CS771: Introduction to Machine Learning

Assignment - 2

Question 1:

In this problem, we first generate bigrams from words using the ***generate_bigrams*** function. Each bigram represents a sequential pair of characters within a word. For example, the bigrams for the word **noise** are (**no**, **oi**, **is**, **se**).

Next, we sort our bigrams in ascending alphabetical order. If the first character matches, we compare the second character. We implemented ***sort_bigrams*** to sort and retrieve the **5 smallest** lexicographically sorted bigrams for **each word**. We used a global array ("array") to efficiently mark the presence of each bigram across multiple words.

We developed ***process_words*** to create an unordered map (*bigram_map*) where:

- **Keys** are tuples of sorted bigrams
- **Values** are lists of words containing those bigrams, limited to 5 words per key to **optimize memory usage and retrieval efficiency**

The next task is to train our model and ensure proper accuracy. We used ***my_fit*** to process a dictionary of words, creating *bigram_map* as the trained model. The model was visualized using ***print_model*** to show the mapping of sorted bigrams to their associated words.

For prediction, we used ***my_predict*** to predict words based on a given list of bigrams. We ensured robustness by **returning an empty list** if the bigrams do **not match any keys in *bigram_map***.

Several factors are involved in ***design decisions*** such as:

1. **Choice of Data Structure:** We opted for an unordered map (dictionary) to efficiently store and retrieve mappings of bigrams to words. This provides fast average-time complexity for insertions and lookups due to hash-based indexing.
2. **Handling of Bigrams:** We calculated the size and storage efficiency considerations of using bigrams (676 possible combinations for two characters in the English alphabet).
3. **Memory Efficiency:** We detailed the use of a global array to minimize memory allocation by reusing it across multiple function calls in *sort_bigrams*.
4. **Scalability and Performance:** We considered scalability with large datasets, including potential limitations and optimizations made to handle such scenarios efficiently. Discussions on runtime complexities for operations like generating bigrams, sorting, and mapping were included.
5. **Hyperparameters:** We defined and discussed the hyperparameters used in our approach (e.g., *lim_bg* for limiting bigrams per word, *lim_out* for limiting output guesses).
6. **Evaluation and Metrics:** We described how to evaluate the effectiveness of the model using metrics like precision, recall, or accuracy in a real-world scenario. We highlighted how the limited number of guesses (*lim_out*) affects the model's performance and trade-offs.

Conclusion

We discussed all key points and the approach of using an unordered map as an alternative to traditional decision tree just because in backend an unordered map works on a red black tree model and based on the motivation of applying things so that our code works we have done this for tasks involving feature extraction and prediction based on sequential data (bigrams). This method emphasizes efficiency, simplicity, and scalability compared to more complex models.

Question 2:

Zipped ***Solution*** *to Assignment 2*