

# Project 1: TextAnalyzer: 200 Points

Due Feb 20

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 an infrastructure for analyzing texts so they can be compared for similarity.

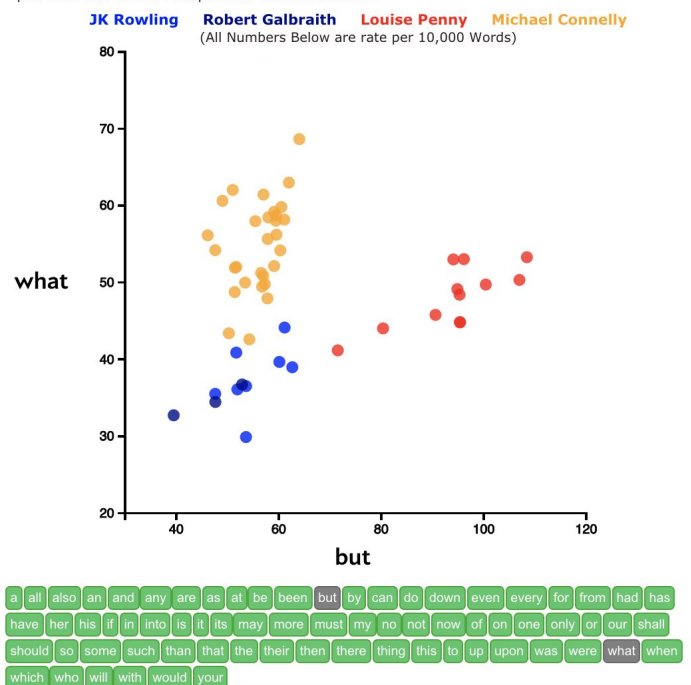
Data scientists have successfully achieved this by comparing the frequency of common words in an author's writings. These frequencies form a descriptor of an author's style, which tends to stay constant across their works. We can compare the frequencies of different writings, to see which writings are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

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.



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 very short text.

File `fightsong.txt`

Hail! Hail! to Michigan,  
the champions of the West!

```
>>> fightsong = TextAnalyzer("files_for_testing/fightsong.txt")
>>> fightsong.line_count()
2
>>> fightsong.word_count()
9
>>> fightsong.vocabulary()
['champions', 'hail', 'michigan', 'of', 'the', 'to', 'west']
>>> fightsong.frequencies()
{'hail': 2, 'to': 1, 'michigan': 1, 'the': 2, 'champions': 1, 'of': 1, 'west': 1}
>>> fightsong.frequency_of('champions')
1
>>> fightsong.frequency_of('ohio')
0
>>> fightsong.most_common()
'hail'
>>> fightsong.most_common(3)
['hail', 'the', 'champions']
>>> fightsong.percent_frequencies()
{'hail': 0.2222222222222222, 'to': 0.1111111111111111, 'michigan': 0.1111111111111111, 'the': 0.2222222222222222, 'champions': 0.1111111111111111, 'of': 0.1111111111111111, 'west': 0.1111111111111111}
```

## 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), lines (list)"""
    def line_count(self):
        """Returns the number of lines in the file"""
    def word_count(self):
        """Returns the number of words in the file. A word is defined as any
        text that is separated by whitespace (spaces, newlines, or tabs) or
        followed by punctuation (like ? or !)."""
    def vocabulary(self):
        """Returns a list of the unique words in the text, sorted in
        alphabetical order. Capitalization should be ignored, so 'Cat' is the
        same word as 'cat'. The returned words should be all lower-case."""
    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. The order of the keys
        doesn't matter."""
    def frequency_of(self, word):
        """Returns the number of times each word appears in the text. Capitalization
        should be ignored, so 'Cat' is the same word as 'cat'."""
    def percent_frequencies(self, word):
        """Returns a dictionary of the words in the text and frequency of the words as
        a percentage of the text The words are the keys, and the percentages are the
        Values. All the words should be lower case. The order of the keys doesn't
        matter."""
    def most_common(self, n=1):
        """Returns a list of the most common n words in the text. By default,
        n is 1. The returned words should be sorted by frequency.

        There might be a case where multiple words have the same frequency,
        but you can only return some of them due to the n value. In that case,
```

```

        return the ones that come first alphabetically."""
def read_sample_csv(self):
    """Reads the sample.csv file and returns the list of fieldnames"""
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."""
def similarity_with(self, other_text_analyzer, n=10):
    """Extra credit. Calculates the similarity between this text and
    the other text using cosine similarity. See the Extra Credit section
    of the project specification for details."""

```

## 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.

| <i><b>Method</b></i>   | <i><b>points</b></i> |
|--|----------------------|
| <code>def __init__(self, filepath)</code>                          | 5                    |
| <code>def line_count(self):</code>                                 | 10                   |
| <code>def word_count(self):</code>                                 | 15                   |
| <code>def vocabulary(self):</code>                                 | 20                   |
| <code>def frequencies(self):</code>                                | 40                   |
| <code>def frequency_of(self, word):</code>                         | 15                   |
| <code>def percent_frequencies(self):</code>                        | 25                   |
| <code>def most_common(self, n=1):</code>                           | 45                   |
| <code>def read_sample_csv(self):</code>                            | 10                   |
| <code>def write_analysis_details(self, csvfile):</code>            | 15                   |
| <i><b>Total</b></i>  | <b>200</b>           |
| <code>def similarity_with(self, other_text_analyzer, n=10):</code> | 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 `line_count()` pass, then you get  $\frac{2}{3}$  of the possible points for `line_count` (8 points out of 12 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 osusong.txt

Come on, Ohio!  
Smash through to victory.

File fightsong.txt

Hail! Hail! to Michigan,  
the champions of the West!

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

### Frequency for **osusong.txt**

| word        | Frequency |
|-------------|-----------|
| 'champions' | 0         |
| 'come'      | 1         |
| 'hail'      | 0         |
| 'michigan'  | 0         |
| 'of'        | 0         |
| 'ohio'      | 1         |
| 'on'        | 1         |
| 'smash'     | 1         |
| 'the'       | 0         |
| 'to'        | 1         |
| 'through'   | 1         |
| 'victory'   | 1         |
| 'west'      | 0         |

### Frequency for **fightsong.txt**

| word        | Frequency |
|-------------|-----------|
| 'champions' | 1         |
| 'come'      | 0         |
| 'hail'      | 2         |
| 'michigan'  | 1         |
| 'of'        | 1         |
| 'ohio'      | 0         |
| 'on'        | 0         |
| 'smash'     | 0         |
| 'the'       | 2         |
| 'to'        | 1         |
| 'through'   | 0         |
| 'victory'   | 0         |
| 'west'      | 1         |

Only the word 'to' is included in both, so 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)*

### How to calculate the dot product:

For each word, multiply the frequency from text 1 with the frequency from text 2. Add up all those values.

So, for the texts in this example, it would be:

$$\begin{matrix} 0 * 1 & + & 1 * 0 & + & 0 * 2 & + & 0 * 1 & + \dots + 0 * 1 = 1 \\ ('champions') & & ('come') & & ('hail') & & ('michigan') & & ('west') \end{matrix}$$

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

### How to calculate the magnitude:

For one text, add up all of the frequencies squared. Then, take the square root of that.

So, for **fightsong.txt**, magnitude would be:

$$\text{sqrt}(1^2 + 0^2 + 2^2 + 1^2 + \dots + 1^2) = 13$$

('champions') ('come') ('hail') ('michigan') ('west')

(Tip: Note that only the words that occur in that particular text actually matter in this calculation)

Cosine similarity returns a number between 0 and 1, where 1 means the texts are identical, and 0 means that the texts are entirely different.

You will implement this calculation in the `similarity_with()` method.

```
>>> fightsong.similarity_with(osusong, 7)
0.10482848367219184
```

This method also takes parameter `n` that determines the number of words from each text to include in this calculation. The `n` most common words from each text should be used in the cosine similarity calculation. In the example above, the 7 most common words from each text were used. There are only 7 unique words in each text, so we used all the words available.

## 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”*. As of Python 3.7 and above, the keys do preserve an order, which is the insertion order. But, we will not evaluate you on the order of your dictionaries.