Zeynep Marasli

1) Question Set #1. Letter Bigram Model

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?

- A token for this task is single characters within the data, including letters, punctuation, whitespaces, and the beginning & end of sentence markers (<s>, </s>).
- I chose to define my tokens this way because this would take into consideration the input at the letter level. I chose to also include whitespaces (to delineate word boundaries) and certain punctuation marks such as “?”, “,”, “!”, ““, “:”, “;” that provide extra meaning to the input. I also chose to include letters that may not be present in the target language’s alphabet (ex. à is not a letter in English) for consistency across all 3 languages.
- For preprocessing steps, I converted the input to all lowercase, removed numbers, and also removed any double spaces. Additionally, since each line is a sentence, I added beginning and end of sentence markers to each beginning and end of each line, respectively.

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

- After preprocessing and tokenizing the data, I created a temporary frequency dictionary that kept track of the frequencies for each token type to identify tokens with a frequency less than or equal to a predetermined frequency (UNK_threshold variable in my code). These low frequency tokens are then replaced with the <UNK> token in the dataset. This modified dataset is then used to generate the bigrams.
- During testing, the test string is preprocessed & tokenized, and any tokens that are not present in the training data set are replaced with the <UNK> token prior to bigram generation.
- I chose to implement this technique for OOV words after consulting the lecture notes and the textbook.

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)?

- Without any smoothing, this letter bigram model can be implemented. Because this model tokenizes at the character level, there are 48-56 unigrams for each language, 2304 - 3136 total possible bigrams, and around 30% of those bigrams occurred at least once in the dataset. This is not as sparse as the Word Bigram Model. Without smoothing, the program was able to identify 96% of the test sentences correctly.
- With Laplace smoothing, the program’s performance increased to 99% accuracy. While more advanced smoothing techniques may provide even higher accuracy, 99% accuracy seems sufficient for this model.

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

Without smoothing, the program is correct 96% of the time. With smoothing, the program is correct 99% of the time.

2) Question Set #2. Word Bigram Model.

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?

- For this model, I considered a token as a word, a punctuation mark, and beginning and end of sentence markers. This allows the model to consider the input at the word level but also words that tend to be at the beginning or end of sentences. As with the letter bigram model, I chose to include certain punctuation marks such as ".", "?", ",", "!", """, ":", ";" that provide extra meaning to the input.
- For preprocessing steps, I converted the input to all lowercase, removed numbers, and also removed any double spaces. Additionally, since each line is a sentence, I added beginning and end of sentence markers to each beginning and end of each line, respectively.

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

- I used the same technique I used for the Letter Bigram Model.

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)?

- Yes, this model can be implemented without any kind of smoothing. Out of 7011904 - 8850625 possible bigrams for each language, around 0.34% of those bigrams actually occurred in the dataset. This means that the large portion of the bigram frequency table is 0.
- Without smoothing, my program was able to identify 16% of the sentences in the test set correctly. Unlike the Letter Bigram Model, this model did not benefit from Laplace smoothing, which plummeted it's performance to an abysmal 1%. I'm not exactly sure why add-one smoothing worsened the model's performance, but I assume it may be because it naively shifts the frequency distribution to the unseen bigrams, some of which are simply never going to show up for a certain language.
- After playing around with the hyperparameters, I was able to increase the program's performance to 39.6% by (1) not implementing smoothing, and (2) readjusting my UNK_threshold to 2, so that only tokens at occur at most twice are replaced by the UNK token.
- Because of add-one smoothing's poor performance, more advanced smoothing methods are certainly needed.

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

- The highest performance I could achieve from this model is 39.6%.

3) Question Set #3. Word Bigram Model with Good-Turing Smoothing

Same as Question#2, point c), but replace the add-one smoothing with Good-Turing smoothing. What do you do when the number of words seen once are unreliable? What strategy do you use to smooth unseen words?

Since MLE is more reliable for bigrams at higher frequencies, I decided to only use GT probabilities for bigrams that occurred at most 10 times. To calculate the probability of a sentence in the test set, I followed the procedure outline for the first solution for GT smoothing in the notes, in which conditional probabilities were calculated for each bigram and summed together. With GT smoothing implemented, I was able to increase the language model's performance to 99%.

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).

In terms of performance, the language models in Set #1 and Set #3 are equivalent. Due to the vast sparsity of the word bigram count matrix, no smoothing/simple add-one smoothing is not appropriate for a Word Bigram Model, but seems to be sufficient for Letter Bigram Model.

In terms of implementation, simplicity, and parameter tuning, I personally found Set #1's language model easier to implement and required less time hyperparameter tuning.

For smaller vocabularies, simpler smoothing methods like add-one smoothing are sufficient, however for models with larger vocabularies (and thus larger a ngram matrix / greater parameters), more advanced smoothing techniques seem to be necessary. Thus, I think the language model in Set #3 is the best because it can handle a larger number of parameters.