# CS613 Assignment 2

Link to the Github Repository - https://github.com/Nik2630/NLP_Assignment_2

Original Dataset - raw_reddit_data_filtered.csv
Test Set - Test_Set_Preprocessed.csv
Train Set - Train_Set_Preprocessed.csv

Data Preprocessing:

a. Multi-line comments are broken down into single line comments.
b. Used NLTK sentence tokenizer to divide comments into sentences.
c. Used 'Contractions' lib to expand contractions.
d. Removed symbols and URLs
e. Lowercasing

7. Observations:

a. From Unigram to Quadgram - Perplexity Decreases (from thousands to unit digits) , Number of non-perplexable sentences increase. (From 10% to 60/70%)
b. No-Smoothing to Smoothing - Perplexity Increases (to hundreds of thousands) ,Number of non-perplaxable sentences decrease significantly (to almost negligible amounts, comparatively)

| | Perplexities | | Non-Perplexable Sentences | |
|---|---|---|---|---|
| | No smoothing | Smoothing | No Smoothing | Smoothing |
| Unigram | 4619.33 | 13536.9 | 6387 | 151 |
| Bigram | 427.39 | 341206.12 | 34794 | 230 |
| Trigram | 10.41 | 195924.53 | 1337 | 436 |
| Quadgram | 5.64 | 941196.15 | 44551 | 530 |

-Total number of sentences in Test set are 66563

Justification:

A. Perplexity is an inverse measure of our model's accuracy to predict words based on last n-words. In other words, it's an inverse measure of our model's ability in predicting sentences. We see that as we move from unigram to quadgram, we are able to predict fewer sentences given that our models are trained on the same test set. However, our

accuracy in predicting those words increases. The same holds true as we smooth our models, when we try to predict more sentences, our accuracy on predicting them decreases.

B. Why do the number of non-perplexable sentences increase as we move from unigram to quadgram?
   Non-perplexable sentences are the ones whose last n-words weren't encountered by the model during training (for no smoothing). As we increase n, the number of possible keys, which is the number of possible combinations of n-words from a vocabulary, increases. This means that if our training set remains the same, we are training the model on fewer proportions of possible keys. Since the total number of keys trained from a test-set is almost the same. This makes it less probable for our model to have encountered a key earlier and increases the number of non-perplaxable sentences.
C. Why does perplexity decrease as we move from unigram to quadgram?
   As we increase n, we increase our accuracy of predicting the correct word based on the last n-word. This is because our model is taking more context into account. The more context a model takes into account, the more accurate it is, and the larger training set it requires.
D. Why does perplexity increase with smoothing?
   When we do smoothing, we try to predict the sentences that were previously unpredictable. They had probability zero and infinite perplexity without smoothing. Even though those perplexities are finite after smoothing, they are still very large numbers, thus making the average perplexity many orders higher magnitude than before. If we only consider the sentences that were perplexable before, we will see that the mean perplexity has only slightly increased instead in comparison to this gigantic increase.
E. Why does the number of non-perplexable sentences decrease with smoothing?
   The very goal of smoothing is to give sentences that had zero probabilities before some small probability so that we can decrease the number of non-perplexable sentences.

Perplexities -

| | Add-k Smoothing | Unigram Prior Smoothing | Good Turing Algorithm |
|---|---|---|---|
| Unigram | 61424.73 | - | 8711.57 |

| | | | |
|---|---|---|---|
| Bigram | 543094.67 | 2550480.86 | 57631 |
| Trigram | 96234.91 | 16331.28 | 428773.1 |
| Quadgram | 382132.26 | 27203.25 | 240898.75 |

- Add-k Smoothing: Choosing an appropriate k value decreases the perplexity compared to laplace smoothing. Generally a lower value of k used as it assigns lower probability to new words
- Better than basic Add-k Smoothing, Good-Turing smoothing is used to evaluate the probability of unforeseen events. It works especially well when handling uncommon events. It calculates the probabilities of unseen n-grams using the observed frequencies of n-grams in the training data. It is predicated on utilizing the distribution of observed frequencies; we can calculate the chance of seeing an n-gram 'c+1' times if we have seen it 'c' times in the training data.
  N_c = frequency the frequency of c.
- c* = ((c+1)N_(c+1)) / (N_c)
- P(c) = c*/ N
- We needed to smooth the N_c values when N_c is zero; therefore, we used linear regression to perform the interpolation.
- It assigns higher frequencies based on the observed frequency distribution.

## Contribution -

Darshi Gaurang Doshi (21110050) - Quadgram Without Smoothing and with Smoothing Document, Compilation, Code Debugging, Theory

Ekansh Somani (20110065) - Documentation, Theory

Nikhilesh Myanapuri (22110162) - Preprocessing,Add-k Smoothing,Unigram prior smoothing

Ruchika Sonagote (21110212) - Good Turing Implementation and explanation, debugging

Shreyase Bhadra (22250041) - Code Debugging, Theory

Sriman Reddy (22110124) - Laplace Smoothing, Without Smoothing Codes

## References -


[1] https://medium.com/mti-technology/n-gram-language-model-b7c2fc322799

[2] https://www.geeksforgeeks.org/n-gram-language-modelling-with-nltk/

[3] https://www.cs.cornell.edu/courses/cs6740/2010sp/guides/lec11.pdf

[4] https://www.codingninjas.com/studio/library/smoothing-in-nlp

[5] Lecture Slides