CSCI 680 AI for Software Engineering: Assignment Report

Afia Farjana
afia.farjana@wm.edu

1. Introduction

This project aims to enhance code completion systems for Python by predicting the next token in Python method code. Leveraging an N-gram model, this approach predicts the likelihood of each possible next token based on the occurrences of token sequences in the training data. The predictive model approximates the probability of a token T_n given the preceding sequence of tokens, this helps make coding faster and more accurate.

2. Methodology

2.1 Dataset Preparation

Repository Selection

Initially, the SEART GHS tool was employed to curate a diverse set of Python repositories that were selected based on stringent criteria to include those with at least 500 stars, 500 commits, 500 issues, and 50 watchers, and excluding any duplicates, resulting in a collection of 3000 python repositories. From this set, 100 repositories were chosen at random for deeper analysis.

Repository Cloning and Data Extraction Process

A CSV file containing key details about each project has been created to organize this large dataset. Each link has been verified to ensure they are direct Git links, critical for the accuracy of the dataset and shallow cloning. After that, selected repositories were cloned using shallow cloning to manage data volume efficiently. Python files within these repositories were specifically targeted. Data extraction was implemented using an abstract syntax tree function in Update_method_extractor.py. The collected data encompassed various attributes such as project name, method start, and end lines, the method code itself, its documentation, and a combined field of method with documentation. This meticulously structured data was then compiled into a CSV file, tailored to facilitate detailed analysis and processing. The final step to prepare the dataset for use in the Python script has been converted from CSV format into a JSON file named Dataset_methods.json. This dataset holds 32k Python method codes.

Tokenization

Before tokenization, non-essential content from the dataset such as comments and whitespace was removed to prepare the data for processing. Python's tokenizer module (**RegexpTokenizer**), is used for extracting individual tokens from the method code, preparing them for the N-gram model training. The tokenize function in the script has utilized regular expressions to split method code into tokens, which are essentially the smallest units of text, like words and symbols.

Dataset Splitting

The dataset was divided into training, validation, and test sets, ensuring a robust model that generalizes well on unseen data. The division was stratified to maintain the distribution of code complexities and lengths across all datasets. **Training Set:** Comprises the majority of the dataset with entries up to the 31,700th.**Validation Set:** Consists of the subsequent 100 entries.**Test Set:** Contains the final 100 entries.

2.2 N-Gram Model: Vocabulary Generation

Functionality: By sliding a window across the sequences of tokens in the training data, the function generates both N-length and (N-1)-length tuples. It then stores these sequences in dictionaries to track the frequency of each sequence, which forms the basis of the N-gram model's vocabulary. This mechanism allows the model to learn the context of token sequences and predict the next token based on this learned information.

Model Setup: Models ranging from 2-gram to 7-gram were trained to determine which provided the best predictive accuracy. The variety allows the evaluation of model performance across different context window sizes.

2.3 Model Performance

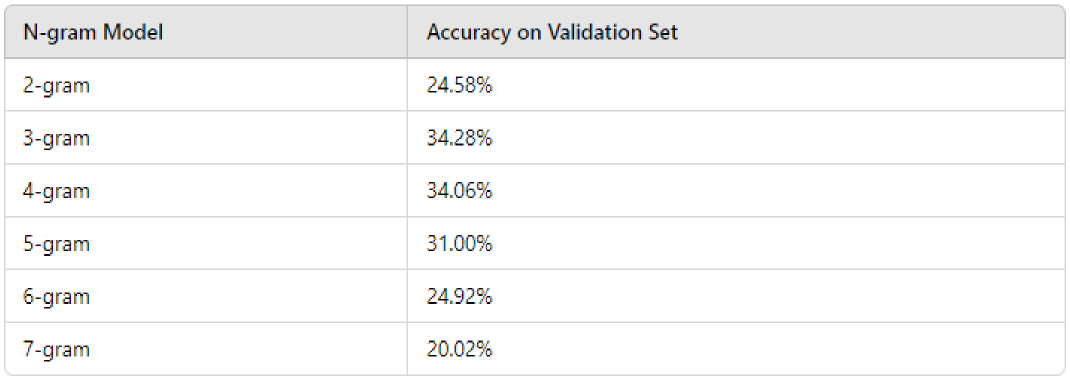
To predict the next token, the model uses the frequencies of token sequences stored in the dictionaries. It selects the token that most frequently follows a given (N-1)-length sequence, maximizing the likelihood of the predictive sequence.

3. Results

The effectiveness of the N-gram model was assessed by its ability to predict the next token in Python methods across various N-gram model sizes. The results demonstrated a clear relationship between the size of the n-gram model and the accuracy of the model.

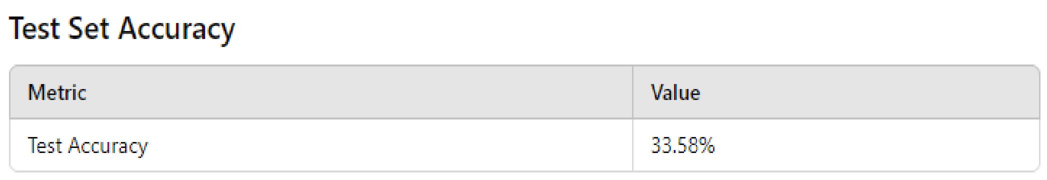
The highest accuracy achieved was with the 3-gram model, which attained an accuracy of approximately 34.28% on the validation set. This indicates the model's effectiveness in utilizing the context provided by the two preceding tokens to predict the next token accurately. This table clearly shows how the accuracy changes with different n-grams for this specific dataset and task. The results highlight that a 3-gram model strikes an effective balance between context awareness and computational efficiency.

Table 1: Summarizing the accuracy results from the output_results5.txt for different n-gram models on the validation set



N-gram Model	Accuracy on Validation Set
2-gram	24.58%
3-gram	34.28%
4-gram	34.06%
5-gram	31.00%
6-gram	24.92%
7-gram	20.02%

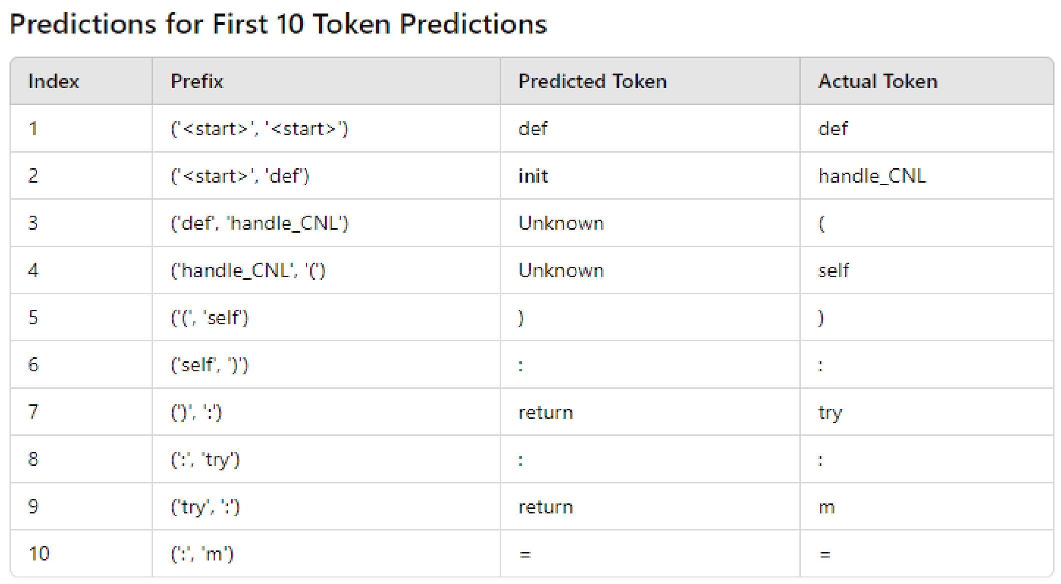
Table 2: Summarizing the test set accuracy



Test Set Accuracy

Metric	Value
Test Accuracy	33.58%

Table 3: Capturing the first 10 token predictions from the test set:



Predictions for First 10 Token Predictions

Index	Prefix	Predicted Token	Actual Token
1	('<start>', '<start>')	def	def
2	('<start>', 'def')	**init**	handle_CNL
3	('def', 'handle_CNL')	Unknown	(
4	('handle_CNL', '(')	Unknown	self
5	('(', 'self')	)	)
6	('self', ')')	:	:
7	(')', ':')	return	try
8	(':', 'try')	:	:
9	('try', ':')	return	m
10	(':', 'm')	=	=

This table reflects the predictions made by the model against the actual tokens that appear in the Python methods from the test set. It illustrates both the model's accuracy and its areas for potential improvement in context-sensitive predictions.

Reference

1. Hamarashid, Hozan K., Soran A. Saeed, and Tarik A. Rashid. "Next word prediction based on the N-gram model for Kurdish Sorani and Kurmanji." *Neural Computing and Applications* 33.9 (2021): 4547-4566.
2. Fink, Gernot A., and Gernot A. Fink. "n-Gram Models." *Markov Models for Pattern Recognition: From Theory to Applications* (2014): 107-127.
3. Qi, Weizhen, et al. "Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training." *arXiv preprint arXiv:2001.04063* (2020).