Mohammed Abrar Ahmed <- replace with your name

CS585 Spring 2023 Programming Assignment #01

Due: **Sunday, February 11, 2024, 11:59 PM CST**
Points: **150**

Instructions:

1. Place **all your deliverables (as described below) into a single ZIP** file named:

`LastName_FirstName_CS585_Programming01.zip`

2. Submit it to Blackboard Assignments section before the due date. **No late submissions will be accepted**.

Objectives:

1. (50 points) Perform basic word frequency distribution analysis for a text corpus.
2. (50 points) Calculate probability of a sentence.
3. (50 points) Language Model word prediction.

Deliverables:

Your submission should include:

- **Make sure your code is sufficiently commented! NO Jupyter Notebook files!**
- **Part A**: Python code file(s). Your py file should be named:

  `cs585_P01A_AXXXXXXXX.py`

  where `AXXXXXXXX` is your IIT A number (this is REQUIRED!). If your solution uses multiple files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well.
- **Part B**: Python code file(s). Your py file should be named:

  `cs585_P01B_AXXXXXXXX.py`

  where `AXXXXXXXX` is your IIT A number (this is REQUIRED!). If your solution uses multiple files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well.
- **Part C**: Python code file(s). Your py file should be named:

  `cs585_P01C_AXXXXXXXX.py`

  where `AXXXXXXXX` is your IIT A number (this is REQUIRED!). If your solution uses multiple files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well.

- this document with your results and conclusions. You should rename it to:

`LastName_FirstName_cs585_Programming01.doc or pdf`
MS WORD or PDF formats only, please.

Part A [50 pts]:

Use Python's NLTK package along with the corpora:

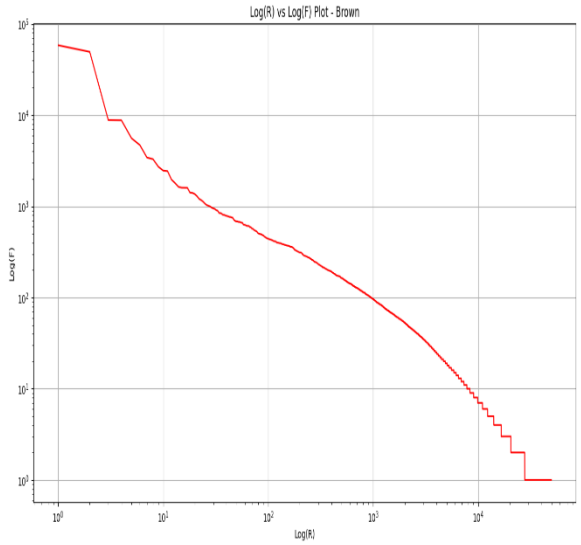
- Brown,
- Reuters,

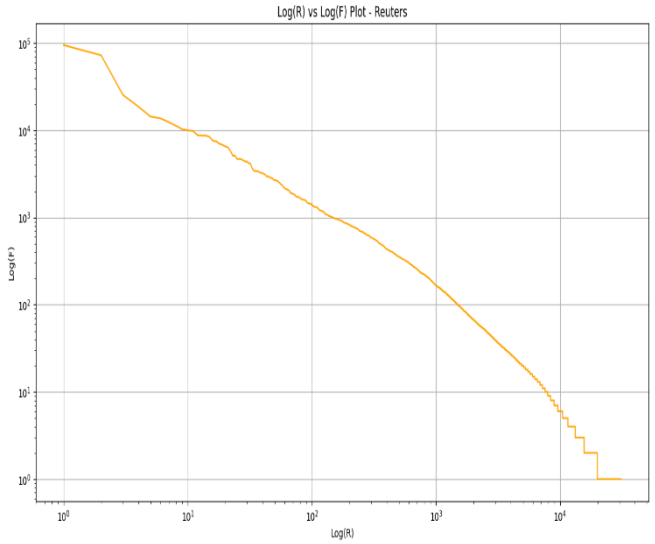
to:
1) **[10 pts]** obtain the word frequency distribution (after removing all stop words; use the `stopwords` corpora for that purpose) for BOTH corpora,
2) **[10 pts]** display a top ten (ranks 1 through 10) words for BOTH corpora on screen (also place them in the table below)

Top 10 words	
Brown	**Reuters**
Brown corpus : Rank of 1: ,, Frequency estimated is: 58334 Rank of 2: ., Frequency estimated is: 49346 Rank of 3: ``, Frequency estimated is: 8837 Rank of 4: '', Frequency estimated is: 8789 Rank of 5: ;, Frequency estimated is: 5566 Rank of 6: ?, Frequency estimated is: 4693 Rank of 7: --, Frequency estimated is: 3432 Rank of 8: one, Frequency estimated is: 3292 Rank of 9: would, Frequency estimated is: 2714 Rank of 10:), Frequency estimated is: 2466	Reuters corpus : Rank 1: ., Frequency: 94687 Rank 2: ,, Frequency: 72360 Rank 3: said, Frequency: 25383 Rank 4: mln, Frequency: 18623 Rank 5: vs, Frequency: 14341 Rank 6: -, Frequency: 13705 Rank 7: dlrs, Frequency: 12417 Rank 8: ', Frequency: 11272 Rank 9: 000, Frequency: 10277 Rank 10: 1, Frequency: 9977

3) **[15 pts]** generate **log(rank) vs log(frequency) plots** for the first 1000 (ranks 1 through 1000) words for BOTH corpora (you can use the matplotlib package or some other plotting package / tool). Place BOTH plots in the table below.

log(rank) vs log(frequency) plots	
Brown	**Reuters**





Did you observe anything interesting when comparing all plots? Write your comments below:

Here is an explanation in my own words:

The log-log plots of word rank versus word frequency follow a Zipfian distribution for both the Brown and Reuters corpora. This means there is a power-law relationship between rank and frequency, with a small number of words occurring very frequently and most words occurring rarely.

For the Brown corpus, there is a steep decline in frequency as rank increases initially. This shows the most common words have very high frequencies. As rank continues increasing, frequency declines linearly on the log scale, indicating the expected pattern of rare words.

Similarly, the Reuters corpus exhibits the same general trend, with a steep drop-off showing the high frequencies of common words, followed by a linear decrease in log frequency as rank gets higher.

The slopes of the log-log plots, which reflect how quickly frequency drops as rank increases, indicate differences in vocabulary diversity between the corpora. The Brown corpus has a steeper slope, meaning frequency drops off more rapidly. This suggests less rich vocabulary compared to Reuters.

In summary, both corpora demonstrate the expected Zipf's law relationship between rank and frequency, but Reuters has a less steep slope, indicating more vocabulary variety compared to the Brown corpus. The core pattern is consistent, but slope differences reflect corpus distinctions.

4) **[15 pts]** use frequency counts obtained earlier to calculate the unigram occurrence probability for the TWO ("technical" and not technical) words. Use lowercasing first! **Display all relevant counts and probability on screen for BOTH corpora (also: enter final values in the table below)**. It can be zero for some words.

"technical" / seldom used in casual conversation word (for example "adiabatic"	
Brown	**Reuters**
Word: MachineLearning Count: 0 Total Words: 686163 Probability: 0.0 Word: ArtificialIntelligence Count: 0 Total Words: 686163 Probability: 0.0	Word: MachineLearning Count: 0 Total Words: 1265276 Probability: 0.0 Word: ArtificialIntelligence Count: 0 Total Words: 1265276 Probability: 0.0
Non- technical / casual / daily-use word (for example "dinner")	
Brown	**Reuters**
Word: Sports Count: 49 Total Words: 686163 Probability: 7.14116033653811e-05 Word: Cooking Count: 32 Total Words: 686163 Probability: 4.663614913657542e-05	Word: Sports Count: 15 Total Words: 1265276 Probability: 1.1855120938040396e-05 Word: Cooking Count: 8 Total Words: 1265276 Probability: 6.322731166954877e-06

Part B [50 pts]:

Use Python's NLTK package along with the Brown corpus for the following tasks:

1. **[1 pts]** Ask the user to enter a sentence S from a keyboard.
2. **[1 pts]** Apply lowercasing to S.
3. **[45 pts]** Calculate P(S) assuming a 2-gram language model (**assume that probability of any bigram starting or ending a sentence is 0.25**)
4. **[3 pts]** Display the sentence S, list all the individual bigrams and their probabilities, and the final probability P(S) on screen. It is fine if it is zero.

Part C [50 pts]:

Use Python's NLTK package along with the Brown corpus (after removing all stop words; use the `stopwords` corpora for that purpose) for the following tasks:

1. **[1 pts]** Start by asking the user for initial word/token W1. Apply lowercasing to W1 (and all future entries). If the word is NOT in the corpus offer two options:
    a. **ask again**
    b. **QUIT**

2. **[45 pts]** Assuming a 2-gram language model, a menu with TOP 3 "most likely to follow W1" words (according to the W1, NEXT WORD probability estimate). For example, if the user started with W1 = "good", the following could be displayed (**NOTE: I made up this selection and corresponding probability estimates**):

```
good …

Which word should follow:
1) morning P(good morning) = 0.25
2) evening P(good evening) = 0.15
3) afternoon P(good afternoon) = 0.14
4) QUIT
```

Repeat (and add subsequent word choices to the "sentence") until user selects (4) and QUITs.

If the user picks a number other than 1,2,3, and 4, **assume user choice is (1)**.