<u>Speech and Natural Language Processing</u>
<u>Assignment -1 : Language Modelling</u>

Suyash Damle
15CS10057

Setting up the environment and running the code:

- This code is meant to run on **Python3, (python 3.6).** Hence, to cater to multiple versions of python, another environment can be created, using
  conda create -name ____ anaconda -python = 3.6
- The new environment can be activate using *source activate <name>.*
- The program, upon running, **asks for the name of the test file via the terminal.**
- Also, by default, first call to the model creation method **generates plots to illustrate proof of Zipf's Law.** These are saved in the same folder by the program.

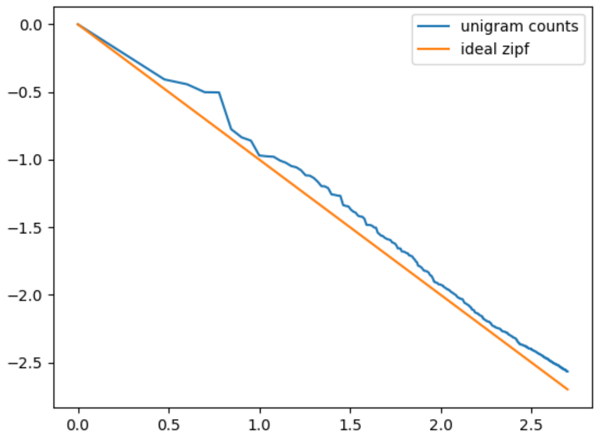
Submitted files:

- The python code file
- .png images of the plots for zipf's law verification
- the generated output file: ***"output.txt"***
- report in pdf format

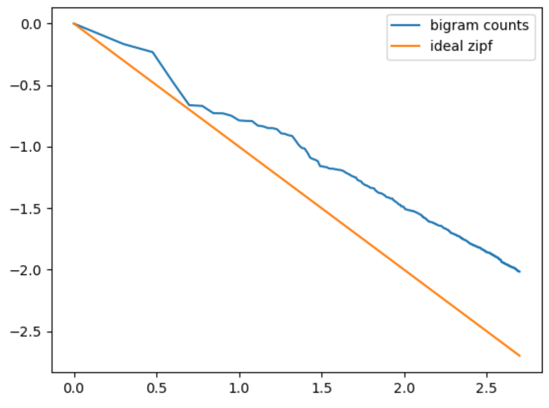
Task 1:

- Language models created:
  - no smoothing used
  - while predicting, absence of token leads to 0 in prob and *-inf* in log – likelihood values

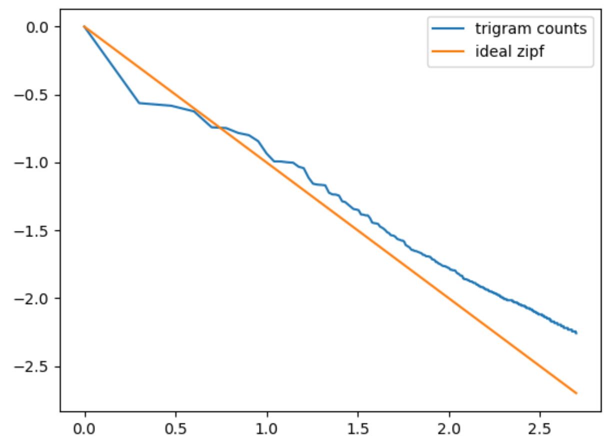
- Zipf's Law verification:
  - Top 500 tokens taken into account
  - log-log plots obtained
  - ideal plot is straight line with negative slope
  - For proper representation, the count values are **normalized wrt the most frequent token** to get values on similar scale
  - Plots for the 3 cases:



Unigram Model



Bigram Model



Trigram Model

Hence, the models closely follow the zipf's law.

- Top 10 tokens:
  - Unigrams

```
('the', 56448)
('START', 40000)
('\\START', 40000)
('of', 31276)
('and', 22092)
('to', 20341)
('a', 17780)
('in', 17705)
('is', 9474)
('that', 8240)
```

  - Bigrams

```
(('of', 'the'), 8508)
((None, 'the'), 5798)
(('in', 'the'), 4985)
(('to', 'the'), 2819)
(('and', 'the'), 1848)
(('on', 'the'), 1821)
(('for', 'the'), 1591)
((None, 'in'), 1585)
((None, 'it'), 1516)
(('it', 'is'), 1390)
```

  - Trigrams

```
((None, None, 'the'), 5798)
((None, None, 'in'), 1585)
((None, None, 'it'), 1516)
((None, None, 'he'), 1377)
((None, None, 'this'), 1052)
((None, None, 'but'), 1038)
((None, None, 'a'), 955)
```

```
((None, None, None), 920)
((None, None, 'and'), 831)
((None, None, 'i'), 675)
```

- Scores on test sentences

```
at n =  1  score =  [-25.867 -17.558 -16.59  -22.23  -18.492] ; perpl =  [36.086 20.96  17.722 47.093 24.64 ]
at n =  2  score =  [-31.484 -25.902 -29.072    -inf -22.932] ; perpl =  [234.099 397.288 826.304     inf 200.035]
at n =  3  score =  [  -inf   -inf   -inf   -inf -4.807] ; perpl =  [  inf   inf   inf   inf 5.292]
```

** Here, “scores” refers to the log-likelihood scores and “perpl” means perplexity

- Observations:
  - Several of the log-likelihood scores and preplexity values are ***inf*** indicating that these were not found in the corpus and hence indicating the need of smoothing
  - In the case where it exists, the perplexity of the sentence (the last one) is quite low, indicating **better language model**

Task 2 (with Laplacian Smoothing):

- Scores for test sentences:

```
at n =  1 , k =  0.0001  score =  [-25.867 -17.558 -16.59  -22.23  -18.492] ; perpl =  [36.086 20.96  17.722 47.093 24.64 ]
at n =  2 , k =  0.0001  score =  [-35.539 -28.907 -32.094 -46.631 -25.979] ; perpl =  [  472.658   795.391  1661.015 47763.625   404.425]
at n =  3 , k =  0.0001  score =  [-41.634 -23.553 -34.632 -21.105  -8.01 ] ; perpl =  [1.506e+04 3.508e+03 1.632e+05 1.502e+03 1.605e+01]
```

```
at n =  1 , k =  0.001  score =  [-25.867 -17.558 -16.59  -22.23  -18.492] ; perpl =  [36.086 20.96  17.722 47.093 24.64 ]
at n =  2 , k =  0.001  score =  [-38.159 -30.686 -33.966 -45.227 -27.974] ; perpl =  [  744.314  1199.937  2560.422 34534.053   641.277]
at n =  3 , k =  0.001  score =  [-40.376 -22.342 -32.023 -21.758 -11.9  ] ; perpl =  [1.126e+04 2.306e+03 6.606e+04 1.883e+03 6.182e+01]
```

```
at n =  1 , k =  0.01  score =  [-25.867 -17.558 -16.59  -22.23  -18.492] ; perpl =  [36.086 20.96  17.722 47.093 24.64 ]
at n =  2 , k =  0.01  score =  [-41.17  -32.109 -35.891 -44.02  -30.351] ; perpl =  [ 1254.083  1666.92   3993.979 26130.033  1110.392]
at n =  3 , k =  0.01  score =  [-40.794 -22.774 -30.882 -24.139 -17.506] ; perpl =  [12399.983  2678.166 44481.982  4298.334   431.465]
```

```
at n =  1 , k =  0.1  score =  [-25.867 -17.558 -16.59  -22.23  -18.492] ; perpl =  [36.086 20.96  17.722 47.093 24.64 ]
at n =  2 , k =  0.1  score =  [-45.882 -34.083 -38.876 -43.972 -33.862] ; perpl =  [ 2837.66   2630.564  7960.293 25838.651  2499.506]
at n =  3 , k =  0.1  score =  [-42.745 -25.118 -30.644 -27.173 -23.717] ; perpl =  [19461.548  6034.874 40966.246 12299.623  3712.991]
```

```
at n =  1 , k =  1.0  score =  [-25.867 -17.558 -16.59  -22.23  -18.492] ; perpl =  [36.086 20.96  17.722 47.093 24.64 ]
at n =  2 , k =  1.0  score =  [-51.762 -37.727 -42.369 -44.789 -38.147] ; perpl =  [ 7861.535  6104.423 17840.09  31209.85   6726.355]
at n =  3 , k =  1.0  score =  [-44.949 -28.045 -30.612 -29.61  -28.611] ; perpl =  [32385.493 16643.178 40506.96  28627.917 20245.992]
```

** Here, “scores” refers to the log-likelihood scores and “perpl” means perplexity

- Observations:
  - The **perplexity values are typically higher in bigram model and higher still in trigram models** This indicates that o**ur trigram models are not able to learn enough context, probably because of small data-size.**
  - Also, **perplexity increases with k.** That is, the smoothing factor should have small value for better models.
  - No values are ***inf*** this time because of smoothing.

Task 3 (Good Turing Smoothing):

- Assumptions used:
  - If the value for n_{r+1} for some 'r' becomes zero, the next higher (closest) non-zero count is taken into consideration.
    So, if 'a' appears 45 time; 'an':45 and 'the': 48,
    effective count for those occuring 45 times = 48 * (1 / 2)
  - The effective count for the last (most frequent) is taken to be zero.

- <u>Reason that GT smoothing could not be applied to unigram model</u>:
  Good Turing method requires one to evaluate the effective count of tokens occurring 0 times. For bigrams, trigrams, etc, this is evaluated by calculating *all possible bi- (tri-) grams* and subtracting the count of the seen ones from this.
  This is not possible to be done with the unigram model. The GT method assumes the set of all unigrams to constitute the dictionary.

- Scores on test sentences:

```
at n =  1  score =  [-25.867 -17.558 -16.59  -22.23  -18.492] ; perpl =  [36.086 20.96  17.722 47.093 24.64 ]
at n =  2  score =  [-71.336 -48.168 -54.759 -67.688 -49.234] ; perpl =  [ 233655.769   68128.014  312360.552 6194604.382   87147.503]
at n =  3  score =  [-115.604  -64.24   -92.915  -69.147  -45.378] ; perpl =  [3.982e+11 4.668e+09 9.664e+13 2.556e+10 6.761e+06]
```

** Here, "scores" refers to the log-likelihood scores and "perpl" means perplexity

- Observations:
  - Once again, the **perplexity values are typically higher in bigram model and higher still in trigram models**
    This indicates that o**ur trigram models are not able to learn enough context, probably because of small data-size.**
  - The obtained values of perplexity here are higher than Laplacian Smoothing, implying worse language models.

Task 4 (Interpolation Method):

- Only the bigram model considered.

- Scores on test sentences

```
at n = 2, lambda =  0.2  score =  [-23.608 -18.801 -17.964 -23.594 -16.262] ; perpl =  [ 59.797  77.013  63.473 233.064  42.834]
at n = 2, lambda =  0.5  score =  [-25.05  -20.291 -19.789 -25.393 -17.759] ; perpl =  [ 76.775 108.649  96.768 353.228  60.529]
at n = 2, lambda =  0.8  score =  [-27.414 -22.66  -23.018 -28.579 -20.082] ; perpl =  [115.634 187.818 204.02  737.479 103.525]
```

- Observations:
  - The perplexity values are **lower than the corresponding ones without smoothing and best of all models.**
  - The perplexity values are **increasing with lambda** in this range of values. So, **interpolation method with lambda = 0.2 is found to give best language model.**