Lab 1

Due Thursday, September 28, 11:59 pm

In this first project, you will write the first part of a working IR system. More specifically, you'll be writing a program that tokenizes the input document text. You will also experiment with different tokenization and token normalization techniques to see how these different techniques affect dictionary size. This project along with projects 2 & 3 will implement a basic IR system, which you may choose to augment in your final project to implement an extension that you find interesting (E.g. relevance feedback, etc).

How to approach this assignment:

1. Read this document entirely before you start to work.
2. Login to JupyterHub: https://cs341hub.mtholyoke.edu. Open a python notebook named ***lab01username***, where ***username*** is your username. You'll plan, write, test, and run your code here.
3. Plan how to approach the solution and generate pseudocode before coding.
4. Write one function at a time, testing each one before you move on to the next.
5. After the program is complete and has been tested and debugged, create a markdown cell in your notebook, answer the assignment questions, and write your answers in that cell.
6. Submit your code and your answers to the questions.

Part 1: Preprocessing

- You'll be writing your program in Python in a Jupyter Notebook which can be used from any desktop or laptop. If you're unfamiliar with Jupyter Notebook, you can get some practice with this tutorial: https://docs.google.com/document/d/1FoBm_uMHOFq2N9pmtuHxrQ9aE_-RAvlsopUq3PUrF5w/edit?usp=sharing
- Download the data and instructions from the assignment folder on moodle. It will contain the following files:
  - Lab01 assignment description

  - Collection 1: bible.txt (Bible text)
  - Collection 2: quran.txt (English translation of quran text)
  - Collection 3: wiki.txt. (Wikipedia abstracts). This file is too large for Moodle, so download from the drive here: https://drive.google.com/file/d/1Zx5aHMXHee8RObQKd0PZ1j8PWQAUUUs_/view?usp=sharing

You'll use the following two collections in Part 4:
- Collection 4.1: amuzgoan.txt (extract of bible text written in Amuzgoan)
- Collection 4.2: wiki.txt (extract of bible text written in Maori)

- Write code to perform the tasks below. All of the code with the exception of the stemmer, should be implemented without the use of library functions defined for the task e.g. do not use an existing parser, tokenizer, or stopword remover. The tasks to implement follow:
  - **Tokenization**: convert text into tokens with no punctuation. I recommend that you check the output to make sure things are working as you expect before proceeding. If you'd like to use regular expressions for this you'll find a nice tutorial here: https://regexone.com/
  - **Case folding**: make all text into lower case. As a reality check, bible.txt should contain roughly 13K unique tokens after this step. quran.txt should have about half as many and wiki.txt more than 1 million. Note that your numbers will vary to some extent based on the decisions you made with respect to tokenizing.
  - **Stopping**: remove English stop words (file in starter folder).
  - **Normalization**: Porter stemmer at least. You can try other stemmers as well. You can get Porter stemmer in Python, or you can use Snowball Stemmer which has versions for multiple languages.
- Print new files for collection 1, 2 and 3 after preprocessing.
- Compare the processed file to the new file. Are there any surprises? Discuss what kind of modifications in preprocessing could be applied. For example:
  - Additional words/terms to be filtered out
  - Special tokenization
  - Additional normalization to some terms

Part 2: Text Laws

For each of the three collections, do the following:

1. Print the unique terms with frequency, then plot them in a log-log graph. Report what you notice about the graphs for each collection. What does this suggest about Zipf's law?
2. Plot the growth of vocabulary while you go through the collection and observe Heap's law. When plotting Heaps Law, we need to calculate the number of different terms that occur up to any position within our dataset. We need to construct an array of unique term counts for each token position. For example, consider the input text "a b c a b e". The corresponding term count data is [1, 2, 3, 3, 3, 4]; Note that the term count is only updated when we see a unique token (here at the first occurrence of "a", "b", "c", and "e".

   Advice on how to implement:
   - read text file term by term, keeping count *n* (the number of terms read so far).
   - save new terms in a hash as you read the file. With each new term update the

vocabulary size *v*.
- print the values of *n* and *v* every once in while. Plot *n* vs *v* at the end.

Part 3: Plotting data with pyplot

Now let's plot (Your final write-up should contain graphs with labels). Pyplot is a module of Matplotlib, which provides functions for simple elements such as lines, images, text, etc. A figure can contain one or more plots. If you have not used pyplot before, the instructions below will familiarize you with plotting.

1. Open the file *plotData.py*
2. Notice the two import statements importing the Pyplot module and the Numpy library (lets us work with data in the form of arrays):

   import matplotlib.pyplot as plt
   import numpy as np

3. Just to get familiar with plotting, lets add the following to the file:

   plt.plot([1,2,3,4],[1,4,9,16])
   plt.show()

   Save the file and run it from the terminal. A graph should be displayed in a pop-up window. We passed 2 arrays as input arguments to Pyplot's plot() method to generate a plot of our data. The show() method displays the plot. Notice that the first array appears on the x-axis and the second array appears on the y-axis. Note that our plot doesn't contain any labels or title.

4. Click the little red x in the upper left corner of the plot. Let's add a title and labels by employing the title(), xlabel(), and ylabel() methods. Add the following code **after** the invocation of the plot() method and before invoking show():

   plt.title("My Plot")
   plt.xlabel("X label")
   plt.ylabel("Y label")

   Save the file and run the program.

   Note: For more details on how to add labels, etc use this link):
   https://towardsdatascience.com/matplotlib-tutorial-learn-basics-of-pythons-powerful-plotting-library-b5d1b8f67596

5. We can also specify the size of the graph generated using the figure() method. we pass the values of the length of rows and columns as a tuple to the argument figsize. Add the following before the invocation of the plot() method.

   plt.figure(figsize=(15,5))

   Save the file and run the program.

6. Now let's experiment with setting the color and line type of the plot (this is especially useful when you have multiple plots on the same figure). With each X and Y argument, you can pass a third optional argument in the form of a string that indicates the plot's color and line type. The default when no color or type is specified is **b-**, meaning a blue solid line. Modify the invocation of the plot() method to create a line of green circles:

   plt.plot([1,2,3,4],[1,4,9,16],"go")

   Save the file and run the program

7. We can plot multiple sets of data by passing in multiple sets of arguments of X and Y axis in the plot() method invocation as shown below:

   x = np.arange(1,5) # return evenly spaced value from start to end-1
   y = x3** # cube x to get y's value
   plt.plot([1,2,3,4],[1,4,9,16],"go", x, y, "r^") # plot x and y with red triangles
   plt.title("My Plot")
   plt.xlabel("X label")
   plt.ylabel("Y label")
   plt.show()

8. Add legends to the figure by passing a tuple of legend strings to the legend() method. Try this by adding the following line after the invocation of plot():

   plt.gca().legend(('x^2','x^3'))

9. Alternatively, we can create each plot using multiple invocations of the plot() method and passing the legend text string as a final argument to plot(). Invoke show() only once after all invocations of plot():

```
plt.figure(figsize=(15,5))
x = np.arange(1,5)
y = x**3
plt.plot([1,2,3,4],[1,4,9,16],"-.b",label="x^2")
plt.plot(x,y,"--r",label="x^3")
```

```
plt.legend()
plt.title("My Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```

10. We can also create a line plot. This is done similarly to the way we specified the shape and color of our data points in the previous example by using a string to specify the line type and the color. For example “-b” means unbroken blue line. A dashed line is represented as “--" and we can create a line with unequal length segments with “-.“. Modify your code to create an unbroken blue line for the first dataset and a red dashed line for the second dataset in the code from (7). Save the code and run it.
11. When you are finished exploring different line types, plot the Heaps data you collected in part 2.

Part 4: Zipf's LAW

Next, we want to plot Zipf's law.

1. Look at the contents of amuzgo.txt and maori.txt.
2. Both contain extracts of the Bible. The first is written in Amuzgoan (an Oto-Manguean language of Mexico) and the second is in Maori (a Malayo-Polinesian language). The fact that your Amuzgoan and Malayo-Polinesian language skills might only be rudimentary, does not prevent you from working with texts written in those languages.
3. Pre-process the text files and count the frequency of each of the terms in the two files.
4. Then plot the terms sorted in descending order by their frequency and look at the result. Use plt.loglog() instead of plt.plot().
5. Look at the plots of Zipf's Law for both languages. What conclusion can you draw from your findings?
6. How do these plots compare to that for the 3 English collections?

Submitting your work

Write up a brief, but concise lab report using the graphs to explain your findings and place it in a Markdown cell at the bottom of your notebook. Is there anything surprising about them? Are they consistent with the readings and videos? Why or why not? Include a discussion about the tokenizing process e.g. were there surprises in the output? What kinds of tokens were there that were not tokenized as you would have liked? Can you think of ways to “fix” them? Upload your lab01*username* file to moodle.

Useful tips

You can print the frequency of unique terms in a given collection by typing the unix command line below. Note that you should replace *text.file* with the name of your input file and terms.freq will be the output filename (you can select a different name if you prefer.

- cat *text.file* | tr '[:punct:]' " "|tr " " "\n" | tr "A-Z" "a-z" | sort | uniq -c | sort -n > *terms.freq*