Project 1: TextAnalyzer: 200 Points

Due March 11

Introduction

In this project...

The code

Grading

Tips

Extra Credit: Calculating similarity

Miscellaneous

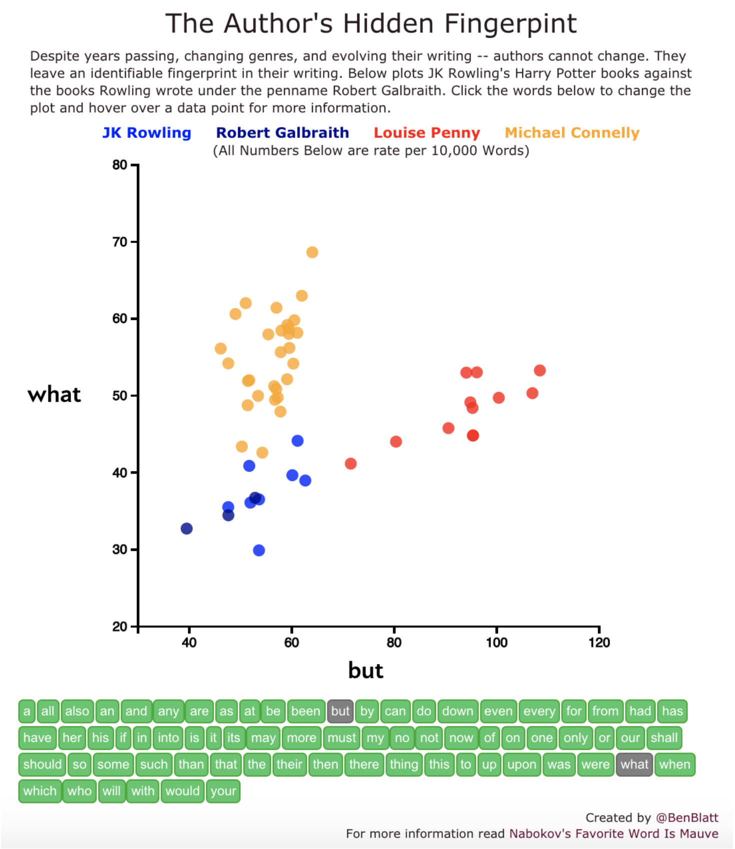
Introduction

What if there was a way to identify the author of an anonymous text? In this project, we will build the infrastructure for analyzing texts so they can be compared for similarity.

Data scientists have successfully achieved this by comparing the frequency of common words in an author's writings. These frequencies form a descriptor of an author's style, which tends to stay constant across their works. We can compare the frequencies of different writings, to see which writings are similar.

This method was able to identify "Robert Galbraith" as the pen name of JK Rowling (note the overlap of the light blue and dark blue dots in the graph).

(To try different word combinations on this graph, see [Can You Identify an Author By How Often They Use the Word "The"?](#))



The Author's Hidden Fingerpint

Despite years passing, changing genres, and evolving their writing -- authors cannot change. They leave an identifiable fingerprint in their writing. Below plots JK Rowling's Harry Potter books against the books Rowling wrote under the penname Robert Galbraith. Click the words below to change the plot and hover over a data point for more information.

(All Numbers Below are rate per 10,000 Words)

JK Rowling
Robert Galbraith
Louise Penny
Michael Connelly
what
80
70
60
50
40
30
20
40
60
80
100
120
but
a all also an and any are as at be been but by can do down even every for from had has
have her his if in into is it its may more must my no not now of on one only or our shall
should so some such than that the their then there thing this to up upon was were what when
which who will with would your

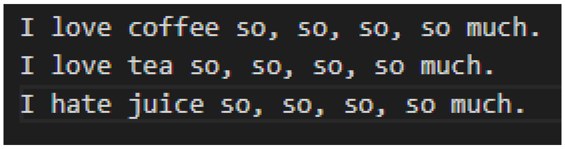
Created by @BenBlatt
For more information read Nabokov's Favorite Word Is Mauve

In this project...

In this project, you will develop a TextAnalyzer class. A TextAnalyzer object will read in a file and do all of the analysis needed to create the frequency “fingerprint” for that text.

Here’s an example of how the TextAnalyzer works, using a short text.

File tinyfile_3.txt



```
I love coffee so, so, so, so much.
I love tea so, so, so, so much.
I hate juice so, so, so, so much.
```

sentence_count

3

word_count

12

vocabulary

['coffee', 'hate', 'i', 'juice', 'love', 'much', 'tea']

frequencies

{'hail': 2, 'michigan': 1, 'champions': 1, 'west': 1}

frequency of ‘much’

3

percent_frequencies

{'i': 0.25, 'love': 0.1666666, 'coffee': 0.083333333, 'much': 0.25, 'tea': 0.0833333, 'hate': 0.0833333, 'juice': 0.0833333}

most common

['i', 3]

five_least_common

[('coffee', 1), ('tea', 1), ('hate', 1), ('juice', 1), ('love', 2)]

The code

You will create a class called TextAnalyzer with the following methods. Implement the methods so that all provided test cases pass:

```
class TextAnalyzer:
   def __init__(self, filepath):
        """Initializes the TextAnalyzer object, using the file at filepath.
        Initialize the following instance variables: filepath (string),
       words (list)"""

    def sentence_count(self):
        """Returns the number of sentences in the file (seperated by .)
        Note that if there are no '.' in the sentences return 1”""

    def words(self):
        “”” Returns a list of words without punctuation and all lowercase. For
         example : 'Cat!' Should be 'cat'. “””

    def remove_stopwords(self, words):
        “”” This takes in the list of words that are not punctuated and are lowercase.
         Returns a list of words with the stopwords provided by the file
        ‘stopwords.txt’ removed. “””

    def word_count(self):
        """Returns the number of words in the file not including the stopwords. A word
        is defined as any text that is separated by whitespace (spaces, newlines, or
        tabs)."""

    def vocabulary(self):
        """Returns a list of the unique words in the text, sorted by
        alphabetical order. Capitalization, punctuation, and stopwords should be
        ignored, so Cat!' is the same word as 'cat'. The returned words should be all
        lowercase, without punctuation or stopwords."""

    def frequencies(self):
        """Returns a dictionary of the words in the text and the count of how
        many times they appear. The words are the keys, and the counts are the
        values. All the words should be lowercase and without punctuation and should
        not include stopwords. The order of the keys doesn't matter."""

    def frequency_of(self, word):
```

```
    """Returns the number of times the word appears in the text. Capitalization,
    punctuation, and stopwords should be ignored, so 'Cat!' is the same word as
    'cat'. If the word does not exist in the text, then return 0"""

def percent_frequencies(self):
    """Returns a dictionary of the words in the text and the frequency of the
    words as a percentage of the text. The words are the keys, and the
    counts are the values. All the words should be lowercase, without
    punctuation or stopwords. The order of the keys doesn't matter."""

def most_common(self):
    """Returns the most common word in the text and its frequency in a list.
    There might be a case where multiple words have the same frequency,
    in that case return one of the most common word which should be lowercase,
    without punctuation or stopwords."""
    # Example output : ['officer', 6]

def five_least_common(self):
    “””Returns the five least common words in the text and its frequency as a list
   of tuples. If there are not five words in the text, return all the least common
   words. There might be a case where multiple words have the same frequency, in
   that case, return any of the least common words which should be lowercase,
   without punctuation or stopwords.”””
   # Example output: [(‘ants’, 1), (‘apple’, 1), (‘bat’, 1), (‘cat’,2)]

def read_sample_csv(self):
    """Reads the sample.csv file and returns the list of fieldnames"""
    # Output Format: filepath, total words, word count removing stopwords, line
    count, most common word

def write_analysis_details(self, csvfile):
    """Writes the details of the textual analysis to the csvfile.
    Refer to sample.csv for an example of how this should look.
    Note that for most_common, just write the word and not its frequency"""
    # Output Format: filepath, total words, word count removing stopwords, line
   count, most common word


def similarity_with(self, other_text_analyzer):
    """Extra credit. Calculates the similarity between this text and
    the other text using cosine similarity. Words should be lowercase, without
    punctuation or stopwords. """
```

Grading

There are unit tests included in the stub code that test each method. We will use the same tests that we provide to you in order to calculate your final grade.

Method	***points***
`def __init__(self, filepath)`	5
`def sentence_count(self):`	10
`def words(self):`	10
`def remove_stopwords(self, words):`	10
`def word_count(self):`	10
`def vocabulary(self):`	15
`def frequencies(self):`	30
`def frequency_of(self, letter):`	15
`def percent_frequencies(self, letter):`	25
`def most_common(self):`	25
`def five_least_common(self):`	25
`def read_sample_csv(self):`	5
`def write_analysis_details(self, csvfile):`	15
Total	**200**
`def similarity_with(self, other_text_analyzer, n=10):`	15 pts extra credit

If all of the unit tests for a method pass, you get all of the points for that method! If only some of the tests pass, you get a fraction of the points for that method. For example, if 2 out of 3 tests related to word_count() pass, then you get ⅔ of the possible points for line_count (10 points out of 15 points).

Tips

Work on one method at a time. Choose the one that you think is the easiest, and work on it until you can get all the tests related to that method to pass. This is a great strategy, since ***the solution to some methods can be used to quickly complete other methods.***

Make sure you are using Python 3!! Some of the tests won't pass if you are using Python 2.

Extra Credit: Calculating similarity - 15 points

Now let's see how one text compares to another text. Here are two different texts:

File tinyfile_1.txt	File tinyfile_3.txt
['coffee', 'is', 'so', 'good']	['i', 'love', 'coffee', 'so', 'so', 'so', 'so', 'much', 'i', 'love', 'tea', 'so', 'so', 'so', 'so', 'much', 'i', 'hate', 'juice', 'so', 'so', 'so', 'so', 'much']

One way to measure their similarity is to compare the frequencies of the different letters in these texts. We can use the frequency that's calculated by the TextAnalyzer, but let's make sure that the letters are the same in each.

Frequency for **tinyfile_1.txt**

word	**frequency**
‘coffee’’	1
‘good’’	1

Frequency for **tinyfile_3.txt**

word	**frequency**
‘i’	3
‘love’	2
‘coffee’	1
‘much’	3
‘tea’	1
‘hate’	1
‘juice’	1

Only one of the words are used in both texts, therefore we don’t expect these texts to be very similar! These frequencies create a sort of vector for each text. We can measure the similarity of two vectors using something called the *cosine similarity*. So, we can measure the similarity between two texts using the cosine similarity as well.

The cosine similarity of two vectors is:

*The Dot product of the two vectors / (Magnitude of the first vector * Magnitude of the second vector)*

Step 1 : Get the most common word’s frequency from both text

Step 2 : Create a list of the shared words

Step 3 : Calculate the dot product of the shared words (i.e) in this case it will be only ‘coffee’ which is only 1 but if there were 2 words shared words : ‘coffee’ : 1, ‘coffee’ : 2; ‘good’ : 1, ‘good’ : 1, then the dot product will be 1 * 2 + 1 * 1 = 3

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

More info : https://simple.wikipedia.org/wiki/Dot_product

(Tip: Note that only the words that both texts have in common actually matter in this calculation)

How to calculate the magnitude:

Step 4 : The most common word’s frequency is the magnitude for each of the text

Step 5 : Now dot product / text1 mag * text 2 mag

Here the answer would be 1 / (2 *1) = 0.5

Miscellaneous

Useful string methods: split() and strip()

```
>>> s = “I love cats. I love every kind of cat!\n”
>>> s. split()
['I', 'love', 'cats.', 'I', 'love', 'every', 'kind', 'of', 'cat!']
>>> s = 'cats.'
>>> s.strip(“.!”)
'cats'
>>> s = 'cat!'
>>> s.strip(“.!”)
'cat'
```

Useful function: sorted()

```
>>> l = ['love', 'every', 'kind', 'of', 'cat']
>>> sorted(l)
['cat', 'every', 'kind', 'love', 'of']
>>> sorted(l, reverse = True)
['of', 'love', 'kind', 'every', 'cat']
>>> sorted(l, key = lambda x : x[-1]) # sort by the last letter
['kind', 'love', 'of', 'cat', 'every']
```

According to the Python documentation: *“It is best to think of a dictionary as an unordered set of key: value pairs”*.