GenAI for Software Development: Assignment 1

Jaden Kingsley
jakingsley@wm.edu

Introduction

Code completion is a crucial feature in software development, assisting developers by predicting and suggesting the next tokens in an incomplete code snippet. In this project, I implement an N-gram probabilistic language model for Java code completion. The N-gram model predicts the next token in a sequence by learning the probabilities of token occurrences n-tokens before the next. To build the model, I first gathered a dataset of Java repositories using the GitHub Search tool, then extracted, cleaned, and tokenized the methods. The corpus was split into training, evaluation, and test sets. I evaluated the models based on perplexity - a lower perplexity indicates a better performing model. This report details the process, along with insights into the model's performance on different datasets. The source code for the model can be found at https://github.com/jakingsleyWM/N-GramCodeRecommender.git .

Implementation

Dataset Preparation: I used the GitHub Search tool (https://seart-ghs.si.usi.ch/) to gather my corpus of repositories. Using the following filters, I gathered a list of 123 repositories: language = “Java”, minimum number of commits = 100, minimum number of stars = 5000, maximum number of code lines = 50000, last commit was made within 6 months, and excluding forks. From this data selection, I randomly selected a section of 4 repositories to use as the corpus for the model. I cloned and extracted 363,284 methods by processing each commit to the “main” branch and using the JavaLang package.

Cleaning: I made sure to include only relevant, useful methods by removing duplicates, removing those which included non-ASCII characters, removing length outliers (outside the 5th-95th percentile), removing comments, and removing boilerplate methods (also known as “setters” and “getters”). After cleaning, the corpus contained 41,493 methods.

Tokenization: I used the Pygments package to tokenize the methods.

Dataset Splitting: To create the training, test and evaluation sets, I randomly split the corpus into 80% for training, 10% for testing and 10% for evaluation.

Model Training & Evaluation: I trained three different N-gram models with varying context window sizes to assess their performances. I used n = 3, n = 5, and n = 9. I used perplexity to evaluate the performance of each n-value, with a lower perplexity associated with higher performance. After evaluating the model on the evaluation set, I selected n = 5 as the highest-performing model, with a perplexity of 7.23.

Model Testing: Using the 5-gram model, I generated predictions for the test set, and reported the first 100 predictions in the JSON file named “results_student_model.json”, along with the perplexities. The perplexity of the 5-gram model on the test set is 4.18.

Training, Evaluation, and Testing on the Instructor-Provided Corpus: Finally, I repeated the process using the training corpus provided by Professor Mastropaolo, which I renamed to “teacher_data.txt” to match the corresponding JSON file, “results_teacher_model.json”, with the predictions and perplexities. The best-performing model corresponds to n = 3, with perplexity = 193,167.52 on the evaluation set, and perplexity = 99.72 on the test set.