CS3023
Intermediate Programming
Program Assignment 3

Preparation

Review the materials up to, and including, OOP and unit testing. You also need to know the Python string processing and file I/O. Review the appropriate reference materials (sample programs, Python documentation, textbooks, and online references).

Acknowledgment

This assignment is one of the nifty assignments submitted to SIGCSE 2013 (ACM Special Interest Group on Computer Science Education) by a professor from University of Toronto. We customized it for this course.

Problem Statement

We want to venture into data science business. For this assignment, you will implement a program that will predict the authorship of an unknown text (literary work). Techniques you learn for authorship detection have applications in plagiarism detection, email-filtering, and forensic research. The main objective of this assignment is to review OOP and unit testing.

Author Detection

The first step in detecting (guessing) the author of an unknown text is to collect linguistic "signature" of known text. A signature of a text is a collection of statistical "features." An example of a feature is the number of words per sentence. Another feature may be the frequency of a certain phrase in a text. For this assignment, you will collect five features to formulate a signature. We express the signature as a numerical value.

Suppose we have already collected signatures of work done by 10 known authors. Given a mystery text, how can we guess its author? One approach is to compare the signature of the mystery text against the known signatures. We guess that the one with the least difference is the author (has the highest probability of being the author). How well does this approach work depends on the quality of the features we use. In addition to selecting a good feature set, we also weight the features, i.e. features that are considered more important or critical in making distinction are given higher values, to increase the accuracy of prediction.

Features

In this assignment, we will collect five features:

- Average Word Length
- Type-Token Ratio

- Hapax Legomana Ratio
- Average Sentence Length
- Sentence Complexity

Average Word Length

This calculates the average number of characters per word in the document.

Type-Token Ratio

This calculates the number of different words used in a text divided by the total number of words. The idea here is to measure the repetitiveness of words in the text.

Hapax Legomana Ratio

This feature, like the type-token, is a ratio using the total number of words as the denominator. The numerator for this feature, however, is the number of words occurring exactly once in the text.

Average Sentence Length

This calculates the average number of words per sentence.

Sentence Complexity

This feature calculates the average number of phrases per sentence.

Sentences, Phrases, and Words

To implement the feature extraction routines, we need to define precisely what is a sentence, a phrase, and a word. We simplified the definition of these terms to keep this program assignment manageable.

We will call an element of the list returned by the string method split a *token*. For example, when you execute

```
s = "Hi, my name is Jan. And, this is Jim."
s.split()
```

it will return this list with 9 tokens:

```
['Hi,', 'my', 'name', 'is', 'Jan.', 'And,', 'this', 'is', 'Jim.']
```

A *word* is a non-empty token that has at least one character that is not punctuation. A *sentence* is a sequence of words terminated by exclamation point, question mark, a period, or the end of a file. A *phrase* is a non-empty portion of a sentence that is separated by a colon, a comma, or a semi-colon. Notice, the sentences do not include their terminator character. The last sentence is not terminated by a character; it finishes with the end of the file. We have this definition to handle the case when the last character in the text is not a character that terminates a sentence. Sentences can span multiple lines of the file.

Starter Program and Support Files

The starter program named PA3_starter_signature.py is provided to you. Your task is to complete the class implementation. The FileHelper class for handling file input/output and string processing services are provided in the PA3_helper.py file. You use the helper class as-is; you do not make any changes to this file.

A quick check unit tester code is also provided so you can perform rudimentary tests on your feature extraction routines. Before you do a full test of your code, you should run this quick check program to iron out simple errors. The file is PA3_quick_tester.py.

Signatures for 13 authors are provided. Put these files, and nothing else, in one directory. Also provided are 5 mystery text files. If you look inside the file, you will actually find the name of the author. This is helpful to manually check if your code is detecting correctly or not. You may place the mystery text files anywhere, but we suggest you to place them in a separate directory.

Specification

Please review the starter code for more details on these functions. Implement the following seven functions. Notice that these functions are presented here in a non-OOP format. When these are placed in an OOP format, making them methods, you will have self as the first parameter, as declared in the source file.

1. split_on_separators(original, separators)

   Return a list of non-empty strings in original, separated by single characters in separators. For example:

   ```
   split_on_separators("Hooray! Finally, we're done.", "!,")
   ```

   will return the list

   ```
   ['Hooray', ' Finally', " we're done."]
   ```

   Here we are separating the given string by either the exclamation mark (!) or the comma (,).

2. average_word_length(text)

   Return the average length (float) of words in the parameter text (list of strings).

3. type_token_ratio(text)

   Return the type token ratio (float) of the parameter text (list of strings).

4. hapax_legomana_ratio(text)

   Return the hapax legomana ratio (float) of the parameter text (list of strings).

5. average_sentence_length(text)

   Return the average number (float) of words per sentence in the parameter text (list of strings).

6. average_sentence_complexity(text)

   Return the average number (float) of phrases per sentence in the parameter text (list of strings).

7. compare_signatures(sig_a, sig_b, weight)

   Return a non-negative real number that shows the similarity of two linguistic signatures. The smaller the number the more similar the signatures. The value of zero means the two

signatures are identical. The last parameter weight is a list of floats for the weight factors assigned to the signatures. The formula to use is

$$S_{ab} = \sum_{i=1}^{5} \|f_{i,a} - f_{i,b}\| * w_i$$

The similarity of signatures sig_a and sig_b, S_{ab} , is the sum of absolute difference of each feature $f_{i,a}$ and $f_{i,b}$, multiplied by the corresponding weight w_i.

Before you apply any of the feature extraction functions, remember to preprocess the data as appropriate by calling the provided clean_up function that converts all letters to lowercase and remove all punctuation marks from the beginning and the end (inner hyphen such as built-in remains in the text). Also, notice that we are passing a list of strings as the parameter to the feature extraction functions. Each element in the list corresponds to one line in the file. When processing sentences in the text, remember that they can span across multiple elements in the list.

Requirements

1. You MUST use the support classes as given. DO NOT MODIFY what's given, unless instructed to do so. For instance, you are NOT allowed to change the name of the functions (methods) or the number of parameters of the functions. Review the markers TO DO to locate the places where you need to make changes. We encourage you to add your own (helper) functions/methods to the program as appropriate. Please read the additional information given in the source files.
2. Do not include any input or print statements in the seven functions you are implementing for this assignment. Specifically there should be no input and print statements inside split_on_separators, average_word_length, type_token_ratio, hapax_legomana_ratio, average_sentence_length, average_sentence_complexity, and compare_signatures.
3. Breakdown the tasks into well defined functions/methods of manageable size. Do not write a code unit that is more than one page long. Avoid duplicating the code across multiple functions. If you see the same functionality repeated, define a function to implement this functionality.
4. Format the program in a clean and easy-to-read manner. Insert one or more blank lines between statements as necessary to organize statements in logical groups. Do not pack statements too densely (e.g., 20 statements with no blank lines between them).
5. Be sure to include a header comment in the source file. Include your name, the course number, the assignment number, and the program description. Include any additional information if you wish.
6. Use comments judiciously. Do not include redundant and meaningless comments.

Incremental Test-Driven Development

You are expected to follow the incremental, test-driven development steps to construct this program. The functions you implement for this assignment are fairly independent of each other, so you have a wide freedom to choose the order of implementation.

After the feature extraction operations are implemented, write your main program. The main program should implement the following logic:

1. Get the name of a file that contains a mystery text.
2. Compute and collect the signature of this text using five feature extraction routines.
3. Get the known signatures (13 signatures are provided to you)
4. Compare the known signatures again the mystery signature and guess the author of the mystery text. For comparing the signatures use the following weight distribution

   weights = [0, 11, 33, 50, 0.4, 4]

Submission

You will submit three source files:

1. PA3_<last name>_signature.py

   This is an object-oriented implementation of feature extraction routines.

2. PA3_<last name>_test.py

   This file contains your unit tests for checking the feature extraction routines in the Signature class. You don't need to test the FileHelper class.

3. PA3_<last name>_main.py

   This is the main program that uses the Signature and FileHelper classes.

Go to the Programming Assignment 3 page in the Sakai course website and submit both versions of your program.