GenAI for Software Development: Assignment 1

Gigi Kuffa
jkuffa@wm.edu

Kimberly Sejas
kmsejas@wm.edu

Eden Fitsum
efitsum@wm.edu

1 Introduction

In this assignment, we aim to implement an N-gram probabilistic language model to assist with code completion for Java systems by predicting the next token in a sequence based on the probabilities of token sequences. Specifically, we downloaded a list of Java repositories from the GitHub Search tool, tokenized the source code using the Javalang library, preprocessed the data, trained the model with an N-gram probability dataframe , and predicted code completions on incomplete code snippets. Finally, we evaluated model performance using perplexity values as an accuracy metric. The source code for our work can be found at https://github.com/jdkuffa/cs420-assignment1.

2 Implementation

2.1 Dataset Preparation

GitHub Repository Selection: We began by using the GitHub search tool (https://seaxrt-ghs.si.usi.ch/) to gather the Java repositories that met the following criteria: between 100 and 2000 commits, at least 50 stars, at least 15 contributors, licensed, and excluding forks. These constraints yielded approximately 3,967 repositories, which we sorted by last commits in descending order. From the CSV file downloaded from the site, we selected 30 repositories, extracted Java methods using the PyDriller framework from all files, and saved them to a CSV file for further cleaning and processing.

Cleaning: We focus on including only relevant Java methods by applying several cleaning criteria. Specifically, we removed comments, exact duplicates, and methods that were too long or too short. We also exclude boilerplate methods such as setters and getters, filtered out methods containing non-ASCII characters, and removed lines with unbalanced delimiters.

Dataset Splitting: Using a method, we split the student training TXT file into three separate files based on specified ratios (80-10-10).

Code Tokenization: We read in the student training TXT file, split the text based on whitespace, and collect the resulting tokens.

Vocabulary Generation: We generate a vocabulary that contains 482,212 code tokens. The vocabulary is generated considering the training, evaluation, and test set.

N-Gram Probability Dataframe: From the vocabulary, we built a N-gram probability dataframe to use for generating predictions. The next highest probable word is found by looking up the N-gram prefix in the dataframe.

Iterative Prediction: To complete the sentence, we recursively generated the next word. The length of the sentence is known so recursion stops when the length of the generated sentence equals the known sentence length.

Model Training & Evaluation: We train multiple N-gram models with context window sizes of 3, 5, and 7 to assess their performance. We calculate each model's perplexity, the lower indicating better performance. We select n = 3 as the best-performing model, as it achieves the lowest perplexity of 177.922397.

Model Testing: Using the selected tri-gram model, we randomly select 100 sentences from the test set. We generate the predictions and compute the perplexity of this test set. Our tri-gram model achieves a perplexity of 181.605017 on this dataset.

Training, Evaluation, and Testing on the Instructor-Provided Corpus: Finally, we repeat the training, evaluation, and testing process using the training corpus provided by Prof. Mastropaolo. In this case, the best-performing model corresponds to n = 3, yielding perplexity values of 711.431867 on the validation set and 586.708989 on the test set. As in the previous case, we conclude by generating predictions for the test set, following the same methodology.