GenAI for Software Development: Assignment 1
Ben Neifeld
bmneifeld@wm.edu

1 - Introduction

In this assignment we create an N-gram probabilistic model to determine the probability of a given token appearing next when N tokens are specified. The tokens are derived from a set of Java methods such that the model may aid software development tasks by recommending a certain token to appear. The model is trained on data derived from existing projects on GitHub, found using GitHub search, and the model is tested on the perplexity metric. The source code for the project can be found here: https://github.com/bneif/GenAI-for-SD-Project-1.

2 - Implementation

2.1 - Dataset Preparation

Github Repository Selection. All of the methods are extracted from one repository. We use the GitHub SEART tool (https://seart-ghs.si.usi.ch/) to mine well-developed java repositories, using the following filters: language = Java, minimum number of commits = 50, minimum number of stars = 100, minimum number of contributors = 5, and number of non-blank lines between 25,000 and 50,000. This yields 1,341 repositories, of which we randomly select 8 containing 544,388 methods.

Cleaning/Dataset Splitting. We use the Preprocessing_Code Jupyter notebook included in Lab 0 to clean the code by removing duplicates, filtering ASCII methods, removing outliers, removing boilerplate methods, and cleaning comments. After this process we are left with 61,613 methods, from which we randomly select 25,000 for our corpus. Of these 25,000 methods we randomly split 80% into training_student.txt, 10% into eval.txt, and 10% into test.txt, tokenizing each method along the way.

Code Tokenization/Vocabulary Generation. The above notebook uses Pygments for tokenization. We generate a vocabulary at runtime that contains the eval, test, and either student or instructor training set, plus two special tokens: <s> and <e> to denote the start and end of methods (they do not appear in the corpus otherwise).

2.2 - Training, Testing, and Evaluation

Model Training and Evaluation. We train n-gram models with context window 3, 5, 7, and 9. We use perplexity of the evaluation set as a metric, where lower perplexity is better. The best model was n=9 with perplexity 1.1363982, so we choose the n=9 model. All of the models are saved as pickles. ChatGPT was used to aid implementing my own ideas and to help implement file reading and writing.

Model Testing. We report in a JSON file predictions for 100 of the methods in the test set after the first 9 tokens, as well as the probabilities for those tokens. The perplexity over those methods is 1.1398532.

Training, Evaluation, and Testing on the Instructor-Provided Corpus. We repeat this process using the instructor's data to train the model on a corpus that is combined with the tokens from our own eval and test sets. The best instructor model was n=9 with perplexity 1.10709379 on the evaluation set and 1.1398532 on the test set.