REPORT

Name: Raunakk Banerjee

Roll: 180121032

Link to code:
https://colab.research.google.com/drive/1JHYmBDN5EBh1iVViw5luYGFPwncZzsPG?usp=sharing

I reduced the size of the corpus by taking only the first 100094 characters of the corpus. This was done because the google collab session was crashing when working with the entire corpus.

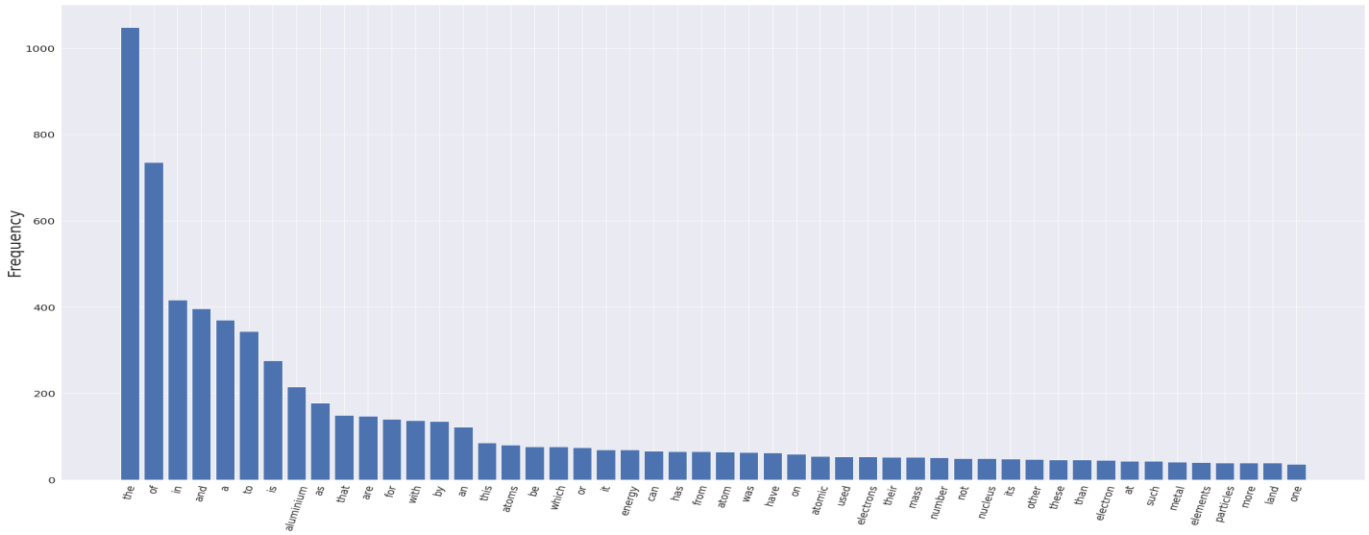
All the results are displayed in the notebook with print statements. To view the results of tasks like byte pair encoding, sentence segmentation etc., please run the notebook.

Q1.3.1)

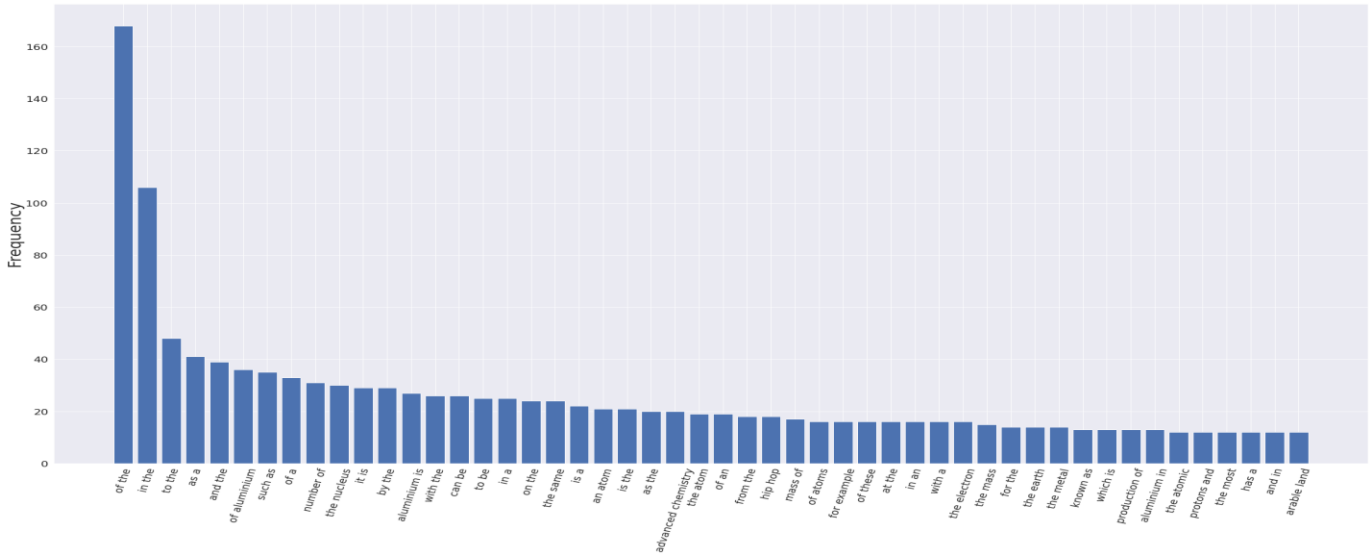
1. Both word tokenization and sentence segmentation can be done using NLTK and SpaCy.
2. Both did a wonderful job. I could not find any difference between the results obtained using both the libraries
3.
    - The most common words like the, in, as appeared more often than others as expected. It was not possible to get much context from the unigrams and bigrams and trigrams showed combinations like “Advanced chemistry” and “protons and neutrons”. It is interesting how “aluminium” turned out to be a very common unigram.
    - The frequency distribution of the bigrams and trigrams were more like blocks as there were many with same frequencies.
    - The unigrams followed Zipf law pretty well but the bigrams and trigrams fared poorly due to multple grams having the same frequency.

Frequency distribution plots:

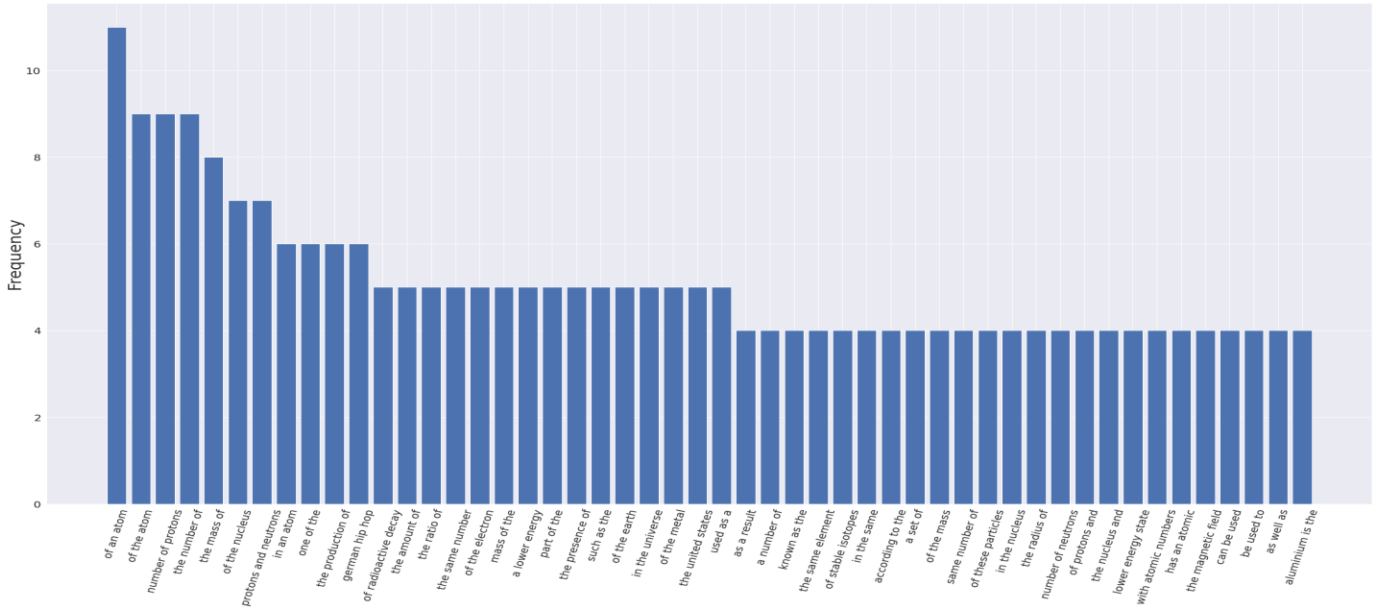
For Unigrams,



For bigrams,

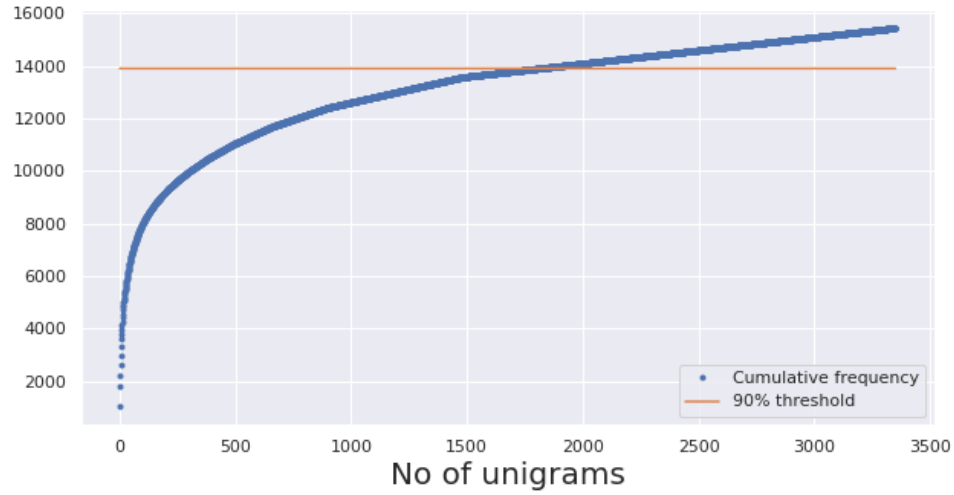


For trigrams,

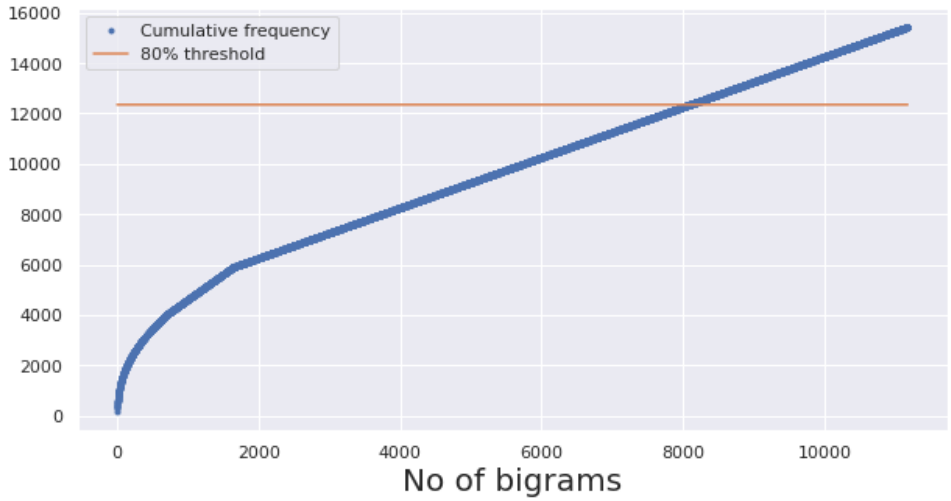


Q1.3.2)

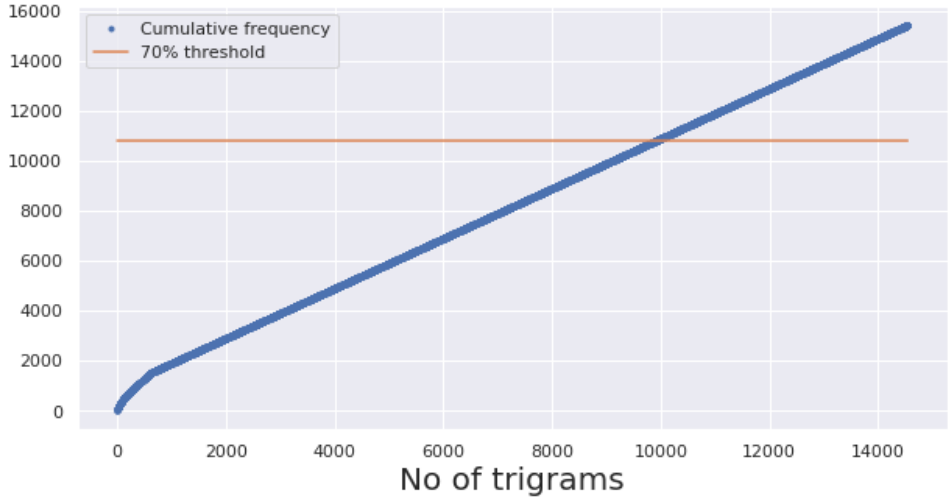
1. The figure below show the cumulative frequency for unigrams and the 90% threshold



2. The figure below shows the cumulative frequency for bigrams and 80% threshold.



3. The figures below shows the cumulative frequency for trigrams and 70% threshold.



4. The left out tokens generally contain morphologically complex words that occur rarely even in natural language. Words with more than two morphological components usually occur once and can clubbed in the OOV type.
5. So, based the observed frequencies. Words with frequency >= 5 can be one type, words with frequency >=2 but <=5 can be the next type and the words with frequency <2 can be the OOV type .

Q1.3.3)

For sentence segmentation,

1. Due numerous newline characters, I first removed all of them.
2. Then I checked if the end of the corpus had been reached. If so, then that should be the end of the sentence as well.
3. Next, I checked for full stops. As decimal numbers also have a fullstop to indicate the decimal, I also checked for a whitespace to immediately follow the fullstop. That would eliminate all decimal numbers.
4. Following that, I checked if the preceding the fullstop there were any words like Mr, Mrs, Jr, Ms. Also to eliminate cases like the fullstop of "J.J Thomson" being taken as a sentence boundary, I checked whether the character preceding a fullstop is uppercase.
5. Lastly I checked for a "?" and "!" as they too indicate an end of sentence.

For word tokenization,

1. I seperated the words by whitespace.
2. Due to this naïve approach, many punctuation marks or symbols were included in the words. To resolve this, I removed all commas, brackets, full stops and double quotes from the words.

Obviously, the heuristics applied by me were not exhaustive and thus did not produce results as good as the results obtained by using the NLTK or SpaCy libraries. There were some stray symbols left out in the word tokenization and some cases of faulty sentence segmentation in the results produced by the heuristics.

But, the results obtained by the heuristics were astonishingly good considering the complexity of the language.

Q1.3.4)

1. I used the morphological analyzer provided by SpaCy. SpaCy allows as to obtain several morphological components like the morpheme using **"token.lemma_**" and the part of speech using **"token.pos_"**. The obtained morphemes and part of speech tags were very accurate. Additional grammatical features such plural/singular can be obtained from the **"token.tag_".**

Q1.3.5)

I have tried to explain my algorithm for Byte Pair encoding by adding comments next to the code.

1. The first sub-words to be tokenized were common sub-words like “in”, “ing”, “the”.
2. The BPE for the words and their morphemes can be similar or different depending on if the word has a simpler morpheme. Simple words like “he”, “of”, “in” etc. do not have different morpheme so their byte pair encoding will be basically the same. But for “walking”, the morpheme would be “walk”, so the BPE would be different.
3. Another point to consider, is as the most frequent sub-word is encoded, the BPE of a word depends on other words and their morphemes as well.
4. In my code, most words donot have a simpler morpheme thus do not effect the BPE drastically but there are some differences in the way the words are encoded .