NLP Assignment 2

Bhavya Narang (2019462)
Yash Aggarwal (2019480)

1. Assumptions :

   Some of the assumptions made during this assignment are as follows :
   1. We have treated every new line in the text as a separate sentence.
   2. We have assumed that the words given in the input text are properly tokenized.

2. Text Preprocessing :
   a. We have treated every new line as a separate sentence. For any sentence, we have tokenized using the word_tokenize function of nltk.
   b. To maintain the coherency of the sentence, we have deferred from using lemmatization or stemming as it will compromise the coherency of the text.
   c. We have not removed the stop words from the text.

3. Methodology :

   In this assignment, we had to implement 3 variations of the bigram model; without smoothing, Laplace smoothing, and add-k smoothing.
   a. We have first made the vocabulary, unigram count set and the bigram count set. We calculate the covariance matrix of the adjacent words and store the count of their occurrence in the entire corpus as a set. This is because computation on a very large 2-D matrix is very expensive.
   b. Without Smoothing : In this case, we first find the word just before the missing word. Now using the options given in the test

case, we calculate the probability for the occurrence of the word 'before' and the 'current' word using the bigram count and then check the unigram count for the word 'before'. If the count is 0, then we can say that the probability of occurrence of the given word is 0.

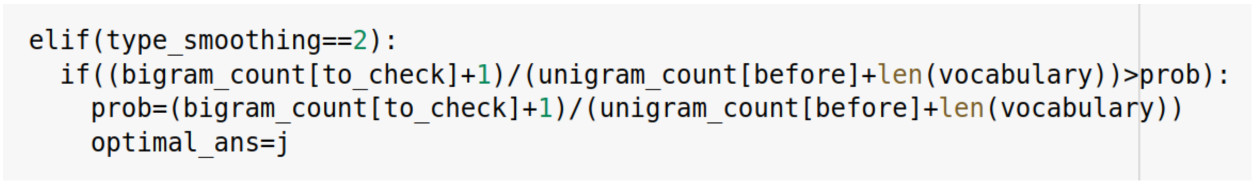
The formula for probability without smoothing used is :

P(xxxxx|word2)=bigram_count(word2, xxxxx)/unigram_count(word2)

```
if(type_smoothing==1 and (before in unigram_count)):
  if(bigram_count[to_check]/unigram_count[before]>prob):
    prob=bigram_count[to_check]/unigram_count[before]
    optimal_ans=j
```

c. With Laplace Smoothing : In this case, the calculations are similar, however, we add a one to the bigram count and the length of vocabulary to the count of the word before.

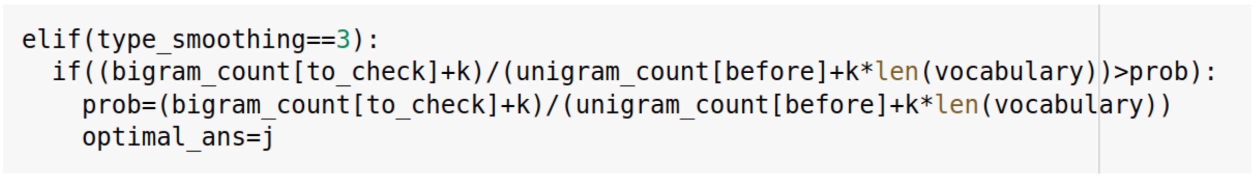
P(xxxxx|word2)=bigram_count(word2xxxxx)+1/unigram_count(word2)+V



```
elif(type_smoothing==2):
  if((bigram_count[to_check]+1)/(unigram_count[before]+len(vocabulary))>prob):
    prob=(bigram_count[to_check]+1)/(unigram_count[before]+len(vocabulary))
    optimal_ans=j
```

d. With add-k smoothing : In add k smoothing, we add a k to the bigram count and k X length of vocabulary to the count of the word before.

P(xxxxx|word2)=bigram_count(word2xxxxx)+k/unigram_count(word2)+k*V



```
elif(type_smoothing==3):
  if((bigram_count[to_check]+k)/(unigram_count[before]+k*len(vocabulary))>prob):
    prob=(bigram_count[to_check]+k)/(unigram_count[before]+k*len(vocabulary))
    optimal_ans=j
```

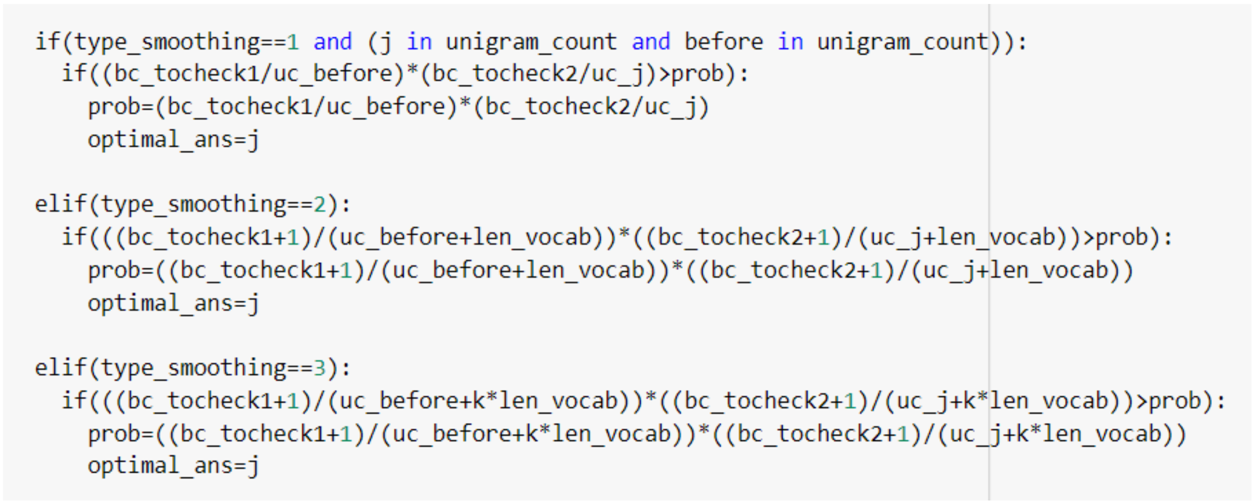
4. Bonus :

The main idea of using future context here is, for general next word prediction we have the availability of previous words only, that is, word1 word2 xxxxx , predicting xxxxx using the previous words and making N-gram models. But in the given question we have added information and hence we can make use of it and have a more certain prediction.

That is, given a question of type: word1 word2 xxxxx word3, we have made use of the bigrams (word2,xxxxx) and (xxxxx,word3) at the same time, hence leading to a better prediction.

P(word2,xxxxx)=bigram_count(word2 xxxxx)/unigram_count(word2)
P(xxxxx,word3)=bigram_count(xxxxx word3)/unigram_count(xxxxx)
And finally multiplying both these probabilities, as the phrase has to occur at the same time.

Code for the same is given below:



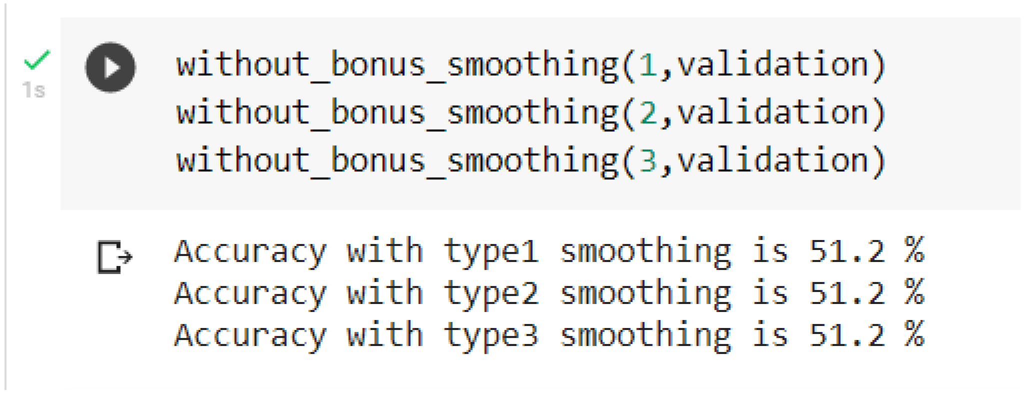
```
if(type_smoothing==1 and (j in unigram_count and before in unigram_count)):
  if((bc_tocheck1/uc_before)*(bc_tocheck2/uc_j)>prob):
    prob=(bc_tocheck1/uc_before)*(bc_tocheck2/uc_j)
    optimal_ans=j

elif(type_smoothing==2):
  if(((bc_tocheck1+1)/(uc_before+len_vocab))*((bc_tocheck2+1)/(uc_j+len_vocab))>prob):
    prob=((bc_tocheck1+1)/(uc_before+len_vocab))*((bc_tocheck2+1)/(uc_j+len_vocab))
    optimal_ans=j

elif(type_smoothing==3):
  if(((bc_tocheck1+1)/(uc_before+k*len_vocab))*((bc_tocheck2+1)/(uc_j+k*len_vocab))>prob):
    prob=((bc_tocheck1+1)/(uc_before+k*len_vocab))*((bc_tocheck2+1)/(uc_j+k*len_vocab))
    optimal_ans=j
```

5. Accuracies :

1. Without Smoothing : type1
2. Laplace Smoothing : type2
3. add-k Smoothing : type3

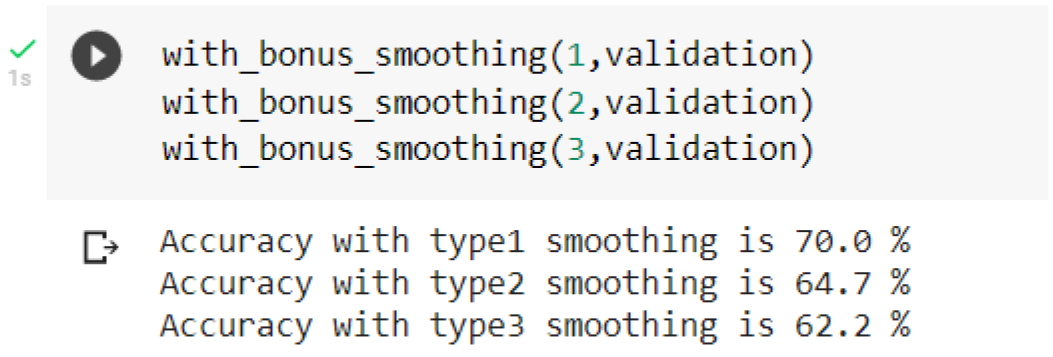


```
without_bonus_smoothing(1,validation)
without_bonus_smoothing(2,validation)
without_bonus_smoothing(3,validation)
```

```
Accuracy with type1 smoothing is 51.2 %
Accuracy with type2 smoothing is 51.2 %
Accuracy with type3 smoothing is 51.2 %
```

For Bonus,

1. Without Smoothing : type1
2. Laplace Smoothing : type2
3. add-k Smoothing : type3



```
with_bonus_smoothing(1,validation)
with_bonus_smoothing(2,validation)
with_bonus_smoothing(3,validation)
```

```
Accuracy with type1 smoothing is 70.0 %
Accuracy with type2 smoothing is 64.7 %
Accuracy with type3 smoothing is 62.2 %
```

Also, output text files for both the types are uploaded along with the predictions.