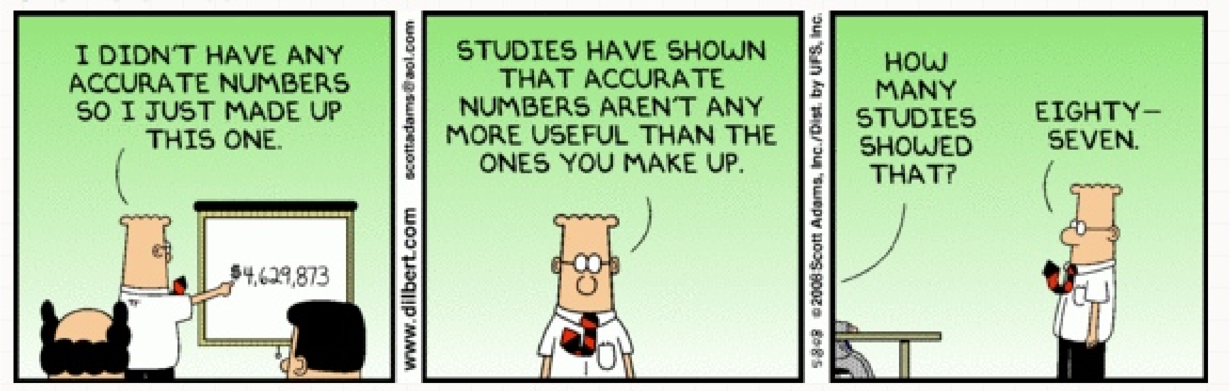
CS51A—Assignment 5
Corpus Analysis

Due: Sunday, February 27, at 11:59pm


I DIDN'T HAVE ANY ACCURATE NUMBERS SO I JUST MADE UP THIS ONE.
$4,629,873
scottadams@aol.com
www.dilbert.com
STUDIES HAVE SHOWN THAT ACCURATE NUMBERS AREN'T ANY MORE USEFUL THAN THE ONES YOU MAKE UP.
© 2008 Scott Adams, Inc./Dist. by UFS, Inc.
5-8-08
HOW MANY STUDIES SHOWED THAT?
EIGHTY-SEVEN.

https://dilbert.com/strip/2008-05-08

For this assignment, we're going to be doing some basic corpus analysis. A corpus is a collection of documents/texts. By looking at different characteristics of those texts, in aggregate, we can often start to understand more about the language used in the corpus.

You may (and we would encourage you to) work with a partner on this assignment. **If you do, you must work with a different partner than you did for assignment 4**.

Additionally, you must both be there when either of you are working on the project and you should only be coding on one computer (i.e. pair programming). If you would like a partner, but don't have one, talk to Prof. Ye or a TA in lab and we can try and pair people up.

Course Feedback

We're about a third of the way through the course. We wanted to check in with all of you, see how things are going, and see if there are things that we can collaboratively do to improve the course and your learning experience. Please take a few minutes to fill out this anonymous course feedback form:

`https://forms.gle/URvwz8zdaDJwFy1d9`

We'll go over the feedback in class next week.

In general, if you ever have feedback or ways in which the course could be improved, please do let us know!

Data

To give us a bit of variety (and allow us to make comparisons) we're going to be looking at text from a few different sources. Follow the same procedure you've done for the last two assignments for downloading the starter:

- Download the following file and unzip it by double-clicking on it:

  `https:/cs.pomona.edu/classes/cs51a/assignments/assign5-starter.zip`

- Delete the zip file. You should now have a folder called `assignment5` with four files.

  - `test.txt`: A simple test file for you to test your functions on before trying them on the full data.
  - `sam.txt`: Text fom Dr. Suess's "Green Eggs and Ham". Mostly just for fun :)
  - `normal.txt`: 100K sentences from English Wikipedia[1].
  - `simple.txt`: 100K sentences from Simple English Wikipedia[2], which contains similar content as English Wikipedia, but are written to be more broadly accessible.

  Open up a couple of them and take a look at what's inside. Each of these files is a text file with one sentence per line.

- Open up PyCharm and create a new project, with a file called `assign5.py` in it. Copy the files from the starter into this project folder. Add your name and the assignment number in comments to the top of this file.

Overview

Our goal for this assignment is to write a program that takes in a file and outputs some statistics about the text in the file. Specifically, the following would be printed out if we ran our program on the test data:

[1] https://en.wikipedia.org

[2] https://simple.wikipedia.org

```
------------------------------
Total sentences:      4
Longest sentence:     6
Shortest sentence:    4
Ave. sentence length: 4.75
------------------------------
Top ten most frequent words
1: this    3
2: sentence     3
3: is      2
4: a       2
5: does    2
6: another 1
7: count   1
8: as      1
9: i       1
10: think  1
```

Sentence Stats

1. [**3 points**] Once we have sentences, the first step when dealing with text is often to break the sentence down into words (aka tokens). Strings have a method called `split` that splits up a string into a list based on whitespace characters, however, this is too simple for real applications. For example, if we split one of the first few sentences from `sam.txt`:

   ```
   >>> "I would not, could not, in a car.".split()
   ['I', 'would', 'not,', 'could', 'not,', 'in', 'a', 'car.']
   ```

   we see that this doesn't handle punctuation correctly, i.e. "car" is a word but instead we receive "car.".

   Write a function called `tokenize` that takes as input a string (representing a sentence) and returns the words in the sentence, lowercased. We'll take a simple approach to this that captures most of the common cases:

   - First split the sentence into "basic words" based on whitespace.
   - Then, for each of these basic words, if it ends in one of:

     ```
     . , ; : ! ?
     ```

     remove that character from the end of the word.

   Hint: Like many of the functions we have seen so far, the easiest way to do this is to iterate over the "basic words" and build up a new list with your final collection of words.

   For example:

```
>>> tokenize("I would not, could not, in a car.")
['i', 'would', 'not', 'could', 'not', 'in', 'a', 'car']
```

2. [**2 points**] Write a function called get_sentence_lengths that takes as input a *filename* and returns a list of the number of words in each line/sentence in the file after tokenizing (i.e. calling tokenize). For example:

   ```
   >>> get_sentence_lengths("test.txt")
   [4, 4, 6, 5]
   ```

   These correspond to the lengths of the tokenized lists returned by calls for each line in the file, i.e.

   ```
   >>> tokenize("This is a sentence.")
   ['this', 'is', 'a', 'sentence']
   >>> tokenize("This is another sentence.")
   ['this', 'is', 'another', 'sentence']
   >>> tokenize("Does this count as a sentence?")
   ['does', 'this', 'count', 'as', 'a', 'sentence']
   >>> tokenize("I think, yes, it does.")
   ['i', 'think', 'yes', 'it', 'does']
   ```

3. [**2 points**] Write a function called print_sentence_stats that takes a *filename* (as a string) as a parameter and *prints* out four statistics about the sentence in the file: 1) the total number of sentences, 2) the length of the longest sentence, 3) the length of the shortest sentence, and 4) the average sentence length. Your results be formatted with the text label, followed by a tab character (\t) and then the number, i.e.:

   ```
   Total sentences:      4
   Longest sentence:     6
   Shortest sentence:    4
   Ave. sentence length: 4.75
   ```

   (This is the output from test.txt)

   Hint: Use your get_sentence_lengths function!

Word Stats

The second part of the analysis looks at the frequency of words, specifically, finding the top ten most frequent words in the corpus. We're going to accomplish this in a few steps:

- First, we'll create a dictionary with all of the word frequencies, specifically, where the key is the word and the value is the frequency of that work in the corpus.

- Given this dictionary, we can traverse all of the entries and find the word with the highest frequency. This will be the first entry in our top ten list.

- To get the next entry, we will *delete* the entry with the highest frequency from the dictionary. If we then search again for the word with the highest frequency, we'll get the second most frequent word. We'll repeat this process until we have the top 10 words.

4. [**3 points**] Write a function called `add_words_to_dict` that takes two parameters as input: 1) a dictionary where the key is a string (i.e., a word) and the value is an int representing the current count of that word and 2) a list of strings representing words. The function should increment the count in the dictionary of each of the words in the list by 1. Specifically, if the word is not in the dictionary yet, it should add it with the value 1. If it is, it should update the value to be one larger. Note, this function mutates the dictionary and does *not* return a value.

```
>>> test_dict = {}
>>> add_words_to_dict(test_dict, ["a", "b", "b", "c"])
>>> test_dict
{'a': 1, 'b': 2, 'c': 1}
>>> add_words_to_dict(test_dict, ["a", "c", "c"])
>>> test_dict
{'a': 2, 'b': 2, 'c': 3}
>>> add_words_to_dict(test_dict, ["a"])
>>> test_dict
{'a': 3, 'b': 2, 'c': 3}
```

5. [**3 points**] Write a function called `get_word_counts` that takes a filename (as a string) as input and creates a dictionary of the words frequencies in the file. You should tokenize each line of the file. *Hint:* Don't work too hard in this function. Most of the work has already been done. Specifically, think about how both `tokenize` and `add_words_to_dict` can be used in this function.

```
>>> get_word_counts("test.txt")
{'this': 3, 'is': 2, 'a': 2, 'sentence': 3, 'another': 1, 'does': 2, 'count': 1,
 'as': 1, 'i': 1, 'think': 1, 'yes': 1, 'it': 1}
```

6. [**3 points**] Write a function called `dict_count_max` that takes a dictionary of word frequencies as input and returns the word in the dictionary with the highest frequency.

7. [**3 points**] Write a function called `print_top_ten` that takes a filename (as a string) as input and prints the top ten most frequent words, along with their frequencies (see formatting in the example above).

   Hints:

   - Use your `get_word_counts` function.

- To print the top 10, find the most frequent word using `dict_count_max`, print it out, and then delete it. Repeat this 10 times.
- To remove an item from a dictionary, use the `pop` method, which takes the key to remove as input. Here's an example of using `pop` in the python shell:

  ```
  >>> test = {"a":1, "b":10, "c":5}
  >>> test
  {’a’: 1, ’b’: 10, ’c’: 5}
  >>> test.pop("b")
  10
  >>> test
  {’a’: 1, ’c’: 5}
  ```

8. [**1 points**] Finally, write a function called `print_all_stats` that takes a filename (as a string) as input and prints the sentence level and word level stats formatted as seen in the example above (in the Overview section)

Analysis

Now we can use our functions to analyze a few corpora. At the end of your file, add the following delimiter:

```
# --------------------------------------------------
# Analysis section
```

9. [**2 points**] Run your program on both `normal.txt` and `simple.txt` and include their output as a triple-quoted string. Additionally, include 2-3 sentences analyzing the differences, i.e. what does the data imply about the different sources? Does it make sense?
10. [**3 points**] Do one additional analysis of your own choosing that compares the two corpora. You must write at least one function to help you with this analysis. Include the quantitative results for both corpora along with 1-2 sentences talking about the results. Here are some suggestions, though feel free to get creative:

    - Vocabulary size, i.e. the total number of *unique* words.
    - Average word length.
    - Use of punctuation, e.g. the number of commas per sentence.

When you're done

Make sure that your program is properly commented:

- You should have comments at the very beginning of the file stating your name, course, assignment number and the date.
- You should have comments delimiting the two sections.
- Each function should have an appropriate docstring.
- Include other miscellaneous comments to make things clear.

In addition, make sure that you've used good *style*. This includes:

- Following naming conventions, e.g. all variables and functions should be lowercase.
- Using good variable names.
- Good use of booleans. You should NOT have anything like:)

  ```
  if boolean_expression == True:
  ```

  or

  ```
  if boolean_expression == False:
  ```

  instead use:

  ```
  if boolean_expression:
  ```

  or

  ```
  if not (boolean_expression): # or some other way of negating the expression
  ```

- Proper use of whitespace, including indenting and use of blank lines to separate chunks of code that belong together.
- Make sure that none of the lines are too long.

1 Ethics

Trigger Warning: The article talks about a suicide hotline. If this a topic you are uncomfortable reading about, please submit a pdf indicating that "I did not complete the reading according to the trigger warning". You can skip reading to the next section.

For this week, we will read a recent Politico article that showed how an AI-driven crisis text system gathered data that it shared with for-profit spinoffs. https://www.politico.com/news/2022/01/28/suicide-hotline-silicon-valley-privacy-debates-00002617 The article raises not only the usual concerns that we have seen about biased AI but also implications about privacy. Please summarize the article and discuss something that caught your attention.

When you're done

You will need to submit your `.py` file (and not any of the starter code that you downloaded) along with a pdf of your response for the ethics prompt. Submit them using the courses submission mechanism under "assign5". If working with a partner, proceed with a single submission for your code; both partners need to upload a pdf response, clearly indicating their name.

Grading

	points
Sentence Stats	7
Word Stats	13
Analysis	5
Comments, style	3
ethics prompt	1
total	29