Assignment 1 Report

Introduction

This report presents a detailed technical review of my Python script main3.py, designed to create, evaluate, and manage n-gram models for natural language processing tasks. This script utilizes Python's libraries and custom functions to implement a text-processing application that predicts the next token in sequences of words from a given dataset.

Script Overview and Configuration

The script leverages several Python libraries such as pickle, sys, logging, re, and JSON, along with the defaultdict and Counter from the collections module to support its functionalities. It is structured to include error handling, data loading, model building, model evaluation, and result reporting.

Logging Configuration: To manage the script's output and debug information efficiently, the logging is configured to the INFO level. This configuration filters out less critical messages, and ensures that only essential information is logged, thereby maintaining clear and concise log outputs.

Basic Workflow

Data Preparation: It starts by loading the prepared dataset from dataset.json. Then the tokenization process. The next process begins by loading the data and splitting it into training, validation, and test sets.

Model Training: Different n-gram models have been trained using segments of the dataset designated as the training set.

Model Selection and Validation: Each model's performance has been evaluated on a validation set, and the best-performing model has been identified and saved for further analysis.

Testing and Reporting: The selected model has then been tested on a predefined test set, and detailed results have been documented both in the console and an output file for comprehensive evaluation and future reference.

1. Collecting the Dataset

The process began with the SEART GHS platform, which has been used to gather a significant pool of 3000 Python projects. 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.

From the extensive list, a focused group of 100 repositories has been selected for deeper analysis. Due to space limitations, the dataset was refined to include just six Python projects. For these

projects, method code and corresponding documentation were extracted. 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.json. This dataset holds 32k Python method codes with documentation.

2. Tokenization and N-Gram Model Explanation

Tokenization: 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. This tokenization is crucial as it transforms raw text into a structured format that the n-gram model can utilize for learning and prediction.

N-Gram Model: An n-gram model predicts the occurrence of a word based on the occurrences of its previous N−1N-1N−1 words in the sequence, making it a type of probabilistic language model. For instance, a 3-gram model predicts the next word based on the previous two words. This model has been used extensively in language processing to generate text that resembles human language. In main3.py, the build_ngram_model function constructs these models by creating a dictionary where each key is a sequence of N−1N-1N−1 tokens, and the value is a Counter object that tracks how often each subsequent token occurs.

3. Detailed Function Descriptions

Data Loading (load_data): The load_data function is designed to load and parse JSON formatted data. It includes robust error handling to manage JSON decoding errors and file-not-found exceptions, logging errors appropriately without stopping the execution flow.

Model Building (build_ngram_model): This function has been created for the n-gram models by iterating over the tokenized text. It builds a comprehensive dictionary mapping token sequences to their subsequent tokens, which helped to predict the next token in new sequences effectively.

Model Evaluation (evaluate_model_single_token): In the evaluate_model_single_token function, the script assesses the model's accuracy by comparing predicted tokens against actual tokens in the test set. This function also logs the first ten predictions for detailed inspection, facilitating a straightforward evaluation of the model's performance.

Prediction (predict_next_token): This function is implemented to fetch the most likely next token for a given prefix from the model, showcasing the application of probabilistic language models in predictive typing or automated suggestions.

Saving the Model (save_model): The best model is serialized using the pickle module and saved to a file for future use. Detailed results, including prediction accuracy and individual predictions, are written to both the console and an output file, ensuring that the results are documented comprehensively for further analysis or reporting.

Conclusion

The main3.py script demonstrates an advanced application of natural language processing techniques in Python, with a well-structured and efficient workflow that incorporates robust error handling, data processing, and model evaluation strategies. The careful design and implementation facilitate ease of use and extendibility, making it a valuable tool for tasks requiring predictive text capabilities.

1. **Insufficient Training Data**: Your model might not have encountered enough variation in the training data to be able to generate meaningful completions for the test set inputs.
2. **Training-Test Distribution Mismatch**: If the training data is very different from the test set in terms of token distributions (e.g., different function types, variable names, etc.), the model may struggle to generalize to the test set.
3. **N-gram Limitation**: The N-gram model may have inherent limitations when it comes to learning long-term dependencies and generating meaningful sequences. N-gram models are limited to their immediate local context (in this case, 10 tokens), and they do not have the capability to "understand" or "remember" larger structural patterns in code the way more advanced models like RNNs or Transformers can.