Tokens Homework 0

By Gregory Sylvester

Data

The data that was used was pulled from Gutenberg[1], which is a free online text provider. The initial data was going to be the ‘book of Rose’, however later on I changed to the “The King James Version of the Bible”. The reasoning is that the bible provided a lot more tokens to and meets the 50,000 requirements.

Methodology

This homework entailed three main phases. The first part was to compile all the need resources. To edit the homework I used PyCharm and created an environment with NumPy and matplotlib. As well as importing all the necessary text need.

The second part was to do the text preprocessing. All the arguments entail are *path* which gives the program the path to the preprocess, the default document to edit should be the “The King James Version of the Bible” (note depending on how your file structure is set up you may need to edit the path variable when running, look to ReadMe file for help). The function for *lowercase* is simple, all it does is return a NumPy array of all the tokens lowercased. The *tokenization* function takes in a string and removes any non-alphabet characters. The function will then split by spaces and return the output. The function *stemming* takes in the array of tokens and removes any prefixes and/or suffixes from the predetermined list (source is [2], also the lists were generated from ChatGPT so I didn’t have to Unpremeditated (found this on google), which would just become “premeditat” after stemming. For better stemming function thou it is highly recommend that I use predefined functions in the future instead of using this since words may have more complex structure which is not captured by my function. *Stopwords* function just compares all tokens to predefined set which I found online [3]. *Shortwords* is the function I got to choose which just removes any tokens that are very short and would contain little to no useful information. *Visualization* is the function which takes in the tokens and applies the NumPy unique function to find the ‘corpus’ and associated frequency. To create the sorted bar graph ChatGPT helped with ordering the bar graph for matplotlib by giving me the sorted index line (line 115). I also limited the bar graph to the top 100 words to make it easier to view. For future I need to add a input value for how big you want the bar graph max to be the default is 100(if you want you can manually change it in the code).

The final part was to compile deliverables to GitHub and write this report.

Sample Output

For this section I just ran the functions in my editor, I haven’t tested running it from command prompt. If there is and error running the program from command prompt let me know.

The last ten are:

('molid', 1)
('moderately', 1)

('modest', 1)
('modification', 1)
('modified', 1)
('moist', 1)
('moistened', 1)
('mole', 1)
('moles', 1)
('accurate', 1)

First ten:

('shall', 9840)
('his', 8473)
('lord', 7966)
('him', 6659)
('them', 6430)
('thou', 5474)
('thy', 4600)
('god', 4472)
('which', 4420)
('said', 3999)

Discussion

This homework assignment taught me in depth methods for how stemming works and other simple techniques to reduce the complexity of text data. It also showed me how to sort bar graphs for future bar graphs.

Most of the top words on my list are common words related to the top and all are very ‘related’.

When looking at the max with log on it looks like a curve(which we know when we remove log will be a straight line). It appears to look like the same type of distribution as zipfs law other than at the tail ends of my graph where excetions to the funcitons aren’t caught.

References

[1]https://www.gutenberg.org/

[2]https://www.fldoe.org/core/fileparse.php/16294/urlt/morphemeML.pdf

[3]https://productresources.collibra.com/docs/collibra/latest/Content/DGCSettings/ServicesConfiguration/co_stop-words.htm#:~:text=The%20list%20typically%20contains%20articles,%2C%20was%2C%20will%20and%20with.