INTRODUCTION

Defect prediction is an important element of code based projects. Targeted prediction is critical in large updates, where widespread changes can swamp review and test capacity.

METHODOLOGY

We leveraged prior experience to build a random-forest model. We then selected a training database, which had to be active, large, and well documented in order to provide the information we needed to assess the performance of our model. We trained on the active, well-documented llama_index repo, whose many open issues supplied ground-truth labels.

From commit history we extracted features such as commit count, cyclomatic complexity, file age, and bug-fix frequency. This allows the model to analyze the files in commits, collect information, and then make assumptions based on patterns from the past.

Applied to staff repos of varied size, the model almost consistently flagged the top known buggy files.

INSIGHTS

After analyzing the performance of our prediction model on several repos of varying sizes, we found that several features are more likely to be a predictor of bugs than others. Some of these features, and our estimates of why these types of features lead to bugs, are as follows:

1. **Number of Authors**
    a. More distinct authors often means mixed styles and conflicting changes, raising defect risk.
2. **Weighted bugs**
    a. weighted_bug_density = bug_ratio * (1 + 0.5 * (complexity_factor - 1))
        i. bug_ratio = bug_fix_count / n_commits
        ii. complexity_factor = file_complexity / avg_repo_complexity
    b. Higher weighted bug density signals files with both past defects and above-average complexity.
3. **Relative Risk**
    a. relative_risk = weighted_bug_density * (1 + 0.3 * (commit_factor - 1))
        i. commit_factor = file_commits / avg_repo_commits
    b. Relative risk scales bug density by commit frequency, offsetting age-related skew.

CUSTOM HARNESS JUSTIFICATION

While a standard harness was provided, we continued using our custom implementation as it was already integrated with our machine learning pipeline and advanced metrics. Changing our functionality into the provided harness would have required significant rework without adding any benefits to our prediction capabilities

RESULTS

We found that relative risk proved to be an okay predictor of the top files. When sorted by highest risk, we had relatively accurate results in timeframes where there was sufficient commit history. However, our model lacked performance in short time periods, and in projects that had very little commits, or small time frame, which resulted in either large accuracies for short sprints, or failing to create any predictions. We felt that although it could be a great idea to create a system that adapts to each project, it falls short due to inconsistencies. See our results table in the repo for a complete list.

CONCLUSION

We felt these three metrics provided the best combination of factors in order to predict the likelihood of a defect. They consistently identified the most error-prone files across diverse repositories, letting testers focus reviews where they matter most. We believe that some of the failures of our project were likely caused by overfitting in training due to the extremely small ground truth files we had to work with. Because we did not know where the ground truths were coming from, when filtered to just our top 3 risky files, we tended to miss, although expanding the list to 5 often included these files. Short windows of repos were included late in the project, which also caused issues. Future work will refine thresholds and integrate the model into continuous integration pipelines for real-time alerts, as well as working to improve the training system to combat overfitting of the models