Project-1

CSCE 5290-Natural Language Processing

Yaswanth Sai Satish Sreerama-11601948

Introduction:

In this project, initially create a virtual environment with specified requirements and later open the unigramLg.ipynb file and process the input text file to convert them into bigrams, trigrams and 4 grams and calculate the perplexity score and diversity score and plot them.

Definitions and Function used for N-Grams:

N-gram models are used in natural language processing to predict the probability of the next word in a sentence based on the previous n-words. In the above code, bigram, trigram, and 4-gram models have been used to generate sentences from a text corpus. The bigram model considers the probability of the next word in a sentence based on the previous word, the trigram model considers the probability of the next word based on the previous two words, and the 4-gram model considers the probability of the next word based on the previous three words.

The **nltk. ngrams()** function is used in the code to generate bigrams, trigrams, and 4-grams. The function takes two arguments: the input sequence of tokens (words) and the value of n (2 for bigram, 3 for trigram, and 4 for 4-gram). The function returns an iterator that generates tuples of n consecutive tokens. The generated tuples are used to create frequency distributions and compute the perplexity and diversity scores for each n-gram model.

Implementation:

1. Open Anaconda Prompt or a terminal window.
2. Create a new virtual environment named "venv".
3. Activate the virtual environment.
4. Install requirements specified in the requirements.txt
5. Then open the Jupyter notebook/jupyter lab.
6. Open unigramsLg.ipynb.
7. Now import text file given as the corpus and then use the nltk. ngrams generate bigrams, trigrams and four grams.
8. Then generate 5 sentences of 10 words each using bigrams, trigrams and four grams using frequency distribution for selection of words in the sentences.

9. Calculate the average sentence length and number of unique sentences for all the sentences.
10. Calculate perplexity and diversity score for the all the models.
11. Plot the graphs for the perplexity and diversity scores.

Analysis:

Comparison of the n-gram models:

To compare the performance of the three n-gram models, perplexity and diversity scores were computed for each model. The perplexity score measures how well the n-gram model can predict the next word in the sequence, and the diversity score measures the number of unique n-grams in the model.

The perplexity and diversity scores were calculated over different sizes of the training set. The perplexity score decreases, and the diversity score increases with the size of the training set for all three models. This indicates that the models are improving with more data but are limited by the size and specificity of the text corpus. The bigram model has the highest perplexity score and the lowest diversity score, which suggests that it is the least effective model for generating coherent and diverse sentences. The trigram model has a lower perplexity score and a higher diversity score than the bigram model, indicating that it performs better than the bigram model. The 4-gram model has the lowest perplexity score and the highest diversity score, suggesting that it is the most effective model for generating coherent and diverse sentences.

Conclusion:

In conclusion, the bigram, trigram, and 4-gram models have been used to generate sentences from a text corpus of papers related to blockchain technology. The generated sentences show some coherence and meaningfulness, but the quality of the output is limited model used for sentence generation. The perplexity and diversity scores indicate that the 4-gram model is the most effective model for generating coherent and diverse sentences.