CSCI 1100 — Computer Science 1
Homework 6
Files, Sets and Document Analysis

Overview

This homework is worth **100 points** total toward your overall homework grade. It is due Thursday, November 07 , 2024 at 11:59:59 pm. As usual, there will be a mix of autograded points, instructor test case points, and TA graded points. There is just one "part" to this homework.

See the handout for Submission Guidelines and Collaboration Policy for a discussion on grading and on what is considered excessive collaboration. These rules will be in force for the rest of the semester.

You will need the data files we provide in `hw6_files.zip`, so be sure to download this file from the `Course Materials` section of Submitty and unzip it into your directory for HW 6. The zip file contains data files and example input / output for your program.

Problem Introduction

There are many software systems for analyzing the style and sophistication of written text and even deciding if two documents were authored by the same individual. The systems analyze documents based on the sophistication of word usage, frequently used words, and words that appear closely together. In this assignment you will write a Python program that reads two files containing the text of two different documents, analyzes each document, and compares the documents. The methods we use are simple versions of much more sophisticated methods that are used in practice in the field known as natural language processing (NLP).

Files and Parameters

Your program must work with three files and an integer parameter.

The name of the first file will be `stop.txt` for every run of your program, so you don't need to ask the user for it. The file contains what we will refer to as "stop words" — words that should be ignored. You must ensure that the file `stop.txt` is in the same folder as your `hw6_sol.py` python file. We will provide one example of it, but may use others in testing your code.

You must request the names of two documents to analyze and compare and an integer "maximum separation" parameter, which will be referred to as `max_sep` here. The requests should look like

```
Enter the first file to analyze and compare ==> doc1.txt
doc1.txt
Enter the second file to analyze and compare ==> doc2.txt
doc2.txt
Enter the maximum separation between words in a pair ==> 2
2
```

Parsing

The job of parsing for this homework is to break a file of text into a single list of consecutive words. To do this, the contents from a file should first be split up into a list of strings, where each string contains consecutive non-white-space characters. Then each string should have all non-letters removed and all letters converted to lower case. For example, if the contents of a file (e.g. `doc1.txt`) are read to form the string (note the end-of-line and tab characters)

```
s = " 01-34  can't   42weather67  puPPy, \r \t and123\n  Ch73%allenge 10ho32use,.\n"
```

then the splitting should produce the list of strings

```
['01-34', "can't", '42weather67', 'puPPy,', 'and123', 'Ch73%allenge', '10ho32use,.']
```

and this should be split into the list of (non-empty) strings

```
['cant', 'weather', 'puppy', 'and', 'challenge', 'house']
```

Note that the first string, `'01-34'` is completely removed because it has no letters. All three files — `stop.txt` and the two document files called `doc1.txt` and `doc2.txt` above — should be parsed this way.

Once this parsing is done, the list resulting from parsing the file `stop.txt` should be **converted to a set**. This set contains what are referred to in NLP as "stop words" — words that appear so frequently in text that they should be ignored.

The files `doc1.txt` and `doc2.txt` contain the text of the two documents to compare. For each, the list returned from parsing should be further modified by removing any stop words. Continuing with our example, if `'cant'` and `'and'` are stop words, then the word list should be reduced to

```
['weather', 'puppy', 'challenge', 'house']
```

Words like `and` are almost always in stop lists, while `cant` (really, the contraction `can't`) is in some. Note that the word lists built from `doc1.txt` and `doc2.txt` should be kept as lists because the word ordering is important.

Analyze Each Document's Word List

Once you have produced the word list with stop words removed, you are ready to analyze the word list. There are many ways to do this, but here are the ones required for this assignment:

1. Calculate and output the average word length, accurate to two decimal places. The idea here is that word length is a rough indicator of sophistication.

2. Calculate and output, accurate to three decimal places, the ratio between the number of distinct words and the total number of words. This is a measure of the variety of language used (although it must be remembered that some authors use words and phrases repeatedly to strengthen their message.)

3. For each word length starting at 1, find the set of words having that length. Print the length, the number of **different** words having that length, and at most six of these words. If for a certain length, there are six or fewer words, then print all six, but if there are more than six print the first three and the last three in alphabetical order. For example, suppose our simple text example above were expanded to the list

```
['weather', 'puppy', 'challenge', 'house', 'whistle', 'nation', 'vest',
 'safety', 'house', 'puppy', 'card', 'weather', 'card', 'bike',
 'equality', 'justice', 'pride',  'orange', 'track', 'truck',
 'basket', 'bakery', 'apples', 'bike', 'truck', 'horse', 'house',
 'scratch', 'matter', 'trash']
```

Then the output should be

```
1:   0:
2:   0:
3:   0:
4:   3: bike card vest
5:   7: horse house pride ... track trash truck
6:   7: apples bakery basket ... nation orange safety
7:   4: justice scratch weather whistle
8:   1: equality
9:   1: challenge
```

4. Find the distinct word pairs for this document. A word pair is a two-tuple of words that appear `max_sep` or fewer positions apart in the document list. For example, if the user input resulted in `max_sep == 2`, then the first six word pairs generated will be:

   ```
   ('puppy', 'weather'), ('challenge', 'weather'),
   ('challenge', 'puppy'), ('house', 'puppy'),
   ('challenge', 'house'), ('challenge', 'whistle')
   ```

   Your program should output the total number of distinct word pairs. (Note that `('puppy', 'weather')` and `('weather', 'puppy')` should be considered the same word pair.) It should also output the first 5 word pairs in alphabetical order (as opposed to the order they are formed, which is what is written above) and the last 5 word pairs. You may assume, without checking, that there are enough words to generate these pairs. Here is the output for the longer example above (assuming that the name of the file they are read from is `ex2.txt`):

   ```
   Word pairs for document ex2.txt
     54 distinct pairs
     apples bakery
     apples basket
     apples bike
     apples truck
     bakery basket
     ...
     puppy weather
     safety vest
     scratch trash
     track truck
     vest whistle
   ```

5. Finally, as a measure of how distinct the word pairs are, calculate and output, accurate to three decimal places, the ratio of the number of distinct word pairs to the total number of word pairs.

Compare Documents

The last step is to compare the documents for complexity and similarity. There are many possible measures, so we will implement just a few.

Before we do this we need to define a measure of similarity between two sets. A very common one, and the one we use here, is called *Jaccard Similarity*. This is a sophisticated-sounding name for a very simple concept (something that happens a lot in computer science and other STEM disciplines). If $\mathcal{A}$ and $\mathcal{B}$ are two sets, then the Jaccard similarity is just

$$J(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A} \cap \mathcal{B})|}{|\mathcal{A} \cup \mathcal{B})|}. \tag{1}$$

In plain English it is the size of the intersection between two sets divided by the size of their union. As examples, if $\mathcal{A}$ and $\mathcal{B}$ are equal, $J(\mathcal{A}, \mathcal{B}) = 1$, and if $\mathcal{A}$ and $\mathcal{B}$ are disjoint, $J(\mathcal{A}, \mathcal{B}) = 0$. As a special case, if one or both of the sets is empty the measure is 0. The Jaccard measure is quite easy to calculate using Python set operations.

Here are the comparison measures between documents:

1. Decide which has a greater average word length. This is a rough measure of which uses more sophisticated language.
2. Calculate the Jaccard similarity in the overall word use in the two documents. This should be accurate to three decimal places.
3. Calculate the Jaccard similarity of word use for each word length. Each output should also be accurate to three decimal places.
4. Calculate the Jaccard similarity between the word pair sets. The output should be accurate to **four** decimal places. The documents we study here will not have substantial similarity of pairs, but in other cases this is a useful comparison measure.

See the example outputs for details.

Notes

- An important part of this assignment is to practice with the use of sets. The most complicated instance of this occurs when handling the calculation of the word sets for each word length. This requires you to form a list of sets. The set associated with entry `k` of the list should be the words of length `k`.
- Sorting a list or a set of two-tuples of strings is straightforward. (Note that when you sort a set, the result is a list.) The ordering produced is alphabetical by the first element of the tuple and then, for ties, alphabetical by the second. For example,

```
>>> v = [('elephant', 'kenya'), ('lion', 'kenya'), ('elephant', 'tanzania'), \
         ('bear', 'russia'), ('bear', 'canada')]
>>> sorted(v)
[('bear', 'canada'), ('bear', 'russia'), ('elephant', 'kenya'), \
 ('elephant', 'tanzania'), ('lion', 'kenya')]
```

- Submit just a single Python file, `hw6_sol.py`.
- A component missing from our analysis is the frequency with which each word appears. This is easy to keep track of using a dictionary, but we will not do that for this assignment. As you learn about dictionaries think about how they might be used to enhance the analysis we do here.

Document Files

We have provided the example described above and we will be testing your code along with several other documents (few of them are):

- Elizabeth Alexander's poem *Praise Song for the Day*.
- Maya Angelou's poem *On the Pulse of the Morning*.
- A scene from William Shakespeare's *Hamlet*.
- Dr. Seuss's *The Cat in the Hat*
- Walt Whitman's *When Lilacs Last in the Dooryard Bloom'd* (not all of it!)

All of these are available full-text on-line. See `poetryfoundation.org` and learn about some of the history of these poets, playwrites and authors.