SARAH SAGRI-IIT2024019

Evaluation Report

I had to build a model in which a developer's role was to be predicted by the model

ColumnTransformer and Pipeline objects: I prevented data leakage through this. Three models were trained and evaluated: A Logistic Regression baseline, an improved LightGBM model, and an XGBoost model. The final LightGBM model was then fine-tuned using RandomizedSearchCV to optimize for the primary evaluation metric.

Model Evaluation

The main evaluation was done according to **Macro F1 Score**, because it is goodd when there is class imbalance, identified during EDA.

Final Model Confusion Matrix:

The confusion matrix for the LightGBM model showed a very good performance on the diagonal, with high accuracy for the frontend, backend, and qa roles.

Major Failure Modes

A qualitative analysis of the model's prediction errors revealed two primary failure modes:

1. **Fullstack vs. Backend/Frontend confusion for the model**: The model's most frequent errors involved interpreting the fullstack commits wrongly. When a fullstack developer's commit message was heavily focused on backend tasks (e.g., "fix API authentication query"), the model would confidently, but incorrectly, predict backend. Because vocabulary and file types (.py, .js_ts) used by these roles are very similar and almost all file types used by one will affect the other role so it has to be modified as well according to me. Someone may not have full fullstack knowledge but have minimal knowledge so as to modify the other end according to what they want to perform.
2. **Noise in Commit Messages**: messages which were not giving any info regarding what changes were made (e.g., "minor fixes," "refactor component") were of minimum use, forcing the model to rely more on file extensions. In cases where the file extensions were also ambiguous, this led to incorrect predictions.

Lessons Learned & Betterments (According to me)

This project showed the predictive power of good features, particularly the tf idf representation of fileextensions and commitmessage.

For future work, two paths could yield significant improvements:

1) **Advanced Text Features**: Using pre-trained sentence embeddings (e.g., from Sentence-Transformers) instead of TF-IDF, because then the model would actually understand the meaning of the messeges and the problem I stated above that there were commit messages

which were not giving any hints would be useful too, also the messages with content could have more role in predictions

2) **Feature Interaction**: Exploring features that capture the interaction between commit type and text (like, do bugfix messages use different words than feature messages?) could provide a better signal to the model.

Justification of Design Decisions

Principle of "reasoning over recipes" was kept in mind by me, I explained each and every step through comments, I also focused on reproducibility (by pipelines, columnTransformer, using relative paths, fixing the seeds and not keeping them random, etc). The most important design decisions were the structure of the preprocessing pipeline, the feature engineering through data, and the evaluation strategy.

First, the decision to use a unified sklearn pipeline with a columnTransformer was because this structure covered all preprocessing steps, from text vectorization to numerical scaling, into a single object, which can then be used in different models testing along with some minor changes.

Second, my feature engineering was directly by the Exploratory Data Analysis. The EDA revealed that fileextensions and commitmessage contained powerful, distinct signals for each role. Instead of just counting files, I treated the file extensions as a "document" and applied a TF-IDF vectorizer. This captured the unique combination of technologies used in each commit and proved to be one of the most predictive features in the final model.

Finally, the EDA also showed a significant class imbalance. The use of stratify=y in the train-test split to ensure representative samples, and the choice of **Macro F1 Score** as the primary metric to guarantee that performance on the qa class was weighted equally. This prevented the above stated imbalance.

This document provides plain-language annotations for ten commits, explaining the label assignment based on the data footprint.

1. Index 6 (Backend): Labeled backend because the commit modifies only Python (.py) files and the message is "authentication logic in backend API" and "database query".
2. Index 0 (Frontend): Labeled frontend because the commit modifies JavaScript/TypeScript (.js_ts) files and the message is "responsive UI component," "dropdown," "modal," and "CSS and animation effects."
3. Index 12 (QA): Labeled qa because the commit modifies a test file (test_js) and the message starts with "QA:" itself and describes updating test data for model training.
4. Index 8 (Fullstack): Labeled fullstack because the commit involves a mix of frontend (.css, .js_ts) and backend (.py) files, and the message describes both UI work ("responsive UI component") and backend logic ("API and database validation").
5. Index 3 (Frontend): Labeled frontend because the commit is a refactor of JavaScript/TypeScript files (.js_ts) and the message describes updating the "responsive layout and CSS style consistency."
6. Index 4 (Fullstack)**:** Labeled fullstack because the commit message describes both frontend work ("Implement responsive UI layout for login page") and backend work ("Updated API endpoint to handle authentication and session logic").

7. Index 14 (Backend): Labeled backend because the commit is a bugfix in a SQL file (
8. .sql) and the message is "authentication logic in backend API" and updating the "database schema."
9. Index 28 (QA): Labeled qa because the commit modifies Python test files (.test_py) and the message states, "Added QA test cases for commit message analysis model."
10. Index 58 (Fullstack): Labeled fullstack because the commit modifies both frontend (.css) and backend (.py) files and the message is "responsive layout issue on login page UI" as well as "backend API endpoint to handle authentication."
11. Index 68 (Backend): Labeled backend because the commit is a bugfix in a Python file (.py) and the message is "database query and schema" and "token session management for secure login endpoint."