ELL881 NLP Assignment-1 Report

-Rajdeep Das
2019MT10718

- **Dataset Preprocessing**

Tokenization was done using the nltk library (word_tokenize and sent_tokenize). A high number of occurrences of lines containing page number and book title and author were selectively removed from the dataset as not only did they not provide any information about the language structure but they also cluttered the N-gram model frequencies (ex:- Page | 2 Harry Potter and the Philosophers Stone - J.K. Rowling). All the text was converted to lowercase. Furthermore, punctuation tokens of singular length (ex:- comma, exclamation mark, quotes, slash etc.) were removed from the token list as yet again they were taking up the highest frequencies of N-grams for all values of n.

1. Training Data:

Includes data from books 1 through 6.

Total sentences	51670
Total tokens	1155432
Total tokens after removing particular punctuation	920732
Total types/words in vocabulary	21079

2. Testing Data:

Includes data from book 7.

Total sentences	11138
Total tokens	256058
Total tokens after removing particular punctuation	204766
Total types/words in vocabulary	11449

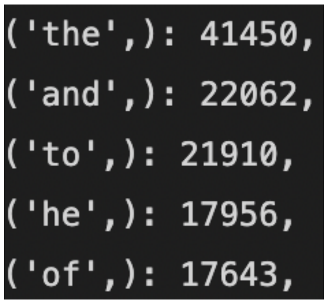
There are 2068 types in the test-set that are not in the training set will be considered as OOV (out-of-vocabulary) types.

- **N-Gram models**

Unigram Types	21079
Bigram Types	277884
Trigram Types	642415
4-gram Types	835723
5-gram Types	896438

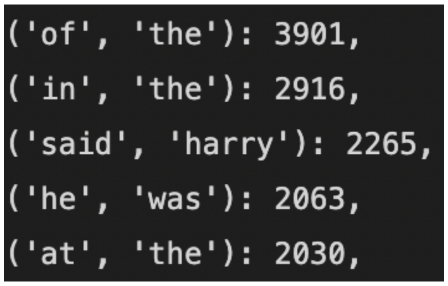
Top 5 types by count of all N-grams:

1. Unigram



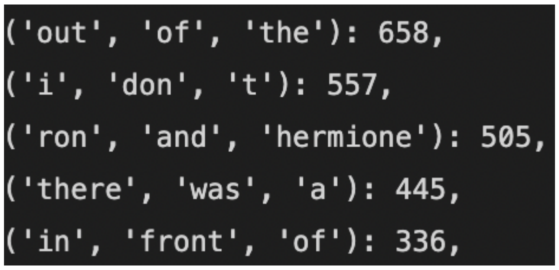
```
('the',): 41450,
('and',): 22062,
('to',): 21910,
('he',): 17956,
('of',): 17643,
```

2. Bigram



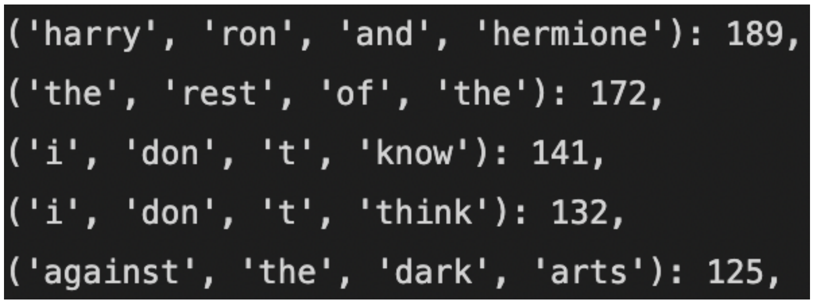
```
('of', 'the'): 3901,
('in', 'the'): 2916,
('said', 'harry'): 2265,
('he', 'was'): 2063,
('at', 'the'): 2030,
```

3. Trigram



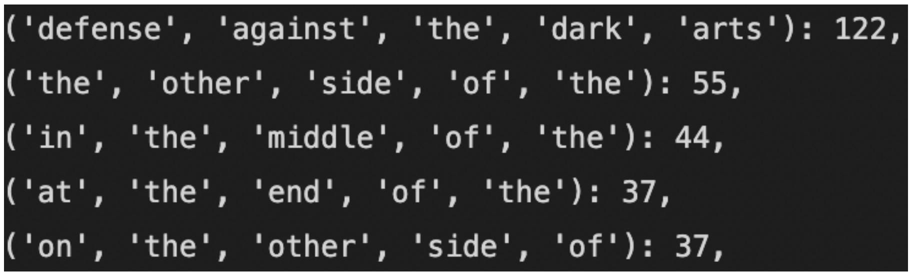
```
('out', 'of', 'the'): 658,
('i', 'don', 't'): 557,
('ron', 'and', 'hermione'): 505,
('there', 'was', 'a'): 445,
('in', 'front', 'of'): 336,
```

4. 4-gram



```
('harry', 'ron', 'and', 'hermione'): 189,
('the', 'rest', 'of', 'the'): 172,
('i', 'don', 't', 'know'): 141,
('i', 'don', 't', 'think'): 132,
('against', 'the', 'dark', 'arts'): 125,
```

5. 5-gram



```
('defense', 'against', 'the', 'dark', 'arts'): 122,
('the', 'other', 'side', 'of', 'the'): 55,
('in', 'the', 'middle', 'of', 'the'): 44,
('at', 'the', 'end', 'of', 'the'): 37,
('on', 'the', 'other', 'side', 'of'): 37,
```

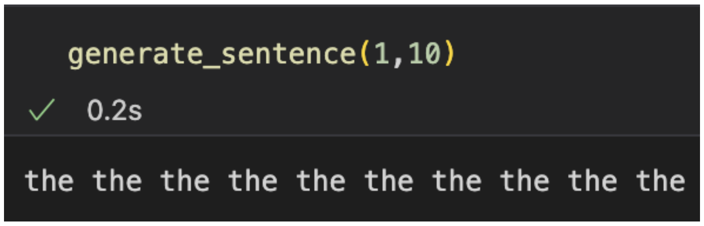
We can observe that as the value of n increases in the model, the context increases (word tuples form phrases), and the model starts to copy/overfit the training set. The most common word in English is ‘the’ and it is reproduced by our model in the unigram case. However, as the complexity of the model increases, we can see that it has high occurrences of ‘Harry Potter’ specific phrases like “ron and hermione” and “defense against the dark arts”. Due to this reason, models up to (n=5) have been created and not any further.

- **Sentence Generation**

(Remark: Perplexity is mentioned in terms of Log2-perplexity i.e $\log_2$(PP) throughout the report)

The function ‘generate_sentence’ takes in 2 parameters, n (n-gram’s model complexity) and word_limit (length of the sentence needed). Some sentences that have the maximum probability of occurrence created by setting each value of n are as follows:

1. Unigram

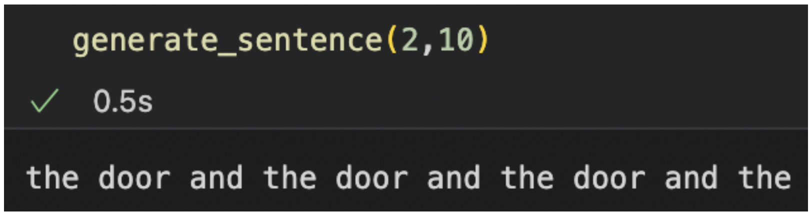


```
generate_sentence(1,10)
✓ 0.2s

the the the the the the the the the the
```

Log2-perplexity: 4.47
Quite evident as ‘the’ has the highest frequency of occurrence in the corpus.

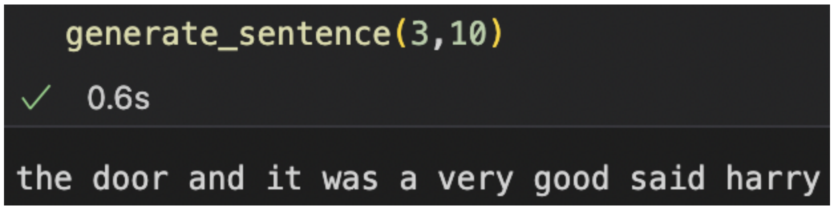
2. Bigram



```
generate_sentence(2,10)
✓ 0.5s
the door and the door and the door and the
```

Log2-perplexity: 4.58
Cyclic sentences formed as "the door", "door and", "and the" have some of the highest bigram occurrences.

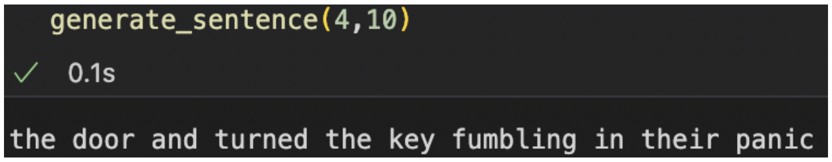
3. Trigram



```
generate_sentence(3,10)
✓ 0.6s
the door and it was a very good said harry
```

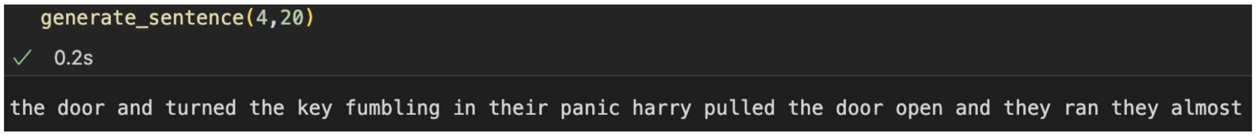
Log2-perplexity: 3.67
Some semantic sense forming.

4. 4-gram



```
generate_sentence(4,10)
✓ 0.1s
the door and turned the key fumbling in their panic
```

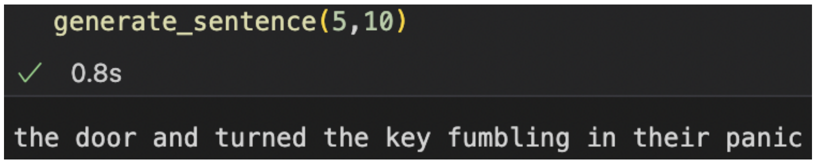
Log2-perplexity: 2.04



```
generate_sentence(4,20)
✓ 0.2s
the door and turned the key fumbling in their panic harry pulled the door open and they ran they almost
```

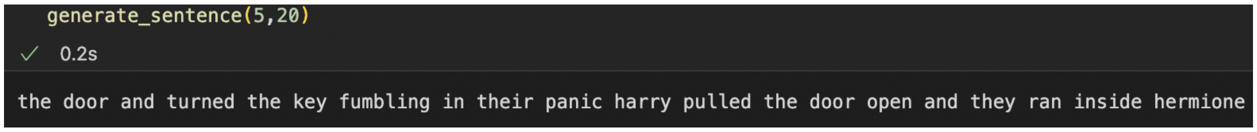
Log2-perplexity: 1.56

5. 5-gram



```
generate_sentence(5,10)
✓ 0.8s
the door and turned the key fumbling in their panic
```

Log2-perplexity: 1.98



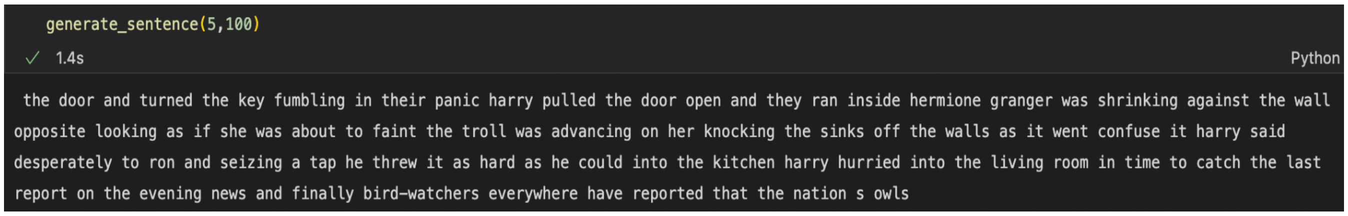
```
generate_sentence(5,20)
✓ 0.2s

the door and turned the key fumbling in their panic harry pulled the door open and they ran inside hermione
```

Log2-perplexity: 1.18

For 4 and 5-gram models the same output is received for 10-word sentences, maybe due to the same max-frequency combination for each respective tuple. But we can see the difference by raising the word limit. The sentences formed make much more sense than the previous ones.

A larger example:



```
generate_sentence(5,100)
✓ 1.4s                                                                                          Python

 the door and turned the key fumbling in their panic harry pulled the door open and they ran inside hermione granger was shrinking against the wall
opposite looking as if she was about to faint the troll was advancing on her knocking the sinks off the walls as it went confuse it harry said
desperately to ron and seizing a tap he threw it as hard as he could into the kitchen harry hurried into the living room in time to catch the last
report on the evening news and finally bird-watchers everywhere have reported that the nation s owls
```

Log2-perplexity: 0.35

With a 100-word sentence based on a 5-gram model we can see the effects of overfitting as someone having read ‘Harry Potter’ can easily figure out that this paragraph has been the result of mashing together different arcs of the story found across the series.

Also as these are maximum probability sentences formed, the perplexity keeps decreasing as the n increases (generally) because the context in the case of a higher n is much less than in the case of a lower n. This leads to a higher probability (smaller denominator) and in turn a lower perplexity.

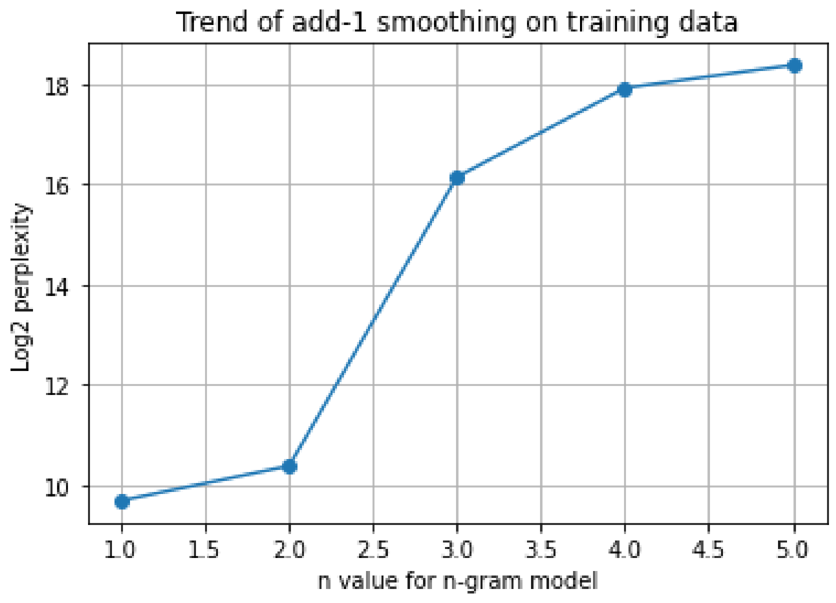
(NOTE- end of sentence punctuation was removed to get better context between 2 different sentences as well as for easier sentence generation)

- **Smoothening**

1. Add-One

On training data:

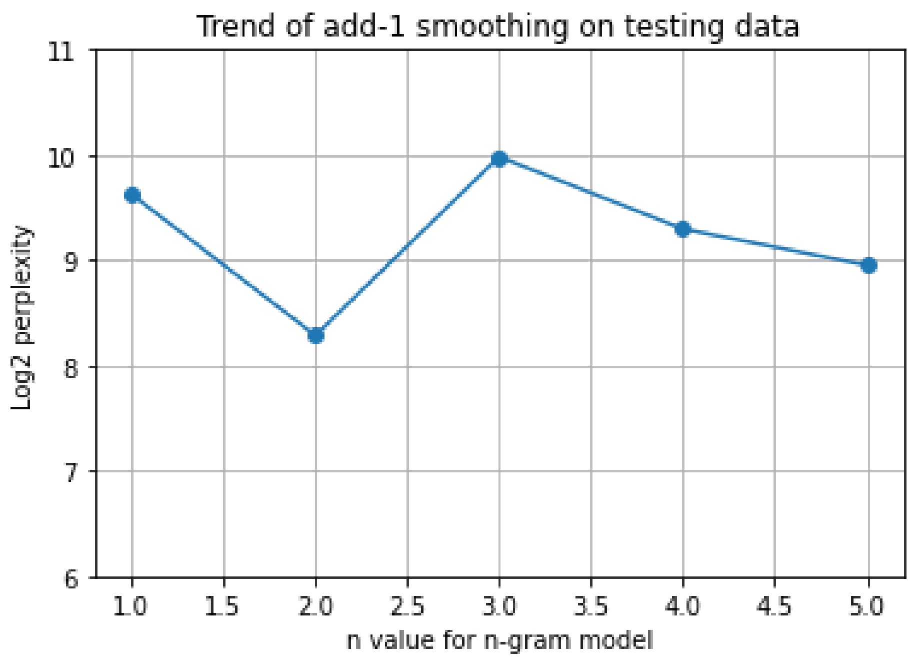
n-value	Log2 perplexity
1	9.68
2	10.37
3	16.15
4	17.92
5	18.38



The effect of increasing sample space as n increases is seen to be dominant in this graph causing the denominator while causing probability to increase leading to less probability and high perplexity.

On testing data:

n-value	Log2 perplexity
1	9.62
2	8.29
3	9.98
4	9.30
5	8.95

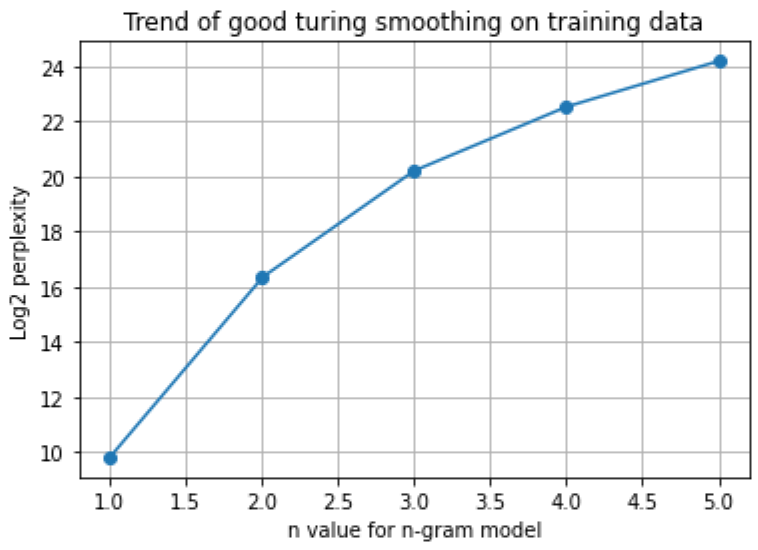


Performance of this model is not at par with other smoothing method models as it reaches fairly high perplexities.

2. Good-Turing

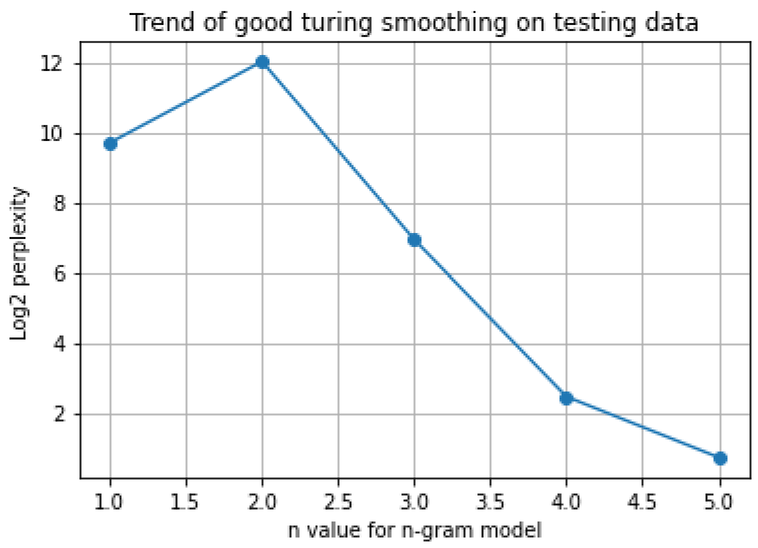
On training data:

n-value	Log2 perplexity
1	9.80
2	16.31
3	20.19
4	22.52
5	24.18



On testing data:

n-value	Log2 perplexity
1	9.70
2	12.02
3	6.94
4	2.46
5	0.73

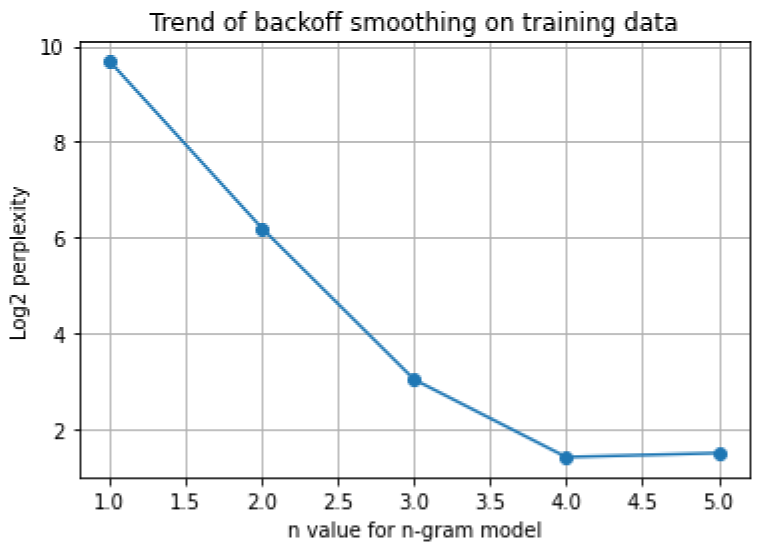


5-gram models take into consideration tuples of length 5. As they are tuples of length 5, the possibility of two of them being the same are very less. This leads to more unique tuples i.e. the number N_1 (frequency of 5-tuple elements with frequency 1) becomes almost the same as N (total number of 5-tuples). This causes a very high probability to be multiplied many times (each time a unique 5-tuple is encountered), leading to an extremely skewed

3. <u>Backoff</u>

<u>On training data:</u>

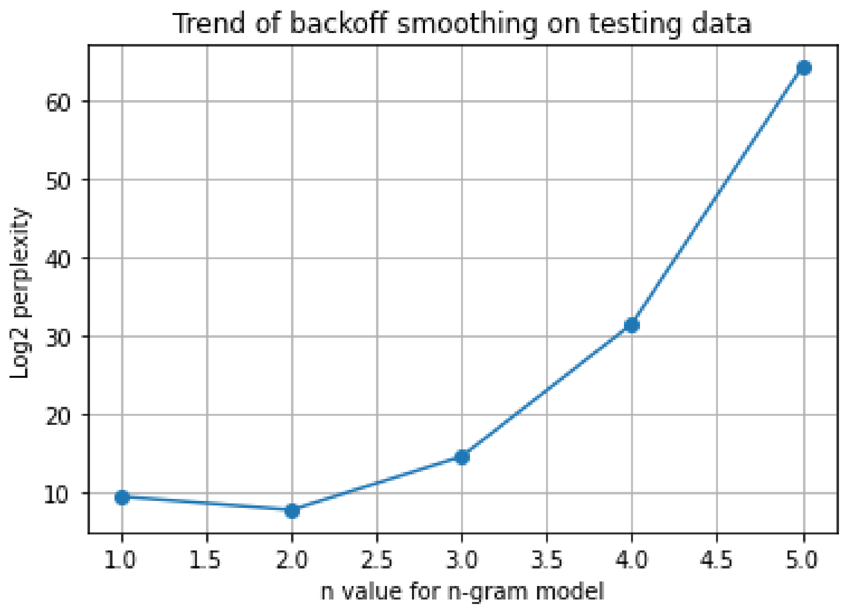
n-value	Log2 perplexity
1	9.68
2	6.19
3	3.03
4	1.41
5	1.49



We can see the bent elbow shape while training at n=4 indicating some degree of overfitting may be happening after this n-value.

<u>On testing data:</u>

n-value	Log2 perplexity
1	9.52
2	7.82
3	14.65
4	31.57
5	64.36

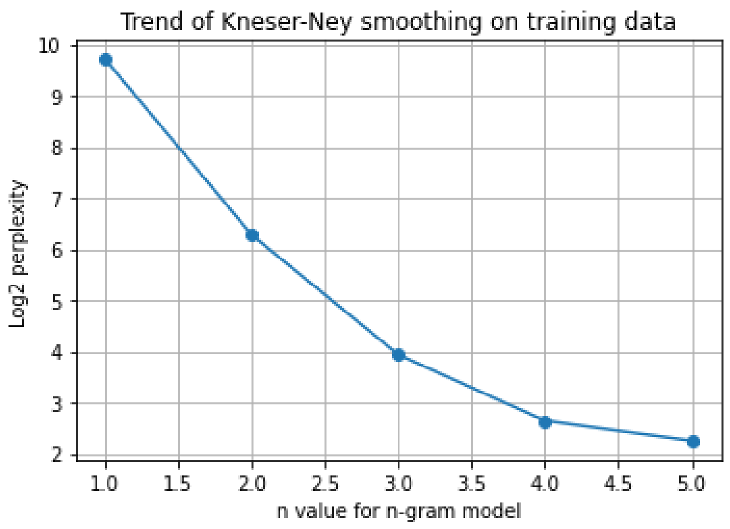


Performs fairly decently till n=3 but after that exponential growth is seen. Can be related to the same reason as the diminishing of Good Turing smoothing method testing data pattern. On every backoff step a penalty of alpha (= 0.4) is added and many of the 5-tuples get backed off as the chance of a 5-tuple of testing data being present in the training data has a lesser chance. So the penalties keep getting added more and more leading to lesser probabilities and thus very high perplexities.

4. Kneser-Ney

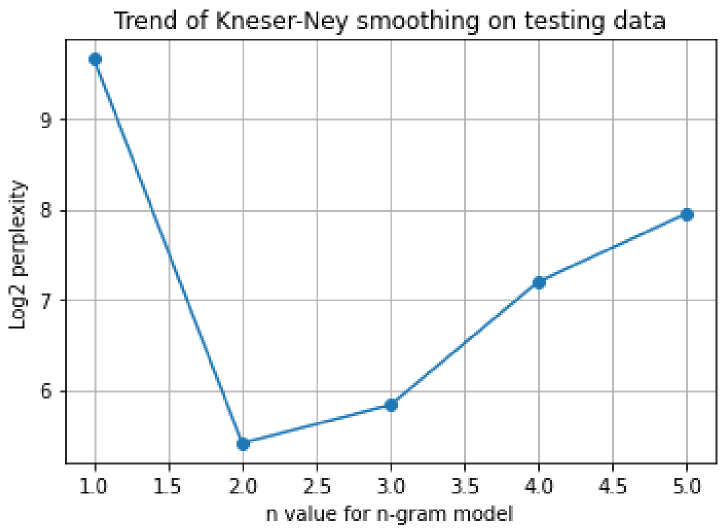
On training data:

n-value	Log2 perplexity
1	9.71
2	6.28
3	3.93
4	2.64
5	2.25



<u>On testing data:</u>

n-value	Log2 perplexity
1	9.66
2	5.41
3	5.83
4	7.20
5	7.94

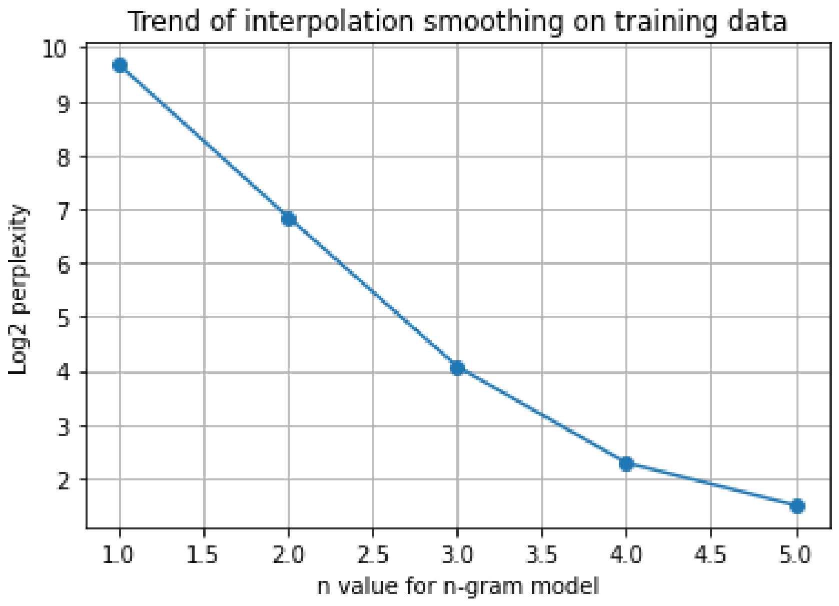


KN model follows all the correct trends that any learning model is supposed to follow. Training perplexity keeps dropping and a local minima found for testing data thus indicating overfitting happening. Fairly good perplexity values achieved.

5. Interpolation

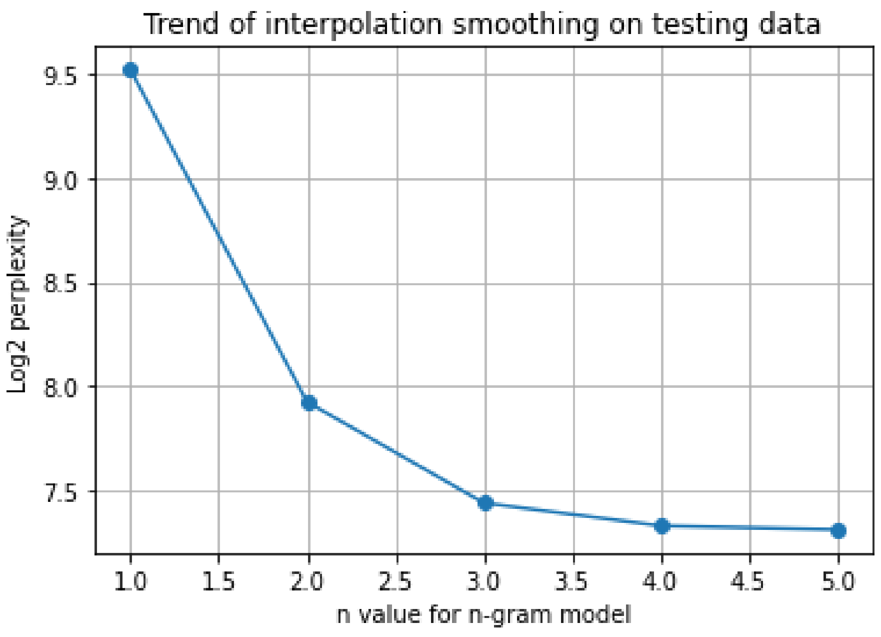
On training data:

n-value	Log2 perplexity
1	9.67
2	6.85
3	4.07
4	2.28
5	1.51



On testing data:

n-value	Log2 perplexity
1	9.52
2	7.92
3	7.43
4	7.32
5	7.31



Attempts made for hyperparameter tuning of weights in interpolation method:

1) Using cvxpy convex optimization library:

Probability of each of the 5 n gram models were stored in a Tx5 array (A) where T is the number of tuples formed from the training data. A variable (x) of length 5 (size 5x1) was made and matrix multiplied with that array (Ax). This would result in the array (Tx1) of final probabilities for each tuple. So an attempt was made to maximize the L_2 norm of the final vector so that the probability of the words can be maximized and thus reduce the perplexity. However the cvxpy library threw an DCPerror saying that the function's maximum couldn't be found.

Then the optimizing function was changed to sum of the final vector (as sum is an affine function), but it yielded very skewed values of [2.16848464e-13, 3.04011103e-1, 5.44029660e-11, 1.24025898e-09, 9.99999999e-01]. This is not practically feasible as most of the weight has been given to the 5th weight corresponding to $P_{5\text{-gram}}$ but in most of the cases for the development data $P_{5\text{-gram}}$ would be 0 which could result in 0 or near 0 probabilities.

2) Sequentially trying out sets of values:

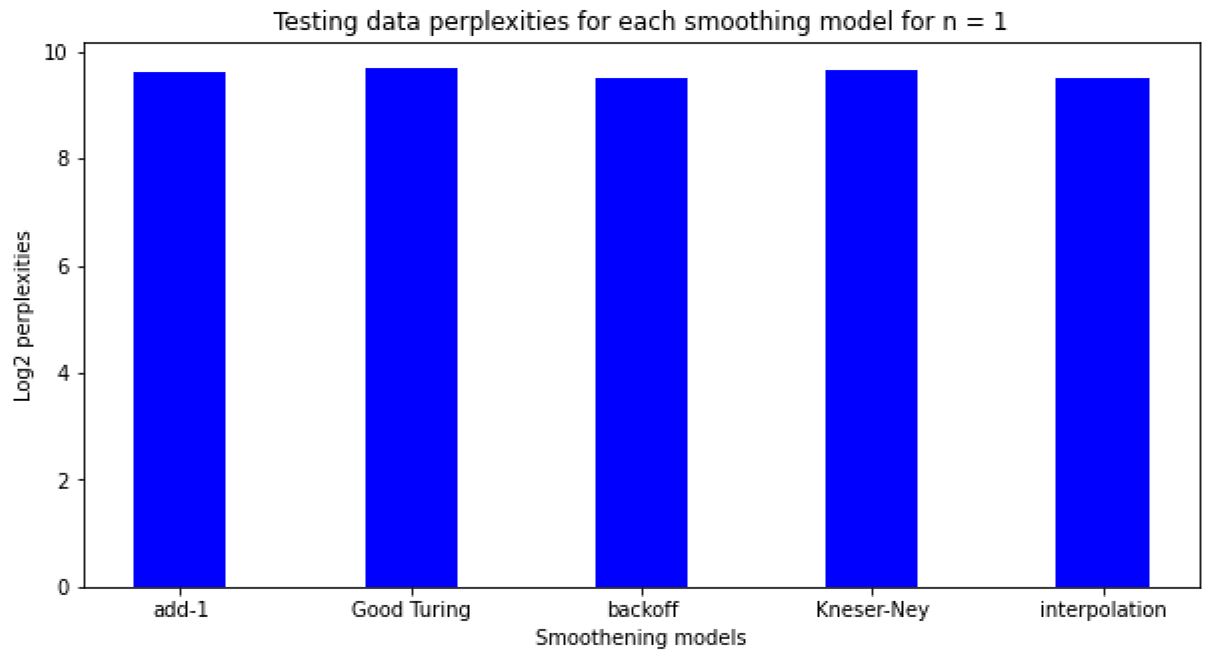
Sequentially tried out values of all the 5 weights using multiple for loops by setting equal interval increments for all the loops. It too replicated the results suggested in the cvxpy library by giving the minimum possible value to the first 4 weights and setting the highest value aside for the last weight.

<u>Other possibilities:</u>

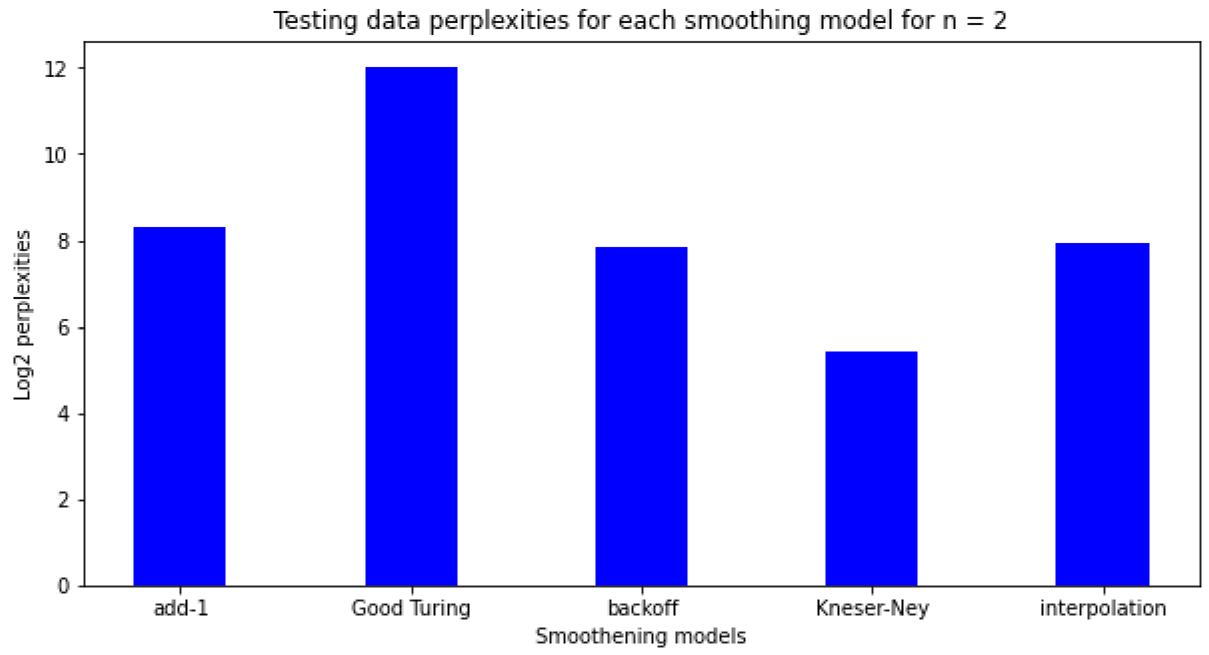
Hyperparameter tuning could also be done for the Backoff and Kneser-Ney methods for the backoff penalty (alpha) and discount value (d) respectively. However I used the general approximations as mentioned in the book in my models.

- <u>**Model Comparison on Testing Data**</u>

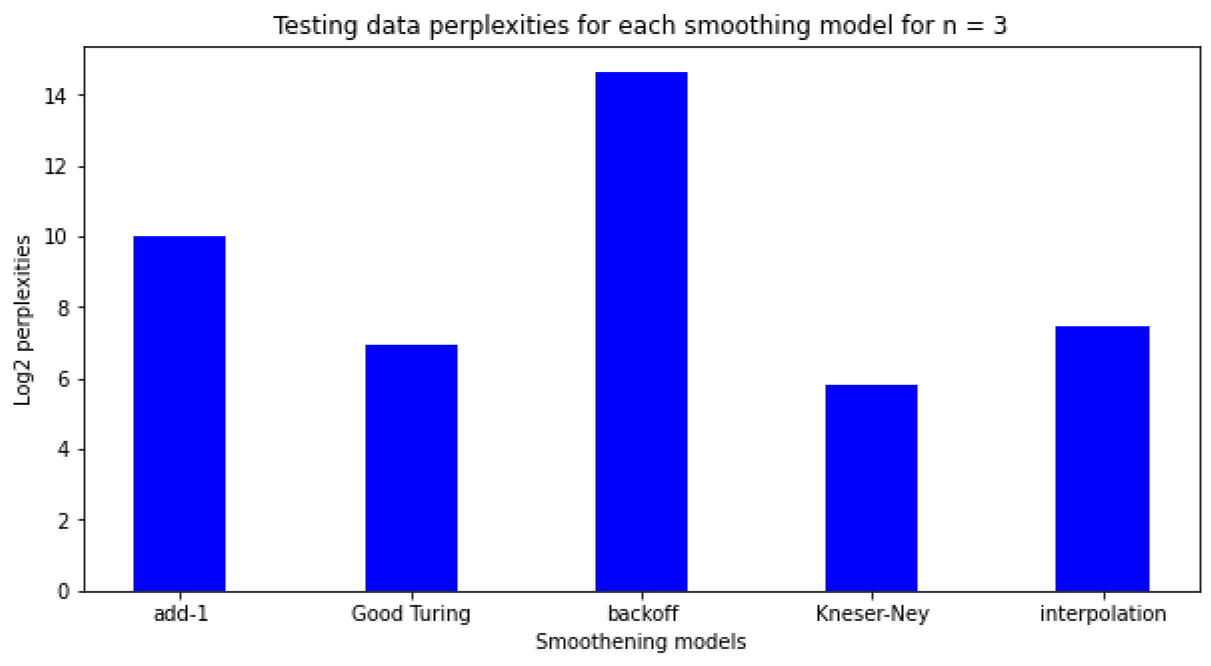
1. <u>n-value = 1</u>



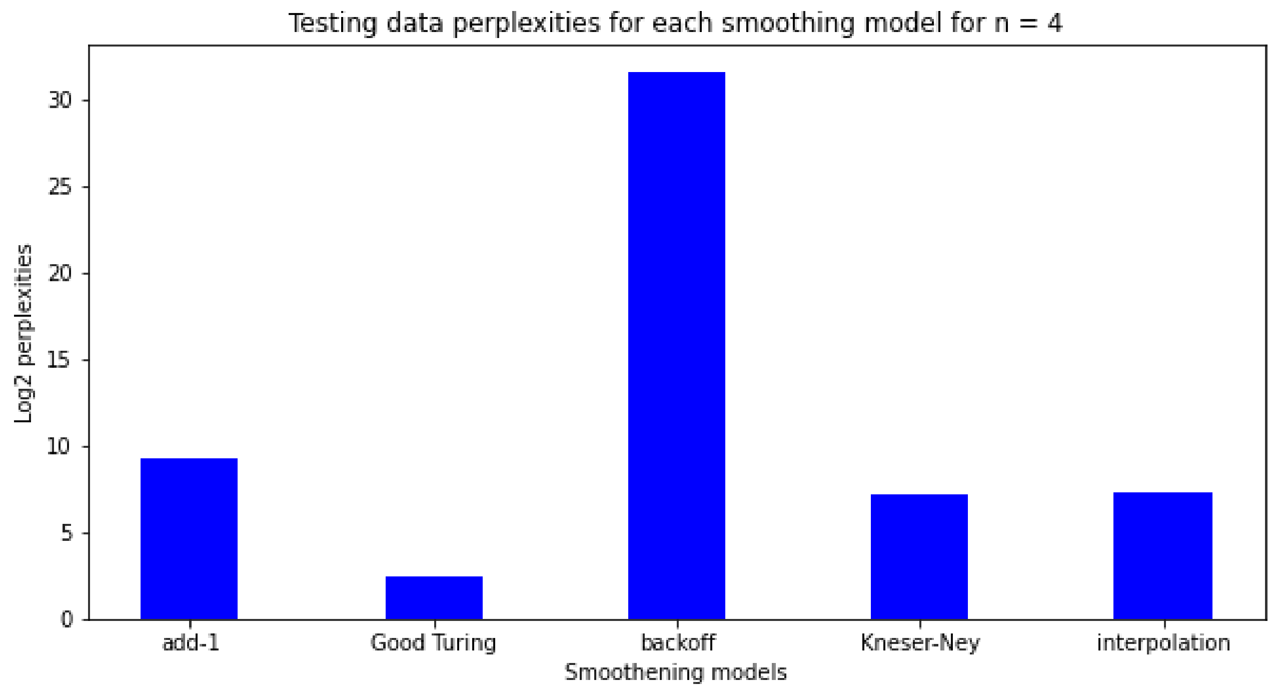
2. <u>n-value = 2</u>



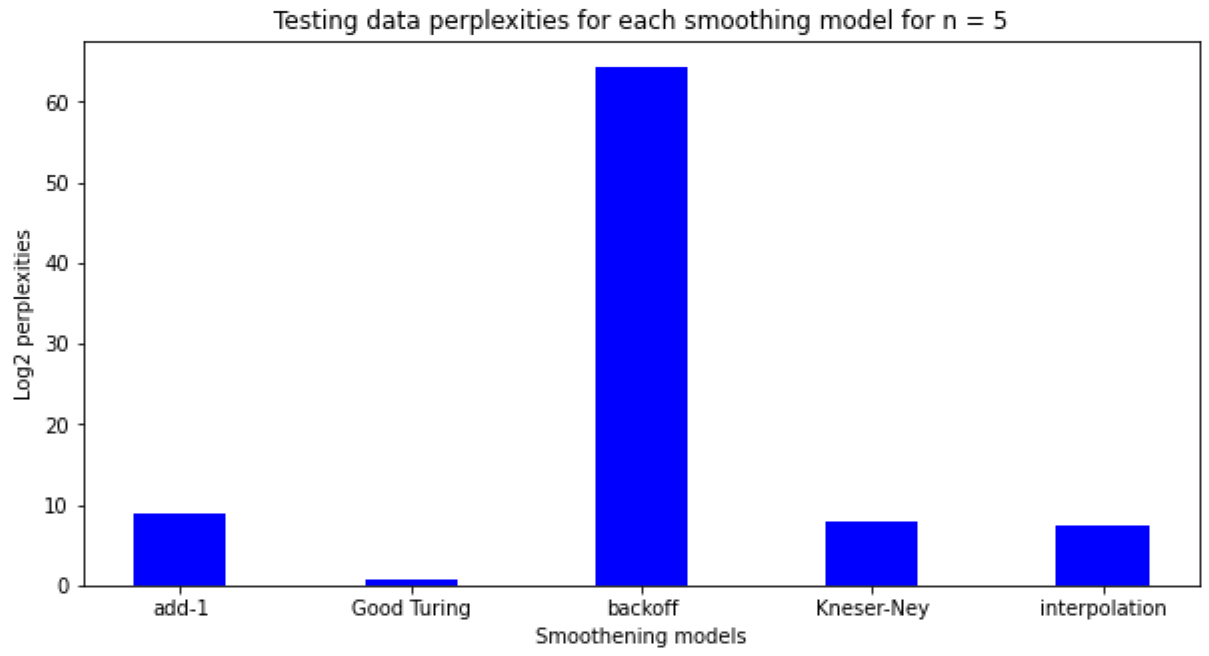
3. <u>n-value = 3</u>



4. <u>n-value = 4</u>



5. <u>n-value = 5</u>



- **<u>Conclusion</u>**

As a general trend **Kneser-Ney** models have shown consistently good values of perplexity with the minimum being $2^{5.414}$ = **42.63** for the **bigram model**. I will consider this to be the best model for over the given datasets by my experiments. (This excludes the exponential behaviour shown by the Good turing model for n values 4 and 5 which give rise to very low but unintuitive perplexity values. As explained in the above sections I believe this behaviour is shown due to the abundance of unique 4 and 5 tuples in the dataset)