Computer Laboratory 3

CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development

1 Introduction

In this lab we are going to write some of the core parts of a simple algorithm for "guessing" the author of an unknown writing sample. The core idea of this algorithm is that we can count the words in a writing sample and compare them with the words of another writing sample. As authors tend to have a relatively unique style of writing, they also tend to prefer certain words over others. While word choice is only one of many possible expressions of writing style, with a large enough corpus of writing samples, this strategy can be rather effective.

As this is only a lab-sized assignment, I've simplified a few parts of the broader process of word identification. Nonetheless, this is an example of the type of coding that might go in to a machine-learning based author identification task. We will be splitting this process up into a few core steps:

1. Input processing (we will need to go from strings containing full writing samples to a list of words)

2. Data summarization (going from lists of words to a dictionary counting each distinct word)

3. Similarity Scoring (a function that takes two dictionaries and computes a score for how similar the two are)

4. Ranking for best-guess (a function that takes many writing samples with known author, and one sample with unknown author, and returns the author whose writing sample matches the unknown the best, based of similarity score)

This basic process, with enough data, can be quite effective at many tasks. For example you can make a basic language guessing algorithm using much of the same logic, but counting letter choice, rather than word choice. (As letter choice can be quite unique to a given language)

2 Software environment setup

Like the previous lab, you will be doing python programming. While you are allowed to prepare your python programs in any reasonable programming environment we continue to recommend PyCharm. The basic PyCharm setup instructions can be found in the previous lab. Some notes here:

1. You will want to make a new PyCharm Project for this new lab. Mixing labs in one project can lead to accidentally submitting the wrong file or other "carry-over" from lab to lab that might interfere with grading.

2. If PyCharm asks you to set a project interpreter, and no default shows up, you should be able to add one. On the CSELabs machines "/usr/bin/python3" is the name of the default python3 interpreter, which is a fine choice.

3. Remember to note where PyCharm makes the python project itself, you may even want to make sure the project is placed somewhere designated for this class when you set it up. Eventually you will need to find those files outside of PyCharm.

3 Lab Reminders

- You are allowed to change lab partners each week if you want, you are never locked into one partner, ask the TAs if you want help finding a partner

- Each programmer in a partnership should be equal – try not to run ahead of your partner, make sure they understand what you are doing, and make sure that each programmer spends equal time on the keyboard. (Remember, true pair programming is two programmers, one keyboard, one screen).

- If you do not finish, you and your partner should continue working together outside of lab to finish the lab. I hold both students in a partnership academically responsible for your solution (I expect either student to be able to explain their code to me)

4 Files

This lab will involve the following files

- `word_count.py` You should be writing this module. This is the name the tests will expect.

- `word_count_test.py` This file is provided. It has tests for all required functions. There is a boolean flag at the top that enables a more detailed test for the count function, I've set this to False to begin with as the test output gets *really* long with the full output. However, if you are having trouble with the count tests, this will print your counts with more detail, which might help you debug your function.

5 Requirements and software design

As mentioned in the introduction, this is a somewhat simplified form of this program. While we will have all the essential parts of a working piece of software, I've reduced those parts to their list, string, and dictionary cores. The following functions are required. Full descriptions will be given later.

- `my_split(text)` This function takes a text sample and returns a list of the words in that text sample (splitting the one string into a list of strings)
- `count(words)` This function takes a list of words and returns a dictionary whose keys are the words in the list, and whose values are the count of times each word appears.
- `word_count_similarity(count1, count2)` This function takes two dictionaries, each a count of words, and computes the "cosine" similarity equation over the two dictionaries. This returns a number from 0 to 1, with higher numbers indicating a more similar word distribution.
- `best_guess(known_author_counts, unknown_count)` This function takes two dictionaries, the first contains word counts with known authors – the keys are author names, and the values are word count dictionaries for those authors. The second is a single word count dictionary with unknown author. This function will return the most likely author's name, based on the word count similarity with the known author's writing samples.

5.1 my_split

Note/Update Experience from the early labs has taught us that this is the hard part of this lab. Don't get discouraged if you spend most of your time on this step. Also note that the other functions can be written and tested without this function (you might need to comment out some tests in the test file, but you shouldn't need to run this function)

The `my_split` function has one parameter – `text` a large string containing a long text sample. In real text, this string may contain many numbers and all the full complexities of English text, punctuation, and formatting. For this lab, however, you can assume that the `text` string contains only letters (uppercase and lowercase, English alphabet) punctuation (,.?!) and spaces.

The `my_split` function should return a list of strings containing the words from the text sample, in the order they appear. You can assume that words are separated with spaces. Words may, or may not, contain punctuation (,.?!) after the last letter, which should be removed.

So given the input `"a dog, a goat, and an apple!"`, `my_split` should return `["a", "dog", "a", "goat", "and", "an", "apple"]`

Note: The python `split` function does much of this behavior and is therefore **NOT ALLOWED** in this lab. I want you to practice splitting this string yourself.

Hints / Advice / Try these approaches if you find yourself getting stuck here:

- Separate the parts of your code that deals with punctuation for the part that splits the words into a list. Only focus on one problem at a time, trying to solve both will give you a headache.
- You only need to worry about punctuation at the end of a word, and you only have to worry about the four punctuation marks: `"?!,."`. Keep this in mind, as it makes removing punctuation a lot easier.
- The splitting problem is easier if you "remove" words from the text string as you add them to a list of results, trying to do this without updating the text string is a fair deal harder. (I.E. go from "a happy dog" to "a" (in one string) and "happy dog" in another string, then next loop can start with "happy dog" and break it down further)
- Start working on splitting the string in a different file, work out an approach that works for isolating *one word* from a longer string, only worry about repeating the process once you have the basic step solved.
- Many of the list functions we learned apply to strings as well, notably you can use `somestring.index("")` To find the index of the first space, and we can use the `in` operator to check if a string contains a given letter.
- The Slicing operator can be used with strings to get a substring containing only one word (so long as you know where the word starts and stops)

5.2 count(words)

The `count` function has one parameter – A list of strings (words). This list of words represents a text sample, and would be the return value of one or more call to `my_split`.

This function should return a dictionary. The return dictionary should have strings as keys and integers as values. The keys should be the unique words in the input list, and the values for those keys would be the number of times those words are in the list.

As an example, if given the input `["a", "dog", "a", "goat", "and", "an", "apple"]`, `count` should return `{'a':2, 'dog':1, 'goat':1, 'and':1, 'an':1, 'apple':1}`

It's quite possible that there is a built-in python function to do this. Python has a fantastic library of built-in functions. You are, of course, not allowed to use that, I expect you to manually loop over the words list counting the words.

A note on tests: The test file has two types of tests for this. The first test simply compares the output of your function with the expected output printing True (equal) of False (unequal) While this is sufficient for most purposes, it is not helpful for debugging, as it doesn't help you discover what is counted wrong! There is a boolean variable at the top of the test file that can be set to True to print all words, in order, from your dictionary, which can then be compared with output for the reference solutions. This can help you debug, but is quite long.

5.3 word_count_similarity(count1, count2)

The word count similarity function takes two dictionaries, both word count dictionaries as would be returned from the count function. These dictionaries may not contain the same words, I.E. count1 may have words in it that are not in count2, and count2 may have words in it that are not in count1. If a word is not in a dictionary, it should be assumed to have a count of 0.

This function should return a double value from 0 to 1 representing the cosine similarity metric computed between these two counts. Cosine similarity metric is only tenuously connected to the trigonometric function $\cos(\theta)$, instead this comes from machine learning metrics, and the study of high-dimension vectors.

Mathematically, the equation to compute is:

$$\frac{\sum_{word \in count1 \cap count2} count1[word] count2[word]}{(\sum_{word \in count1} count1[word]^2)(\sum_{word \in count2} count2[word]^2)}$$

However, since that equation is hard to read without prior exposure to the mathematical syntax I will also summarize it in pseudocode:

1. Compute S1 as the sum of the square of each word count in count1
2. Compute S2 as the sum of the square of each word count in count2
3. Compute S3 as the sum over each word that is in both count1 and count2 of (count1[word] * count2[word])
4. return $\frac{S3}{S1 * S2}$

For example given count1 = `{'a':2, 'b':3, 'c':1}` and count2 = `{'a':1, 'b':3, d:4}` we would have

- S1 $= 2^2 + 3^2 + 1^2 = 4 + 9 + 1 = 14$
- S2 $= 1^2 + 3^2 + 4^2 = 1 + 9 + 16 = 26$
- S3 $= 2 * 1 + 3 * 3 = 2 + 9 = 11$
- result $= \frac{11}{14 * 26} = 0.3021978021$

5.4 best_guess

The best_guess function takes two parameters,

- `known_author_counts` This is a dictionary containing word counts by known authors. The keys of this dictionary would be strings – denoting an author's name, and the values would be word count dictionaries, as would be returned by `count`.

- `unknown_count` This is a single word count dictionary as would be returned by `count`, representing the word counts of a sample of text written by an unknown author.

This function should return a string – one of the known authors names. The known author name that should be returned is the one whose word count dictionary is most similar to the unknown sample (as measured by the word count similarity function). In this way we guess the author whose word choice is most similar to the word choices made in the unknown sample.

6 Deliverable (README THIS HAS CHANGED)

Before submitting your work, please use the tests to carefully check that your code is bug-free, and has every function named as expected, and returning data types compatible with our expectations.

Name your submission file `word_count.py` and make sure that a comment at the top has your name, and if you worked with a partner, your partner's name. **If you worked without a partner and their name is not on your file they will not get credit, OR, you will be accused of cheating** (depending on if they turn in the file or not)

You should only submit the `word_count.py` file. You will submit this through canvas. If you worked with a lab partner, **BOTH STUDENTS SHOULD SUBMIT A COPY.** This is different than previous weeks, we believe it will be easier to grade submissions quickly if each student turns in their own copy.
EVERY STUDENT IS EXPECTED TO SUBMIT A COPY OF THE LAB! Even if they worked with a partner. It is OK if you and your partner submit the same file. We just want everyone to have a file submitted.

This will be due before the beginning of your next lab next week. The exact time will be based on the official start time of your lab. Canvas will accept re-submissions, and will also accept late submissions. Please be sure you do not turn in the assignment late on accident.

If you finish this lab during the lab time, feel free to notify your lab TAs, and then leave early. If you do not finish this lab during the lab time you are responsible for finishing before the next lab.