GenAI for Software Development: Assignment 1

Lynelle Chen
lschen01@wm.edu

Benjamin Tremblay
bptremblay@wm.edu

Rowan Miller
rvmiller@wm.edu

1 Introduction

We have developed a model to automatically complete a Java method given a starting set of code tokens. Our model uses the N-gram method to predict the next token in a sequence given (N - 1) preceding tokens. It is trained on a corpus of public GitHub repositories and stores the frequency of all sequences of N consecutive tokens. When predicting the next token in a sequence, it chooses the one with the highest probability in the training corpus. Our model was evaluated by being asked to automatically complete a method given only its first (N - 1) tokens. The project can be found in the following GitHub repository: https://github.com/LynSChen04/n-grams.git.

2 Implementation

2.1 Dataset Preparation

Data Collection: We gathered data from public GitHub repositories that were tagged as containing Java code using GitHub Search for a csv of matching repositories and PyDriller to then aggregate Java classes and methods in each repository's commits.

Cleaning: We filtered out methods that were duplicates, overly long or short, contained non-ASCII characters, or were boilerplate methods such as a get() or set(). We then used the Python Pygments library to remove comments from the code.

Code Tokenization: We then tokenized the code with the Pygments JavaLexer and stored the tokens in comma-separated value files. Each repository was represented by one file, and each file contained a row for every method in the repository.

Dataset Splitting: We split the data 80% to be training data and 20% reserved for testing, with 100 methods reserved from the testing set for end evaluation.

2.2 Producing the Model

Model Generation: To build our n-gram model, we iterate through each file in the training data using the os library. Using the Natural Language Processing Toolkit library, we create a dict[] of all n-grams in our training data and their respective frequencies.

Optimize Value of N: To determine the best value of N for our model, we built 8 models for values of N from 1 to 8. The following table shows perplexity scores for models with different values of N:

N	1	2	3	4	5	6	7	8
Perplexity	17623	249611	2082707	10257401	33606081	74988767	122124067	173242481

Model Evaluation: We randomly reserved 100 Java methods from our corpus of Github data to use as a final test for our models. We determined N = 2 to be the best-optimized value for N, as although it had a higher complexity, using the unigram models to predict the next token resulted in meaningless loops of tokens that never truly ended the method.