Lab 0: N-Gram Code Suggestion Model

Saurav Banarjee and Asher Montague

Dataset Creation:

Student Mode:

Data Collection

extract_methods_from_java() and *process_student_csv()*

The script begins by accepting a .csv file containing a set of GitHub repositories. It then extracts the java methods from these repositories into a new .csv file.

Data Preprocessing

remove_duplicates(), remove_outliers(), remove_boilerplate_methods(), clean_and_tokenize(), tokenize_methods()

Using the created .csv file of methods, the script then preprocesses the script by importing it into a dataframe, then removing duplicate methods, filtering for ASCII, removing outliers and boilerplate methods, and then finally iterating through the dataframe, cleaning and tokenizing each method.

Teacher Mode:

Unlike the student mode, teacher mode accepts a .txt file where every line is an already tokenized Java method. It then adds these methods to a .csv file, which can then be processed in the same way as in student mode.

Model Training:

train_model.py

After processing the data, the model then executes the training pipeline. Specifically, it randomizes the training corpus, dividing it into training, evaluation, and testing sets based on the proportions in *run_pipeline()*. It then trains a series of N-Gram models for values of n between 3 and 10.

Model Evaluation:

train_model.py

tvaluate_model.py

After training the models, the script then calculates the perplexity of each model using the evaluation corpus in *calculate_perplexity.py*, and saves the best model to a results .json file named student or teacher depending on which corpus it was trained on.

Conclusions:

While the preprocessing stage should have been relatively easy due to the code being provided, actually integrating it with our training pipeline proved to be a substantial bottleneck and cost a lot of time. Calculating perplexity, on the other hand, proved to be relatively straightforward.