CSCI 4030 CSCI/ DASC 6030: Information Extraction and Retrieval

V.N. Gudivada

17 January 2022

1 Programming Assignment 01: Goal

Develop expertise in basic text pre-processing required for Information Retrieval (IR) applications.

2 Problem Statement

Compute $n-$gram frequencies for documents in a text corpus.

3 Text Corpus

Project Gutenberg offers over 60,000 free eBooks in multiple file formats. For example, the following books are available from Project Gutenberg:

1. austen-emma.txt
2. austen-persuasion.txt
3. austen-sense.txt
4. bible-kjv.txt
5. blake-poems.txt
6. bryant-stories.txt
7. burgess-busterbrown.txt
8. carroll-alice.txt
9. chesterton-ball.txt
10. chesterton-brown.txt
11. chesterton-thursday.txt
12. edgeworth-parents.txt
13. melville-moby-dick.txt
14. milton-paradise.txt
15. shakespeare-caesar.txt
16. shakespeare-hamlet.txt
17. shakespeare-macbeth.txt
18. whitman-leaves.txt

The first word in the file name is the author and the second word/phrase is the name of the literary work. For example, carroll-alice.txt is *Alice's Adventures in Wonderland* (commonly referred to as *Alice in Wonderland*), which is an 1865 novel written by English author Charles Lutwidge Dodgson under the pseudonym Lewis Carroll.

For this programming assignment, you may work with the above list as the corpus or create your own corpus. We will refer to these files as *Gutenberg corpus*.

4 Questions to Investigate Using the Corpus

1. Do all letters in the language occur with the same frequency? If not, which letter has the highest frequency? Which letter has the lowest frequency?
2. Does the Zipf's law hold for *Gutenberg corpus*?
3. Which *unigrams* occur more frequently?
4. Which *bigrams* occur more frequently?
5. Which *trigrams* occur more frequently?

5 Distribution of Letters in English Language

Distribution of letters in the *Gutenberg corpus* (for the files selected in Section~3) should look something similar to the bar graph shown in Figure~1.

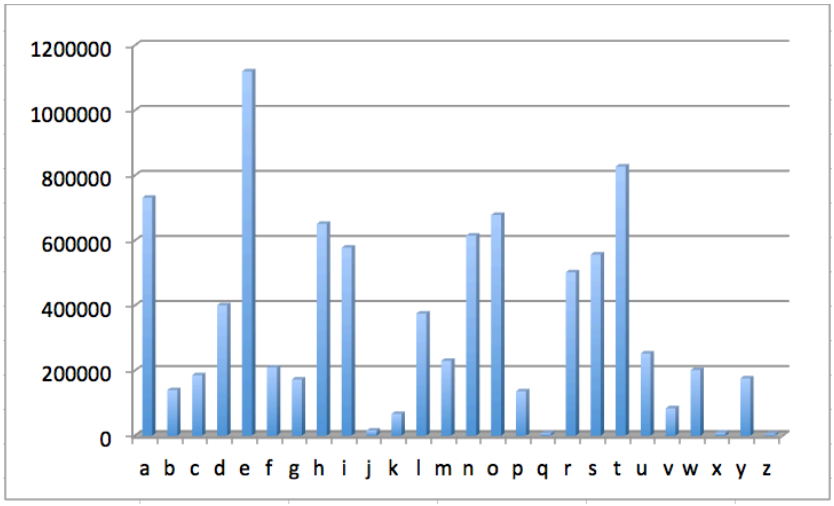

Figure 1: Distribution of letters in English language

6 Zipf's Law

Let us first count the number of occurrences of words in the *Gutenberg corpus*. Then we list the words in the decreasing order of their frequency. For a word w_i, let f_i be its frequency of occurrence and r_i be its rank or position in the sorted list. For example, for the 10^{th} most frequently occurring word, its rank in the list is 10. Zipf's Law states

that for any given word, the product $f_i r_i$ is equal to some constant k. That is, $f_i r_i = k$.

Log-log plot of *frequency of word occurrences* versus their ranks should look similar to the one shown in Figure~2.

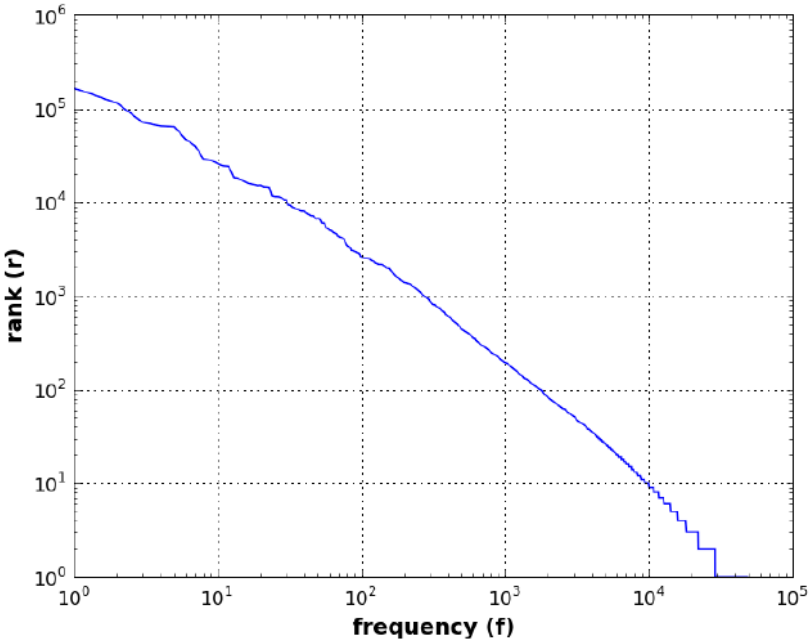

Figure 2: Log-log plot of word frequencies and their ranks for Gutenberg corpus

7 *Assignment Solution Steps*

1. Create a text corpus by downloading select files from Project Gutenberg. You may want to select files in such a way that they all belong to the same *genre*.
2. Read one file at a time from the disk.
    1. For each file:
        1. For each line in the file:
            1. Extract all words in the line.
            2. Keep track of unigrams and their frequency of occurrence
            3. Keep track of bigrams and their frequency of occurrence
            4. Keep track of trigrams and their frequency of occurrence
            5. For each word, extract characters and keep their occurrence count
    2. Close the input file
3. Write characters and their frequency counts to an output file.

4. Using the character frequencies, create a bar graph as shown in Figure~1. You may use a graphing program for this task.

5. For each unigram, compute the product of its frequency and rank. Plot this data as shown in Figure~2. Determine if the Zipf's Law holds for the *Gutenberg corpus*.

6. Sort unigrams in alphabetical order and write them to a disk file. What are the five most frequently occurring unigrams?

7. Sort bigrams in alphabetical order and write them to a disk file. What are the five most frequently occurring bigrams?

8. Sort trigrams in alphabetical order and write them to a disk file. What are the five most frequently occurring trigrams?

8 Assumptions

You may make assumptions about the following:

1. How a word is defined? For example, a word is a sequences of characters from the set {a, ..., z, A, ..., Z} and words are delimited by whitespace.

2. Case is removed by converting all uppercase letters into lowercase.

3. Should we consider grammatical function words such as *the, an, a, that, over, in* or discard them? These words are referred to as *stop words* in the IR literature.

9 Java-specific Knowledge

You will need knowledge of the following Java concepts:

1. Command line arguments to Java programs.

2. Regular expressions and pattern matching using the methods of *Pattern* and *Matcher* classes. Regular expressions define patterns for words and Matcher class finds such patterns in an input.

3. Exception handling using *IOException* class object – how to catch run-time errors and gracefully exit the program rather than crashing.

4. Reading from and writing to disk files using objects of the following classes: *File*, *FileReader*, *BufferedReader*, and *PrintWriter*.

5. Using a dictionary will simplify the code for keeping track of the following along with their frequencies: characters, unigrams, bigrams, and trigrams.

You may implement this assignment in a programming language of your choice. You may not use libraries like NLTK as they will make this assignment trivial. However, you may base your solution on the started code provided by the instructor. The first step then is to understand the instructor's partial solution.

10 Testing the Program

Just because the program compiles and runs for a given set of input values, you cannot conclude that the program solves the problem correctly. How do you prove that your solution is correct? Some of the questions to ask about your code are:

1. What pattern defines a word? Is this pattern general enough to extract all the words?
2. How do you deal with special characters such as quotes and punctuation characters?
3. Does the number of words extracted by your program from, say austen-emma.txt, agree with the number of words given by the *wc* Linux command line program? Is it OK for these counts to differ? Why?
4. What other methods can you think of to validate your solution?

11 Self-assessment

You need to grade your submission using Table 1. The course TA/ instructor will also assign points for the program. Please note that self-assessment is considered as one of the assignment tasks and carries points. If you do not perform self-assessment, you lose points for this part.

12 Hints

1. First document your algorithm(s) in the form of pseudo-code (see Section~7).
2. Manually execute your pseudo-code to validate your solution.
3. Use an *incremental approach* to developing your program from the pseudo-code. Write code for a small logical unit of pseudo-code. Compile and test this code.
4. Work on the next small logical unit of pseudo-code. Repeat this process until all the pseudo-code is translated into real code.
5. Test your program (see Section~10).

Task	Max Points Possible	Points Earned
Program compiles and runs	30	nn
Program seems to produce correct results	20	nn
Testing is done and test results are documented	20	nn
Appropriate data types are chosen for variables	10	nn
Variable names are self-explanatory	10	nn
Program code is commented appropriately	5	n
Self-assessment is performed	5	n
Total points	100	nnn

Table 1: Rubric for self-assessment

13 UNIX/Linux Command Line Solution

We can also use UNIX/Linux command line tools to solve the problem. This solution is intended to show you an alternative way without writing code using a programming language. However, please note that you need to solve the problem by writing a program. You may use the results of this alternative solution to verify the correctness of your solution.

13.1 Extracting Words

Start a terminal window in the Gutenberg corpus directory. The following command will extract all the words across the corpus and displays them in decreasing order of occurrence frequency:

```
cat *.txt | tr -sc 'A-Za-z' '\n' | tr '[:upper:]' '[:lower:]' |
sort | uniq -c | sort -nr > all-words.dat
```

Note that the entire command should be typed on the same line. *all-words.dat* file will have the words in decreasing order of their occurrence frequency.

14 Extracting Characters

```
cat *.txt | grep -o . | grep '[A-Za-z]' | tr "A-Z" "a-z" |
sort | uniq -c | sort -nr > char-counts.dat
```

Note that the entire command should be typed on the same line. *char-counts.dat* file will have the letters in decreasing order of their

occurrence frequency.