category	Meaning
label	Bug Label	1 if file appears in a bug fix commit (fix, bug, error…); otherwise 0

[1] McCabe, T. J. (1976). A complexity measure. IEEE Transactions on Software Engineering, 2(4), 308–320.
[2] Oman & Hagemeister (1992); Microsoft Maintainability Index; Radon.

Data & Model

Experiment Configuration

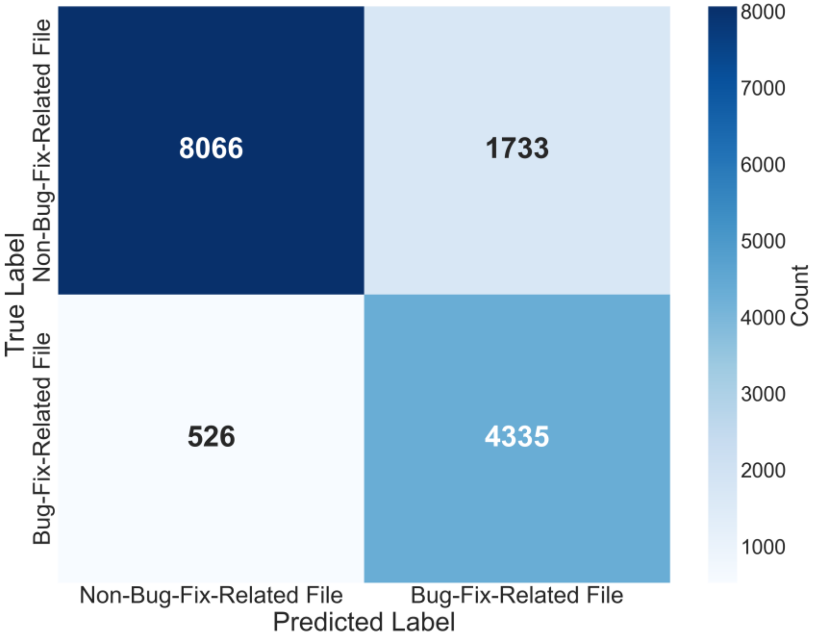
Item	Value
Model	**Random Forest Classifier**
Projects	8 *Python* projects
Train / Test Split	80% / 20%
Train files	58,640
Test files	14,660
Positive ratio	33.2%

The model **outputs** a ranked list of files by predicted risk, as shown below:

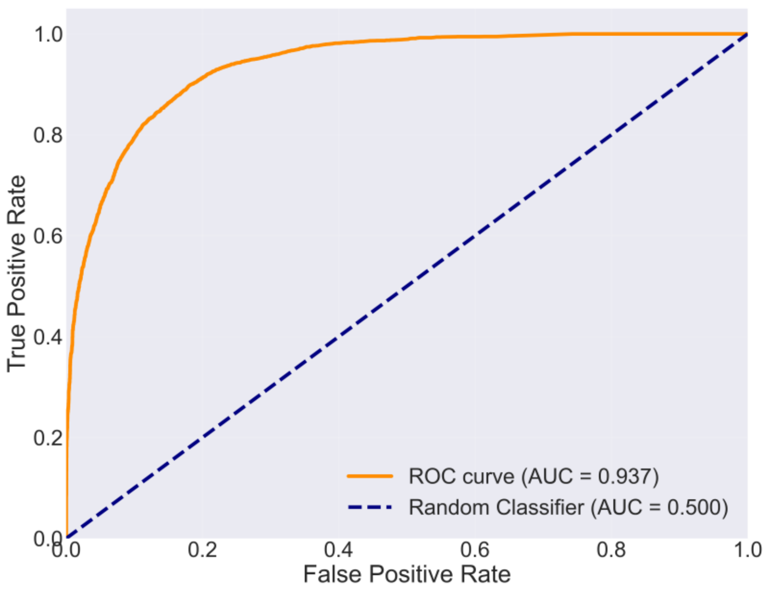
Rank	Project	File	Risk (Probability)
1	Airflow	scheduler_job.py	1.00
2	Ansible	play_iterator.py	0.98
…	…	…	…

Evaluation Results

Confusion Matrix Heatmap



ROC Curve



Model Outputs & Insights

Feature Importance

Feature	Importance
commit_count	0.44
author_count	0.27
churn	0.15
loc	0.06
cc	0.05
mi	0.03

The result indicates…

- Change frequency and ownership dominate bug risk
- Code metrics provide complementary signal
- Git history is the strongest predictor

Limitations & Next Steps

Limitations

- **Label noise:** Labels derived from git commit may not always be accurate.
- **Historical dependency:** Files with little or no version control history are harder to evaluate accurately.
- **Project specific bias:** The model may overfit to patterns from specific projects or teams.

Next Steps

- **Multi-language support:** Extend code metrics to Java, Go, and JavaScript.
- **Improve the model and feature engineering:** Refine feature design and modeling strategies to enhance robustness.

Thank you!