Natural language processing – 2

Mias Ghantous – 213461692

Faisal Omari - 325616894

In our work we programed a class that construct 3 dictionaries the first is a counter of unigrams (frequncy\_dictionary), the second is the bigram counter (frequency\_dictionary\_2\_words), and finally for the trigram (frequency\_dictionary\_3\_words).

We did all of this in the constructer witch take the type of the protocol as a string and also the constructer save the corpus size.

part 1.

Calculate\_prop\_of\_sentence:

1. We calculate the prop of the sentence with the help of the dictionaries that we constructed in the constructer.
2. For the linear smoothing to avoid the division by 0 we decided to use the "get" function to see if we have the key and if not put 1 in the denominator and 0 in the numerator if we are not calculating the unigram, and if we are calculating the unigram then do Laplace smoothing.
3. For the first and the second word we decided to include them to the probability by calculating the unigram and the bigram probability for the first and the second words, and calculate the trigram prop for all the other words.
4. If the sentence has less than 3 tokens then the previous note take care of it.
5. For the in the linear smoothing we decided that for the tri gram for the bigram for the unigram

generate\_next\_token:

first we take the last 2 words from the original sentence because these are the words that are going to affect the probability of the next token, and then we try all the possible tokens that appear in the corpus (we can access them from frequncy\_dictionary.keys() ) and use the previous function that we implemented to calculate the probability and return the token with the max probability.

Part 2.

get\_k\_n\_collocations:

we create a dictionary as a counter of the occurrences of a collocation (as a string), we read the right data from the corpus for each sentence in the corpus, get the tokens if we have less than n tokens then don’t do anything.

The first collocation is from 0 to n-1 increase the counter of the collocation and then go to the next collocation by deleting the first token and add the next token in the sentence do this until the sentence has no more tokens.

In the end use heapq to get the required.

For the print part of the section we did a function called Q2\_text, we made 2 modules one for committee and one for plenary and create the required text with the help of the 2 modules and the previous function.

Part 3

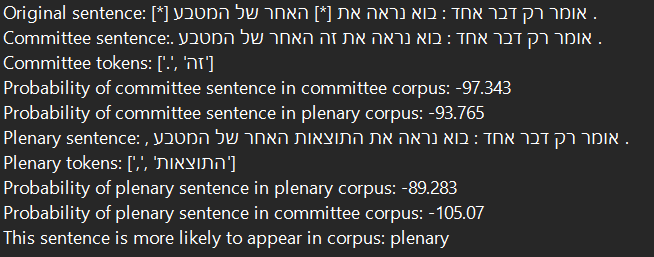
For this part we programed function called Q3\_text, we iterate throw every word in each sentence if the word is [\*] then calculate the next token for both types and complete the sentence regularly if it is a normal word, and at the end save the text in a txt file.

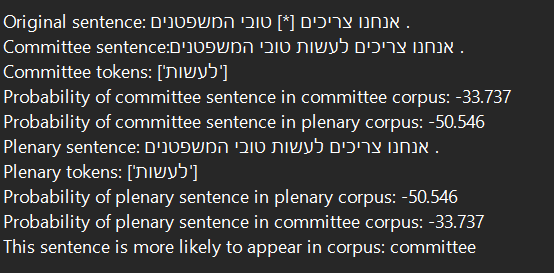
Part 4

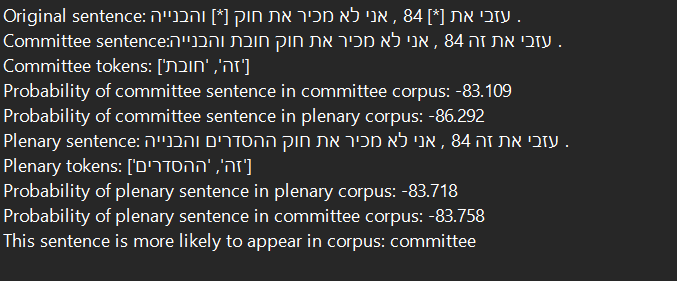
1. In our 2 modules sometimes we have same token predictions but for the most part we don’t have the same results, why? Because we have completely separate data and 2 different ways of talking, and that explain the difference.

but still the very common collocation like "את זה" of course both the models are going to predict the same thing "זה" and that’s why we have similarity.

1. The collections meet our expectations because we have a lot of punctuation marks and words that are used in day to day life like ""אני, אבל and appears that we have some words like "חבר, כנסת, היושב, ראש" that we may not use in the day to day life but they are famous because of the type of the corpus (Knesset corpus) so of course they are going to be famous collocations in our corpus.
2. We have a lot of fine predictions like:







But of course if we get a bigger corpus we are going to have a lot more fine predictions.

1. If we were to use Bigram module then we would have got worst results because then the module would be less knowledgeable on the relations between the words, which results in worst predictions.