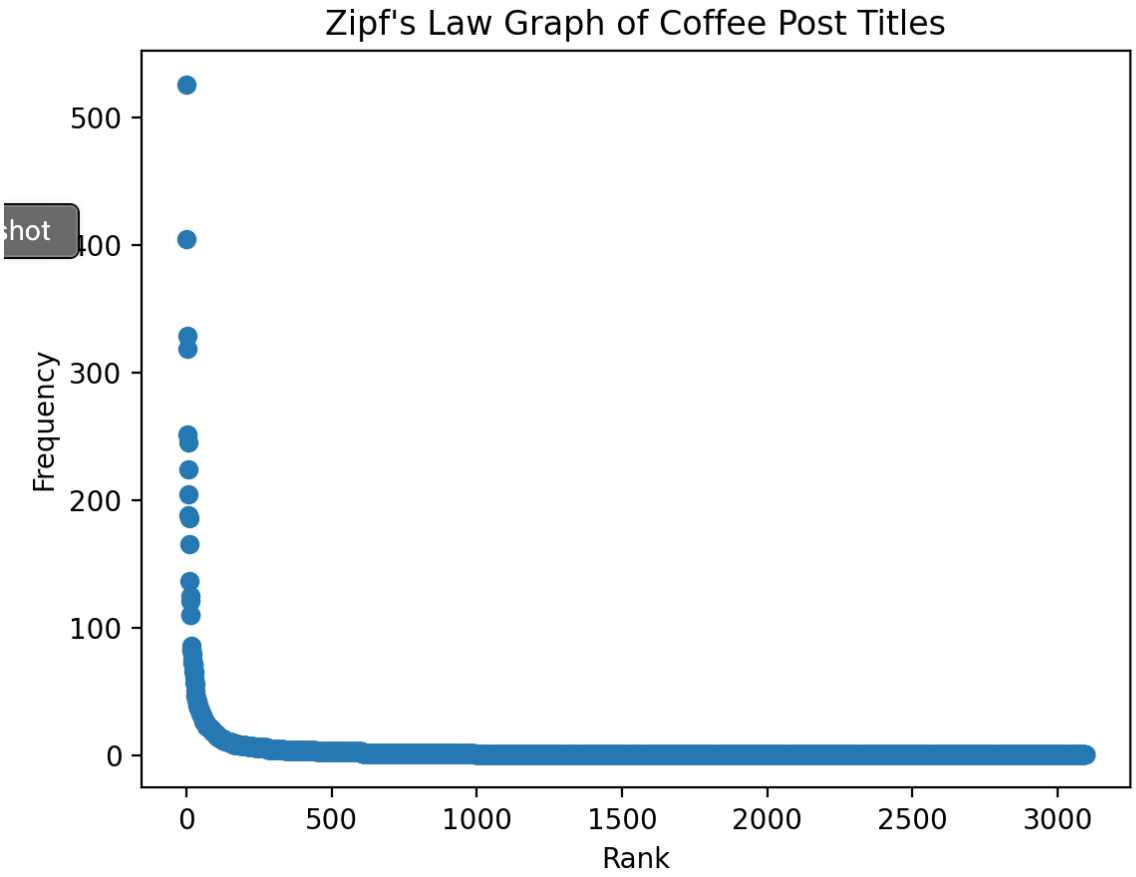
Sean Fletcher
NLP 2023 Assignment 01

Question 1:



In some ways the Zipf's law plot does align with my expectations, and in other ways it does not.

How the Zipf's Law plot does align with my expectations:

The plot created from the titles of the Coffee_Posts.xml file does align with my expectations because of its general shape. As you can see in the image above, the slope of the line LOOKS like the "Zipf's Law line". This is because the words used most frequently are used significantly more than the words used less frequently.

How the Zipf's Law plot does not align with my expectations:

The plot created from the titles of the Coffee_Posts.xml file does not align with my expectations because the word used most frequently "coffee" does not appear twice as frequently as the second most used word, nor does it appear three times as frequently as the third most used word ("coffee" appears 526 times, "the" appears 405 times, "to" appears 329 times" etc…). This means that this data set does not follow Zipf's Law because the frequency of each word is not inversely proportional to its rank. I expected it to more closely match Zipf's Law than it did.

Code available at:

https://github.com/SeanEdwardFletcher/NLP2023/blob/main/Assignments/Ass01/NLP_Ass01_Q01_SeanFletcher.py

Question 2:

Observations:

Word clouds, as an information visualization tool, can be used however anyone might like. I am going to discuss uses for word clouds that do not include stop words, and also word clouds that do include stop words. I will refer to word clouds that include stop words as "stop-word clouds" as stop words make up the vast majority of the images.

In general I feel word clouds without stopwords would be most useful for group-related activities, such as team building or team self-analysis. Say there is a slack channel for all the computer science professors at USM. Using a word cloud of the contents from that slack channel at a department meeting would be a fantastic way to break the ice and start some internal discussion. This idea can be used for small groups or entire companies as long as there is enough data to support creating a word cloud. It would be important to remove the

stopwords in this particular use because in this context the stop words would hold little meaning to the individual participants. As you can see in the Coffee_Posts word cloud examples, the word cloud without stop words has a very semantically rich collection of words.

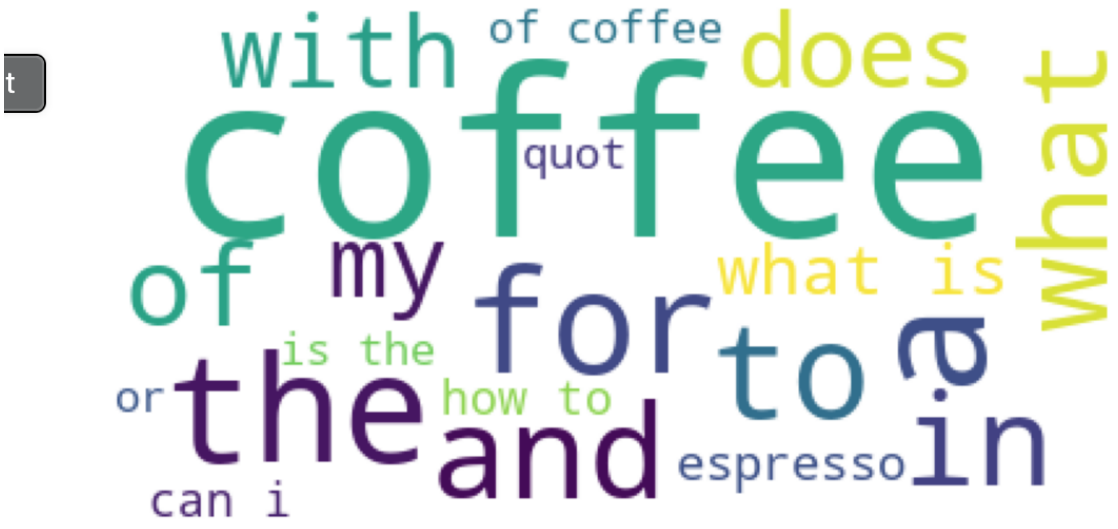
Word clouds that do include stop words can also be useful, depending on the situation. One use could be to generate stop-word clouds using books written in different time periods. You could compare the stop-word clouds from books from the 1720s, 1820s, 1920s and today. The prevalence of various stopwords would undoubtedly change over time and would be quickly and readily understood by viewers. Another use for stop-word clouds could be (though this might be a bit of a stretch) used as a kind of fingerprint. Every person has different speech patterns and habits. A stop-word cloud of the sentences professor Mansouri uses in his lectures would have a large “so” prominently displayed. The stop-word cloud of the sentences from a professor MacLeod lecture, however, might have very different prevalent stop words.

The takeaway is that if you are looking for a semantically rich word cloud, remove the stop words. If you are looking for a syntactically-focused word cloud, then leave the stop words in.

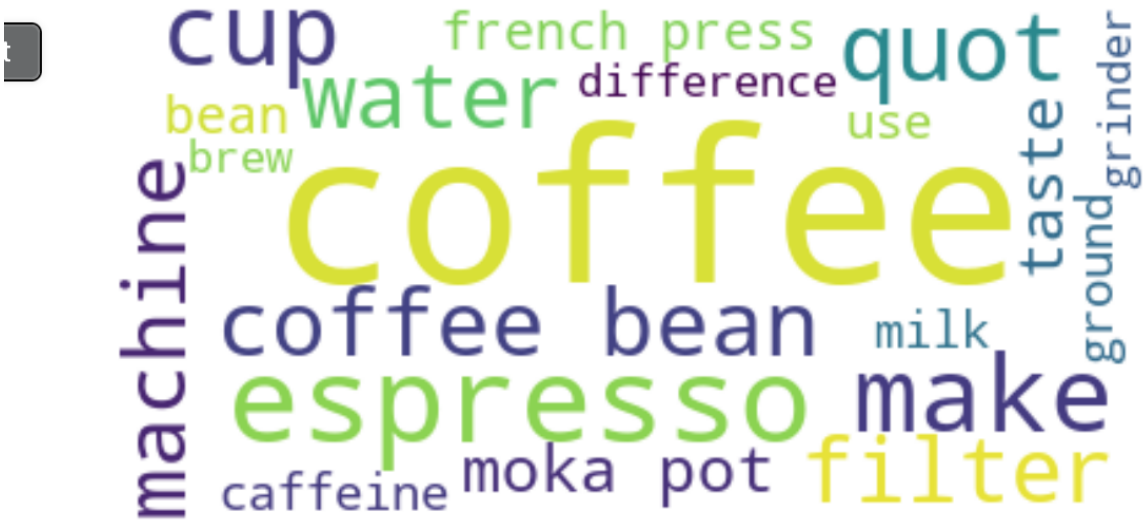
Code available at:

https://github.com/SeanEdwardFletcher/NLP2023/blob/main/Assignments/Ass01/NLP_Ass01_Q02_SeanFletcher.py

Coffee_Posts word cloud (with stop words):



Coffe_Posts word cloud (without stop words):



Question 3:

The first step in training a model with speeches delivered by U.S. presidents will be to gather all the speeches available for each president and to divide them into groups. One group, the largest group, will be used as a training set, a second group will be used as a validation set to help predict the efficacy of a model during the training process, and the third and final group will be held out as a test set for when the model has been trained and validated.

After the data has been collected and separated into sets, the model will then be trained. I will use subword-based tokenization algorithms which will build a set of tokens using the training set as its language. This will start with the individual characters and grow based on which algorithm I use. Eventually all the words from the training set and many of the common English morphemes will be in the model.

This model will also need an evaluation metric so we will be able to tell how accurate it is at its predictions. If we have enough time we could use an extrinsic metric where the model is tested in real time with real people. But we'll still use an intrinsic metric based on testing the model against the validation set. For an intrinsic metric we might choose uni- bi- or -trigram probabilities, or even mix them. We might also want to account for words in the test set that are absent from the training set.

After the model is trained, validated, and finally tested we'll have an intrinsic metric on which to evaluate it. How well did it predict the sentences in the test set? Will it assign a higher probability to a new speech given by the current president when compared to a speech given by a church's minister, the chair of the board of a large company, or a social media influencer? I hope so.

Question 4:

BPE and WordPiece are popular and effective. Really, they are popular BECAUSE they are effective. They are very similar in their implementations as they are both subword-based tokenization algorithms. In this answer I am going to focus on their “bottom-up” implementations, and explain a fundamental difference between them.

A fundamental difference between BPE and WordPiece is that BPE builds its models based on frequency, while WordPiece builds its models based on probability. I’m going to use the following data set to explain.

data set: *low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new*

BPE first uses every character found in the data set as a token (l, o, w, e, s, t, n, r, i, d). Then it finds the pair of characters that appears most frequently and adds that pair as a token. The token pair “we” appears 8 times while the token pair “lo” appears only 7 times. BPE would add “we” as a token before adding “lo” as a token.

WordPiece, like BPE, starts by adding every character. After each single character is added though, WordPiece looks at a token combination’s probability, not its frequency. So while “we” appears 8 times, and “lo” only appears 7 times, “lo” has a higher probability

examples:
“we” → P(‘e’|‘w’) = 8/18 = 44.4%
“lo” → P(‘o’|‘l’) = 7/7 = 100%

and therefore WordPiece would add “lo” before adding “we”.