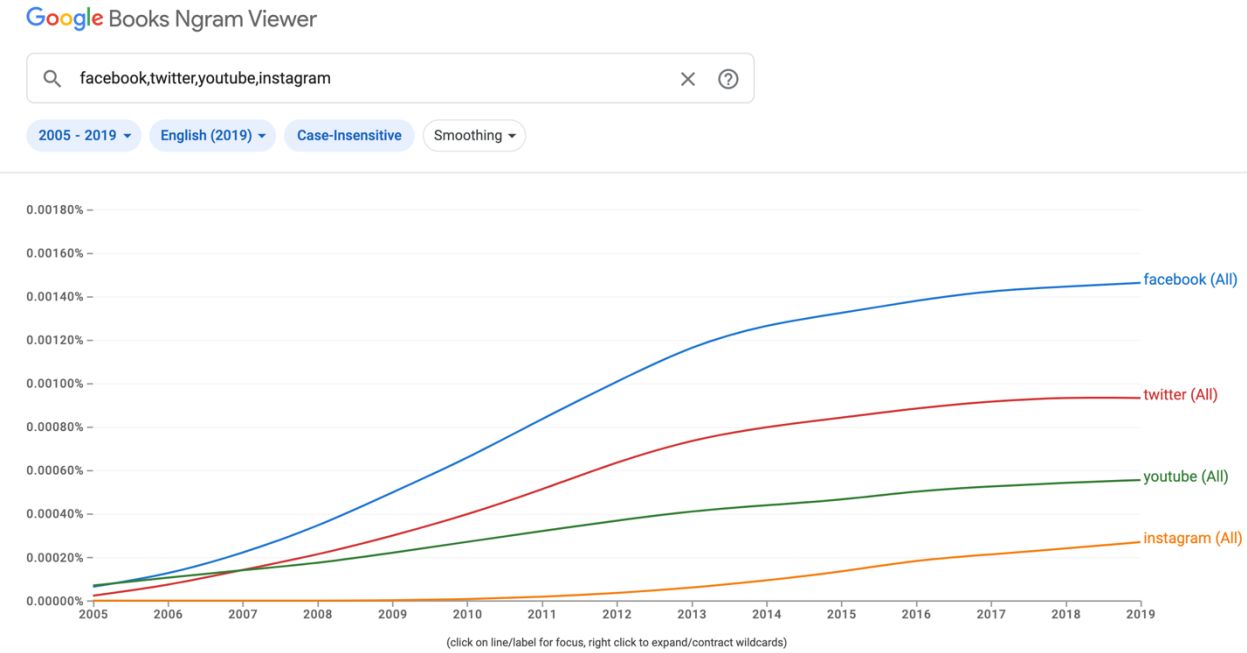
Farishah Nahrin
CS 6301.M02

Narrative - Portfolio Chapter 8: N-Grams

An N-Gram is a “sliding window over text, n words at a time.” Essentially, it is a sequence of N words. Unigrams take one word at a time, bigrams take two words at a time, and trigrams take three words at a time. Essentially, these are called n-grams, and the “n” can represent any number. A probabilistic model of language can be created using n-grams. In order to build a language model as such, a body of text is required. And this body of text can influence the language model. Then you can retrieve a list of unigrams and a list of bigrams for a body of text. Then, you have to make dictionaries of counts for both the unigrams and bigrams, as an example, using the ngram() function in NLTK. Then you can calculate the probability of the frequency, of the sequence of n number of words, using Laplace smoothing or Good-Turing methods. You can use these probabilities to create new corpuses or to predict the prospect of a of a sequence of text. As in, you can use these probabilities to determine that given a start word, what would be the most likely next word. Thus, for language generation, the N-gram counts in NLP applications, the N-Grams are converted to probabilities. The probability of a unigram is just the count of that unigram divided by the total number of words. To specifically know how to calculate probabilities for unigrams and bigrams, there are two methods that you can execute. So first, after retrieving the number of tokens and vocabulary size of the training data, you can calculate the probability using a variation of Good-Turing and laplace smoothing. Then you can add log(p) to prevent an underflow. To calculate the probability using Laplace smoothing, you can use this formula (b + 1) / (u + v) where b is the bigram count, u is the unigram count of the first word in the bigram, and v is the total vocabulary size. This is the simple approach to smoothing. As in, you can do p_laplace = p_laplace * ((n + 1) / (d + V)) to calculate the probability, where n is the number of tokens, d is the unigram count in the unigram dictionary, and V is the number of unique tokens. Smoothing is important in N-grams because when we try to compute a conditional probability of a word or n string of words, there will be a problem because a word might have a count of 0 in a corpus. Multiplying by zero will zero-out the probability which is not the desired result. The problem of zero counts is called the sparsity problem because our data can’t possibly contain every sequence of possible words. One approach to dealing with 0 counts is the smoothing technique. Language models can be used for text generation by using the probability calculations of the frequency of n string of words or a word, as described previously. One way a language model can be used for text generation is for creating text, based on a specific instruction, headline, topic, or word. Another way it can be used is automatically completing a sentence, by predicting the next word, from just a one to a few word input. Limitations could include that some models may not be able to detect the sentiment of a word, or the incorrect grammar, or

incorrect spelling of words; thus, the quality of text can greatly impact the outcome of the model. Also, the models may not be able to detect opinions or strong predispositions and prejudices in a text, which can also impact the outcome of the model. Some applications where the N-grams can be used is for autocompletion of sentences (such as in Gmail, or Google Search), since N-grams are helpful with making next-word predictions. It can also be used in automatic spell checking, and it can also possibly automatically check the grammar in a sentence. N-grams can be used for search-purposes; for example, it can be used to match and find search terms within bodies of text. In conclusion, it can be used for a wide variety of cases where text analyses are required. Some ways that the language models can be evaluated include accuracy, which determines how many words were correctly predicted. And just like any other machine learning model, evaluating the precision, recall, and F-1 score are also good measures. Essentially, language models can be used to determine how well the model captured what it was supposed to, and how useful the model is. The Google Ngram is a search engine that "shows how phrases of text have occurred in a corpus of books over the selected years." You can also determine word or phrase usage, using inflection search, case insensitive search, part-of-speech tags and ngram compositions, which are all features of the Google Ngram. Below is an example of the Google Ngram viewer with the social media sites listed in the search. Facebook has been referenced a lot more frequently than its other social media competitors.



<u>Sources I Used:</u>

- Chapter 8, Exploring NLP with Python by Dr. Karen Mazidi
- https://towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058

- https://github.com/kjmazidi/NLP/blob/master/Part_2-Words/Chapter_08_ngrams/8_ngrams_1.ipynb
- https://www.scaler.com/topics/nlp/language-models-in-nlp/
- https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd
- https://books.google.com/ngrams/info