Answer

1. Question #1

a) **What do you consider a token for this task and why (i.e., do you include punctuation marks as well?)? What kind of preprocessing steps, if any, do you need to apply before you feed the data into your language model?**

   In this model, the lowercase alphabetic characters was only considered as tokens. Punctuation and other symbols are removed. A start token ("^") and an end token ("$") were added to each sentence so that the model learns which letters typically begin or end a sentence. This choice focuses the model on the letter patterns and avoids noise from punctuation.

b) **What technique do you decide to use for out of vocabulary (OOV) words and why?**

   In this model, add-one smoothing technique was used to handle OOV words. With this approach, even if a particular bigram was never seen in the training data for a language, it is assigned a non-zero probability.

c) **Can the letter bigram model be implemented without any kind of smoothing? If not, use add-one smoothing. Is this kind of smoothing appropriate or do you need better algorithms? Why (not)?**

   A letter bigram model without smoothing will give zero probability to any test sentence containing an unseen bigram. This is why add-one smoothing is implemented. While add-one smoothing is not always ideal for all language modeling tasks because it can overestimate the probabilities for unseen events, it is acceptable for this task.

Compare your output file with the solution file (labels.sol found in the src/Data/Validation/folder). How many times was your program correct?

Accuracy: 98.33%

2. Question #2

a) **What do you consider as a word for this task and why (i.e., only alpha-numeric characters, or do you want to use punctuation marks as well as valid word tokens)? What kind of preprocessing steps, if any, do you need to apply before you feed the data into your language model?**

   In this model, a word is defined as any contiguous sequence of letters and digits. Punctuation and other symbols are removed. The tokens are converted to lowercase. '<s>' and '</s>' are added to mark start and end.

b) **What technique do you decide to use for out of vocabulary (OOV) words and why?**

   In this model, add-one smoothing technique was used to handle OOV words, ensuring that unseen bigrams receive a non-zero probability.

c) **Can the word bigram model be implemented without any kind of smoothing? If not, try add-one smoothing. Is this kind of smoothing appropriate or do you need better algorithms? Why (not)?**

   Without smoothing, a single unseen bigram would yield a probability of zero. Add-one smoothing is applied to avoid this issue. While add-one smoothing is not always ideal for all language modeling tasks because it can overestimate the probabilities for unseen events, it is acceptable for this task.

Apply the models to determine the language for each sentence in the test file. Compare your output file with the solution file provided in the src/Data/Validation/folder(labels.sol). How many times was your program correct?

Accuracy: 99.33%

3. Question #3

What do you do when the number of words seen once are unreliable? What strategy do you use to smooth unseen words?

This model uses Good-Turing smoothing to adjust the counts of rare bigrams. Specifically, it applies Good-Turing smoothing only to bigrams with frequencies below 5. For bigrams that never appeared in the training data, it assigns a small probability based on the number of bigrams that appeared only once. If the frequency information for a particular context is missing or seemed unreliable, the model falls back to Add-One smoothing, ensuring that every bigram receives at least some non-zero probability.

Accuracy: 99.00%

Which of the language models at Question Sets #1, #2, and #3 is the best? Comment on advantages and disadvantages of these language models on the task (be as detailed as possible based on your observations).

The word bigram model with add-one smoothing had the best accuracy among three models.

The letter bigram model is simple and computationally efficient, and it benefits from a relatively low level of sparsity because it operates at the character level. However, because it only considers individual letters, it may miss higher-level linguistic structures, making it less sensitive to differences that are critical for distinguishing.

The word bigram model with add-one smoothing leverages richer contextual information by operating on word-level tokens. This allows it to capture meaningful patterns in language usage and word order, resulting in the highest accuracy (99.33%). On the downside, although add-one smoothing helps by ensuring every bigram has a nonzero probability, it can sometimes overestimate the likelihood of rare events.

The word bigram model with Good-Turing smoothing attempts to address the shortcomings of simple add-one smoothing by more accurately adjusting the probabilities of rare and unseen bigrams. This method utilizes frequency-of-frequency information to better estimate the likelihood of low-frequency events. However, this model is complicated and computationally expensive. Also, when the counts of words seen only once are unreliable, the model must fall back to add-one smoothing, which adds complexity and dependency on the quality of frequency data.