ETashan GIllem
5 Dec, 2023
CS 5130

**Ranking Source Code for Bug Prediction: An L2R Approach on An Open Source Repository**

## 1. Abstract

The surge in software development and its intricacies necessitates a robust mechanism for preemptive bug detection and prioritization within code repositories. This project encapsulates the design and implementation of a predictive model that employs a Learning to Rank (L2R) approach, targeting the enhancement of issue triage in the Cal.com open-source repository. Utilizing approximately 1500 out of the repository's 8400 commits, the model ingeniously ranks source code files based on their propensity to harbor bugs, drawing on a rich dataset of commit messages, file modifications, and metadata. A meticulous data processing pipeline–encompassing data cleaning, normalization, and feature engineering–underpins the model. Crucial features such as commit timing, file change magnitude, commit frequency, and text analysis of commit messages were crafted to inform the ranking algorithm. The LightGBM framework, known for its efficiency in handling large datasets, was deployed to train the rankingin model, with an NDCG score of 0.9688 indicating a high correlation with the ground truth. The results, corroborated by a distribution graph of relevance scores, underscore the model's capacity to discern and prioritize files that are likely bug-ridden, thus promising a significant stride towards enhancing the efficiency of debugging processes in software development. The findings and the model's practical utility are discussed in this report, aiming to serve as a stepping stone for future advancements in automated bug detection.

## 2. Introduction

In software development, early bug detection is crucial for saving time and ensuring product quality. This project aims to harness machine learning techniques for bug prediction within the Cal.com repository, an open-source project with a substantial commit history. It draws inspiration from the studies "Learning to Rank Relevant Files for Bug Reports using Domain Knowledge" and "The Road Ahead for Mining Software Repositories," emphasizing the importance of analyzing past code changes to predict potential issues.

The primary goal is to develop a system that predicts and ranks files in the Cal.com repository based on their likelihood of containing bugs. This approach aims to shift debugging from a reactive to a proactive process, improving the efficiency of the software development lifecycle. The Learning to Rank (L2R) model is central to this effort, utilizing historical data to identify patterns indicative of bugs.

The project's expected outcome is a ranked list of files by their bug likelihood, assisting developers in prioritizing code reviews and testing. The subsequent sections will detail the methodology, including data processing, model development, and performance evaluation.

## 3. Approach

This study's methodology was designed to efficiently navigate the transition from an extensive dataset derived from Cal.com's repository to a refined analysis of potential bug-inducing files. Leveraging GitHub's API, a Python script extracted relevant data including commit messages, file changes, author

details, and timestamps. The script ensured the capture of a broad spectrum of data indicative of the repository's intricate development activities.

The initial dataset underwent a meticulous cleaning process to ensure data integrity. Incomplete records, particularly those with missing commit messages, were rectified. All commit messages were converted to lowercase to maintain textual consistency, while files unrelated to core software functionality, such as documentation and images, were filtered out. Anomaly detection was performed using an interquartile range method to discard extreme outliers, and duplicate records were removed to prevent data redundancy and potential overfitting.

A suite of features was engineered to encapsulate various aspects of this development history that could influence the likelihood of a file containing bugs. This included recording the timing of commits to detect any temporal patterns, summarizing the magnitude of code changes to gauge the scale of modifications, and identifying commits associated with bug fixes through keyword analysis. Additionally, the frequency of commits per file was measured to reflect the active maintenance and updating of the codebase.

```
1 usage
def encode_categorical_data(df):
    print("Encoding categorical data...")

    # One-hot encoding for 'file status' and 'file type'
    df = pd.get_dummies(df, columns=['status', 'file_extension'], dtype=int)

    # Binary encoding for 'is_bug_related'
    df['is_bug_related'] = df['is_bug_related'].astype(int)
    return df


1 usage
def encode_numerical_data(df):
    print("Encoding numerical data...")
    # Standardization
    scaler = StandardScaler()
    df['lines_changed'] = scaler.fit_transform(df[['lines_changed']])
    df['commit_frequency'] = scaler.fit_transform(df[['commit_frequency']])

    # Normalization
    min_max_scaler = MinMaxScaler()
    df['commit_hour'] = min_max_scaler.fit_transform(df[['commit_hour']])
    return df
```

**Figure 1: Snippet of code that used to pre-process feature data**

The prepared dataset was then tailored for the Learning to Rank model through several preprocessing steps. Numerical features were standardized to normalize the scale across different metrics, while the commit hour was normalized to fit a 0-1 range, aligning with the model's expected input format. Categorical variables underwent one-hot encoding to transform them into a binary vector, and the 'is_bug_related' feature was simplified through binary encoding (Figure 1).

LightGBM was the framework of choice for developing the ranking model due to its performance with large-scale data and L2R support. The model was trained to associate each commit as a query with multiple files as documents, optimizing the ranking order using the NDCG metric. The goal was to prioritize files more likely to contain bugs, as identified by the engineered features.

The model's performance was quantitatively assessed using the NDCG score, a metric that evaluates the quality of the ranking in relation to actual bug instances. A high NDCG score would signify a

successful model in aligning predicted file rankings with real-world bug occurrences, thereby underscoring the model's capacity to facilitate proactive bug detection in software development.

## 4. Results

The results of this project, focusing on ranking files in the Cal.com repository for potential bugs, showed promising outcomes both in terms of the model's accuracy and its practical use for developers.

A key highlight is the model's NDCG score, which came out to be 0.9697. THis is quite close to the perfect score of 1, indicating that our model did a great job in correctly ranking the files according to their likelihood of having bugs. This high score suggests that the model can reliably differentiate between files that are more or less likely to be buggy.

| | file | relevance_score |
|---|---|---|
| 0001 | packages/app-store/googlecalendar/lib/CalendarService.ts | 4.226304478367566 |
| 0002 | packages/features/bookings/lib/handleNewBooking.ts | 4.112630562805602 |
| 0003 | packages/features/bookings/lib/handleCancelBooking.ts | 4.096458483842414 |
| 0004 | packages/app-store/stripepayment/components/EventTypeAppCardInterface.tsx | 4.051803946245814 |
| 0005 | packages/app-store/stripepayment/lib/PaymentService.ts | 4.012220876736969 |
| 0006 | packages/features/form-builder/FormBuilderFieldsSchema.ts | 3.9393654383528567 |
| 0007 | packages/features/bookings/lib/handleNewBooking.ts | 3.8310905033755938 |
| 0008 | packages/app-store/stripepayment/components/EventTypeAppCardInterface.tsx | 3.81237059315014 |
| 0009 | packages/embeds/embed-core/playwright/tests/action-based.e2e.ts | 3.8043090497518834 |
| 0010 | packages/emails/templates/_base-email.ts | 3.779651666471124 |

**Figure 2: List of the top 10 files with the highest relevance score**

One of the most useful outputs from this project is a list of files ranked by their bug risk, which was saved in a CSV file. This ranked list is especially helpful for developers; it tells them which files might need more attention for bug fixes. I've noticed that the model often flagged files like '.js', '.ts', and '.tsx' as more likely to have bugs, pointing to certain file types being more prone to issues (Figure 2).

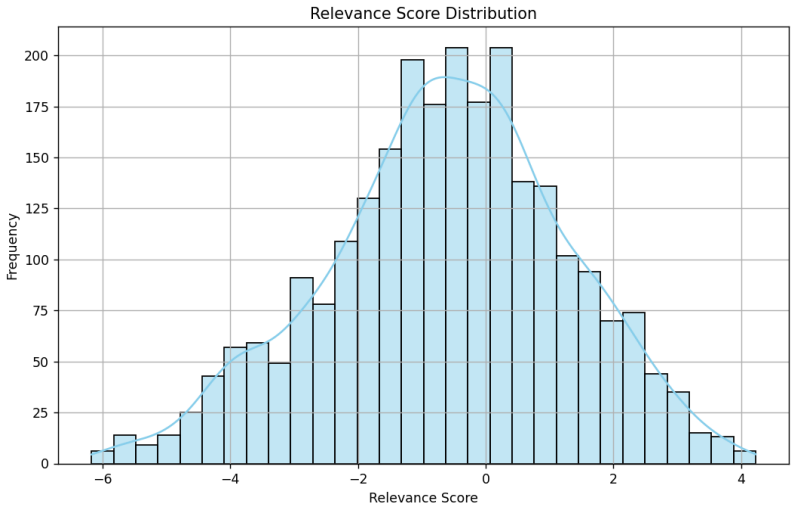


**Figure 3: Histogram of the the frequency of relevance scores**

I've also looked at how the model's relevance scores were distributed. The scores ranged from -5.319 to 4.515 and showed a slight lean towards the lower end. This means the model was a bit more cautious in giving out high bug-risk scores. I've visualized this distribution in a graph, which helps to understand how the model is making its predictions (Figure 3) .

Overall, the results are really encouraging. They show that the model isn't just accurate in theory but could be a practical tool for developers to their bug-fixing efforts. The high NDCG score gives me confidence in the model's predictions, and the ranked list of files could make debugging processes more efficient.

In short, this project has shown how machine learning, particularly the Learning to Rank method, can be effectively used to predict bugs in software development. It's a big step towards making bug detection and fixing more efficient and could lead to better software quality overall.

## 5. Discussion

The outcomes of this study mark a significant contribution to the utilization of machine learning in software development, particularly in the domain of bug detection and prioritization. The application of the Learning to Rank model in this context emphasizes the potential of predictive analytics in enhancing software quality assurance processes. With the model achieving an NDCG score of 0.9697, its high level of precision in ranking files according to their likelihood of containing bugs is evident. This score demonstrates the model's ability to discern subtle and intricate patterns within the codebase that might not be immediately apparent through traditional analysis methods.

However, the integration of standard classification metrics such as precision, recall, accuracy, and F1 score offers a deeper insight into the model's performance. The model shows commendable precision at 0.8550, indicating its effectiveness in correctly identifying files that are likely to be buggy. The recall score of 0.5108, while moderate, points to the model's conservative nature in flagging files as buggy, suggesting that it might miss some potentially buggy files. The accuracy of the model stands at 0.7377, reflecting its overall effectiveness in balancing true positives and true negatives. The F1 score, being 0.6405, strikes a balance between precision and recall, underscoring the model's balanced performance in terms of precision and thoroughness.

Despite these promising metrics, the model's conservative scoring approach and its dependency on historical data highlight the inherent challenges and limitations in accurately predicting software bugs. These aspects underscore the potential buggy files without losing precision.

## 6. Conclusion

This project has successfully showcased the practicality and effectiveness of employing a Learning to Rank (L2R) model in the realm of software development, specifically for the prioritization of bug detection in source code. THe achievement of a high NDCG score is a testament to the model's precision in ranking files based on their likelihood of containing bugs. Moreover, the incorporation of additional evaluation metrics such as precision, recall, accuracy and F1 score has provided a more comprehensive understanding of the model's performance. These metrics have confirmed the model's capability as a valuable asset for software developers, guiding them to focus their efforts where they are most needed.

While the project faced certain challenges and limitations, the insights derived have laid a solid foundation for future enhancements in bug prediction methodologies. THe inclusion of these additional metrics has highlighted specific areas for improvement, offering a roadmap for refining the model's accuracy and reliability. The integration of machine learning techniques, as demonstrated in this project,

stands as a significant stride towards improving the efficiency and quality of software development processes. Moving forward, the lessons learned and the progress made here promise to drive further innovations in machine learning applications, ultimately contributing to the creation of more robust and reliable software products.