Lab 1: Tokenization and Text Normalization

This Lab has five parts, and is due at noon on Friday, September 8. For this assignment, links to two electronic files have been placed under Assignments->Lab 1 in Canvas: tokens.txt.gz and sentences.txt.gz.

(a) **Tokenization**. Write a function(s) to tokenize and normalize English text. The goal is to take an input line of text (think *sentence*) and return an ordered list of normalized tokens. Most tokens are words, but you will also find numbers, punctuation, and other strings. Some common processes include separating punctuation from words, case-normalizing (i.e., lower-casing), and splitting tokens by whitespace. Do not remove stopwords and do not apply stemming. This process might produce some undesirable tokens, or might break up good tokens into worse tokens. Examine the results of your initial processing and try to improve the results. For this part you may not use any built-in libraries to perform tokenization; we want you to implement your own approach and in doing so learn about the variability of natural language.

Below is an example tokenization for a sample sentence. Your tokenizer need not produce the same results.

> **Input:** NAC has developed a National HIV/AIDS/STI/TB Intervention Strategic Plan (2002-2005) that aims to reduce the HIV prevalence rate among Zambians from 19.3% to 11.7% and improve the health status of people living with HIV/AIDS by 2005.
>
> **Output:** ['nac', 'has', 'developed', 'a', 'national', 'hiv', '/', 'aids', '/', 'sti', '/', 'tb', 'intervention', 'strategic', 'plan', '(', '2002', '-', '2005', ')', 'that', 'aims', 'to', 'reduce', 'the', 'hiv', 'prevalence', 'rate', 'among', 'zambians', 'from', '19.3', '%', 'to', '11.7', '%', 'and', 'improve', 'the', 'health', 'status', 'of', 'people', 'living', 'with', 'hiv', '/', 'aids', 'by', '2005', '.']

- 2 points: Describe your processing.
- 2 points: Describe any common sources of error or undesirable results in your initial approach; explain how you addressed them – if you did; and indicate what undesirable results might remain.
- 1 point: Show the results of processing the first ten lines of the tokens.txt input file.
- 4 points: Our assessment of the quality of your tokenization.

(b) **Corpus statistics**. After refining your tokenization method, use it to process the entire tokens.txt file and collect the following statistics based on the tokens that are observed. You can and should compute these statistics in one pass over the input file. It is easy to store counts (i.e., frequencies) for tokens in a dictionary or hashtable.

- 1 point: Report the number of lines you processed.
- 2 points: Report the number of unique tokens observed (vocabulary size), and the total number of tokens encountered (collection size). Unique tokens are sometimes called *types*.
- 4 points: Calculate the total number of times each unique token is seen (*collection frequency* of the type). Then sort the types by frequency and list the most frequent types at ranks 1 through 100 and at ranks 500, 1,000, 5,000, and 10,000. **DO NOT** just provide a list of the top 10,000 most common tokens.
- 3 points: Calculate and report the *number* of tokens that occur exactly one time and the percentage of the vocabulary that such words constitute. Such words are known as *hapex legomena*. We do not want to see the words themselves. Your answer here should be just a single count and its percentage (e.g., "5,000 words or 10.36% of vocabulary").

Sample Output: Here is a sample output if the input had been the full text of *Sense and Sensibility*:

```
1862        Number of lines
6336        Number of unique tokens (vocabulary size)
120785 Number of tokens (collection size)
          1. to          4116
          2. the              4104
          3. of          3572
          4. and              3491
          5. her              2551
                ...
        100. lucy             186
        500. appearance       28
       1000. couple           12
       5000. released         3
      10000. workmen                1
2449        Number of singleton terms
38.652%     Percentage of singletons
```

(c) **Zipf's Law**. Zipf's law predicts that the number of times the i^{th} most frequent word will be seen is about k/i times the frequency of the most common word, for *some k*. Put another way, Zipf's law states that the product of a term's frequency and its rank is roughly constant. Create a graphical plot of frequency (vertical) vs. rank (horizontal) for the vocabulary terms, and determine whether Zipf's law holds for this dataset. You can create a plot programmatically (e.g., using Python's matplotlib) or using a separate tool (e.g., MS Excel or Google Sheets). Please produce a log-log plot (i.e., both axes should use a logarithmic scale).

- 3 points: Create a plot that illustrates the relationship between frequency and rank.
- 1 point: Explain whether the plotted data follows Zipf's law, and how you reach that determination.

(d) **Sentence Boundary Detection**. In English, most sentences are terminated by a final punctuation symbol such as period, exclamation point, or question mark. However, not every such symbol is indicative of a sentence boundary. For example, English abbreviations can be followed by a non-sentence-ending period. Periods are also used in numbers, and tokens like "Yahoo!" use punctuation that doesn't reliably indicate the end of a sentence. We have provided a separate file in which between one and ten sentences are found on each line. Write a program that reads in such lines and for each line outputs n (the number of found sentences), and reports the n distinct zero-based character offsets of the final sentence character. A sentence might end at a final punctuation symbol, or at another character. For example, the sentence (in angle brackets) <I 'conquered.'> ends in a quote character. Name your output file `ID.txt` (using your JHED ID) and submit it in Canvas.

Example input line (in angle brackets): <I came. I saw! I conquered>. Possible output line (in angle brackets): <3 6 13 25>. The first 3 indicates that three sentences were found; offset 6 is for the period; offset 13 is for the exclamation point; offset 25 is the final letter `d`, since the line ended without punctuation. Note that these offsets are zero-based; the first `I` on the line is offset 0, the first space is offset 1, etc. Your output file should contain the same number of lines as the input file `sentences.txt`. You should test your code on this example (not including the angle brackets) and verify that it produces the correct output line before you submit your work.

- 3 points: Provide an output file with the correct number of lines in the correct format.
- 5 points: Our assessment of the quality of results on selected test sentences.
- 3 points: Estimate and report the accuracy of your program in finding sentence boundaries. Don't just guess; justify your answer numerically.

A validator for this part of the Lab called `lab1-validator.py` is available on Canvas under this Module. You should run the validator on your output file and make sure it passes prior to submission:

```
python3 lab1-validator.py <your-submission-file> <source-file-being-analyzed>
```

(e) **NLTK**. NLTK is a toolkit for natural language processing in Python. There is a popular O'Reilly book about using NLTK, available from <https://www.nltk.org/book>. The toolkit has built-in functionality for tokenizing text and for sentence splitting. Work through most of the tutorial (Chapter 3 of NLTK book, at <https://www.nltk.org/book/ch03.html>). Be sure to learn how to use word_tokenize and sent_tokenize. Then use these NLTK methods to process the input files (tokens.txt and sentences.txt) described above.

- 3 points: Compare the quality of NLTK's tokenization output to yours. Describe any significant differences between the two.
- 3 points: Compare NLTK's sentence splitting output with the output of method that you implemented. Again, describe any notable differences.

If you have questions about what is expected for this assignment, please post them to the Canvas discussion forum. You should make a good faith effort when writing your tokenizer and sentence splitter, but attaining perfection is not the objective.

There are 40 possible points for this lab. You should submit a) your source code, b) a file JHEDID.txt that contains your sentence boundary predictions; and c) the other items requested herein, together as a single PDF file. If you are familiar with Jupyter Notebooks, placing this information in a single notebook and exporting it as PDF should work well (although you should check to ensure that no lines are truncated).