"Linear Algebra HW-1 Part-2"

Prepared for

Dr. Mohammad Taheri

TA

Ali Shahsavandi

Prepared by

Ali Maher

10th of May 2024

Introduction

This part of homework is coding assignment that we have to implement a program to check the similarity of two text files. Of course, we should use the tools and operators we have learned in the course. I implement the program in two approaches with python. One of them is very strait forward and simple to understand the other one is quite optimized and a bit harder to find out what's going on.

Let's Get Through

- ### Base Step

First of all, we have to provide some text files. I prepared a text file about 1200 word about "Assembly X86" and the other text file was about the "programming in new generation" with nearly 1700 words. Then, after opening the files in the code, I went for cleaning the data. What I mean by "cleaning data"? I mean removing punctuations, split the words and convert all the words to lowercase, then store them in a list. I wrote a function for this task called, **"cleanData()"**. And then, we faced with a lots of repeated words that we have to extract a unique list of them for more accurate result. So, I implement the **"extractUniqueWords()"** function.

```
def extractUniqueWords(words):
    unique_words = []
    for word in words:
        if word not in unique_words:
            unique_words.append(word)

    return unique_words

def cleanData(file_text):
    file_text = file_text.lower()
    file_text = file_text.translate(str.maketrans('', '', string.punctuation))
    cleaned_data = file_text.split()
    return cleaned_data
```

- **Approach-1**

Let's go for the first and easy approach for solving the problem. In this approach I consider three dictionaries, one for the first text file, one for the second text file and the last one for combining them in one. In these dictionaries I stored the words and related frequencies.

Now we can easily go for the inner product of the these two dictionaries. I get the values and consider them as vectors. As we know result of a inner product is a scalar. So, I did the product in a loop and get the result. I divide the result with a length of the combined dictionary to get a number in scale of 100.

```
Result of Approach-1:
Similarity of our main dictionary to itself:  100.0 / 100
Similarity of text_1 to main dictionary:  42.97 / 100
Similarity of text_2 to main dictionary:  57.03 / 100
Similarity of text_1 to text_2 based on main dictionary:  20.97 / 100
```

As we see in the result the files, I prepared has about 21 percent similarity.

But what are the setbacks of this approach? We get a lot of space for zero and not valuable things. Because our space here is a sparce space. I test the two files and found the following result:

```
print(len(dictionary_value))

print(dictionary_1_value.count(0))
print(dictionary_2_value.count(0))
```

✓ 0.0s

```
950
544
265
```

Yes, this shows that the space we are reserving is much more than we rally need, So, it is not optimized.

- **Approach-2 (Optimized)**

In this approach I used two main idea for implementation:

1. Instead of capturing a sparce matrix, we can just store the item name (here by item I mean words), and its frequency. So good for saving the space.
2. Merge Sort and two-pointer: For the merging the words in the first text file with the second one, I consider two pointers one for iterating on the first list and second one for iterating on the unique words list of second text file. Based on what we have in the Merge-Sort we have to sort the initial arrays. So I wrote a function named **"sortDictionary"**, it sort the dictionaries on their keys, that can be string or integer. So, by these types we need some further considerations in code.

```python
def sortDictionary(dictionary):
    sorted_dict = OrderedDict(sorted(dictionary.items(), key=lambda x: x[1], reverse=True))
    return sorted_dict
```

The iteration continues until one of the pointers reaches the end or both. We should pay attention to the values that the pointers are pointing, they may bring us different conditions (3 conditions, mentioned in code). After the main loop, it may happen that one of the arrays still has some element that doesn't mentioned by the pointers. So, we need to handle this case we two loops.

Now we are finished, and we can calculate the similarity with the product. The production is as the last approach. And we get the following result:

```
Result of Approach-2:
Similarity of text_1 to text_2 based on combined dictionary:  20.97 / 100
```

- **Conclusion:**

We find two implementing approaches for the similarity problem of two text files with python. The second one is much better and optimized than the first one. And we get 20.97 percent as result of my files, which means that these two files are not so similar :)

The project files with doc are also available on my [GitHub](#).

Thanks for your time