Report 1: Finding Similar Items: Textually Similar Documents

Laila Niazy, Svenja Raether

November 12, 2019

1 Introduction

The objective of assignment one is to find textually similar documents in a set of 5-10 documents. The implementation is done in python and can be divided into five mainparts: *Shingling, CompareSets, MinHash, CompareSignatures, and locality-sensitive hashing (LSH).* Ten articles of which two pairs appear to be similar are used for testing the implemented solution.

2 Description of the solution

In this section, we will described our implementation, which is divided into five main parts: Shingling, CompareSets, MinHash, CompareSignatures, and locality-sensitive hashing (LSH).

However, before we go through each function, we will give an overview of our git repository structure.

We have two folder in the first level. The folder "Dataset" contains the documents and the folder "Utils" contains all our functions, which we import in *main.py* to run. In the second level inside the "Utils" folder, we stored the main used functions: *minHash, LSH and CompareSignatures* and two folders named: "Version1" and "Version2". Those folder contain two different implementations of the shingling method. The difference between both will be described later on in Sec.2.2

2.1 Main function

Starting with our description of the file *main.py*, the file runs through the entire procedure using the other files in the project. The steps taken to accomplish the goal of the assignment are summarized below:

1. Check if the data file in the given path exists
2. Read all 10 documents from the data file
3. Depending on the version used: Shingle each document with a chosen shingle size and store the shingles for each document in a dictionary
4. Compare all possible documents and calculate their jaccard similarity. Return the pairs that exceed a chosen threshold t.
5. Calculate the signature matrix based on the minhashing algorithm
6. Compare all possible documents and calculate signature similarity. Return the pairs that exceed a chosen threshold t. (Not recommended for big data implementations since there can be too many possible pairs to be compared)
7. Bonus: perform LSH to preselect the documents by generating a candidate list. Return the LSHlist

2.2 Shingling

Shingling of a document means breaking the words in a document into parts of a chosen shingleSize k. The implementation of this method can be found in the files *Shingling1.py* and *Shingling2.py*. We did two shingle functions and both give as an output the shinglings of each documents as a dictionary. In version1, the keys of the dictionary are the IDs of the documents, which are contained in the dataset as identification of the documents. In version2, the keys of the shingling dictionary are the indices of the documents e.g. 1, 2 etc.

In *Shingling1.py*, the spaces between words are removed and we shingle one long word into a k-shingles. In *Shingling2.py*, the spaces are part of the shingles. The use of space is the major difference between both function because their use results in more shingles to describe a document and this effects the time and power used in the other functions later on to compute jaccard etc. Further, both shingle functions use different hash functions for their shingles. In both function, the shingles are created by iterating over the document and hashing the results. After each shingle is hashed, it is stored in a set, which gets update with more shingle in each iteration. The use of the data structure set means that duplicates will not appear more then once

in the returned set of shingles. Finally, as an output as mentioned above we get a dictionary.

2.3 CompareSets

In version1 inside the file *Shingling1.py*, we have the function "jaccard", which is used to calculate the jaccard similarity of two documents. It takes as input the dictionary with all shingles of all documents and iterate over every unique pair of documents and compute for each two the jaccard similarity. In the end it returns a list, which contains tuples with IDs of the most similar pairs and their jaccard similarity. The most similar pairs has a jaccard value, which exceed a specific threshold, we choose. The jaccard similarity is calculated using the following equation:

$$\mathbf{jacc}(A, B) = \frac{|\ A \bigcap B|}{|A \bigcup B|} \tag{1}$$

Moreover, in version2 we have another file called *CompareSets.py*, which has a function "jaccardSimilarity" and it does the same calculations as "jaccard" but takes only two sets of shingles as an input and returns the similarity of all pairs.

2.4 MinHashing

MinHashing.py is used to calculate the signature matrix. Therefore, it has a function, which takes as input the dictionary of shingles calculated from the shingle step and the number of hash functions needed for constructing the matrix. First, we create a set of all existing shingles, excluding duplicates among the documents. Then, we determine the length of all existing shingles and the number of documents to construct the starting matrix with the number of hash functions and the number of documents as axes and number of shingles as the biggest possible dummy value for each cell. The hash function $(a * i + b)\%c$ is used with a and b being random numbers and c being a large constant prime factor. The minHash algorithm tests whether each shingle is in a given document and replaces the current value of the matrix, but only if the value in the hash function is smaller. This process is repeated until the signature matrix contains the smallest values.

2.5 CompareSignatures

The function in *CompareSignature.py* is used to compare the signatures of the documents given by the signature matrix. We compare each row of one column in the signature matrix with the another column in the signature

matrix, since each column represents a document. We sum up the number of rows, which are equal and divide this number by the the total number of rows to get the similarity value.

2.6 LSH

The LSH technique is used to find candidate pair, that might be similar and its implementation can be found in *LSH.py*. In the case of a large the data set, it is not feasible nor efficient to compare all possible combinations. Therefore, the signature matrix is divided into b bands with r rows each. Those values are chosen based on the size of the signature matrix. For each document, the bands are hashed into k buckets. If two hashed bands end up in the same bucket for at least one of the hash functions (our example only uses one hash function), those bands will be considered candidate pairs. In the next step, the signatures of the candidate pairs get compared using the signature similarity. If they exceed the chosen threshold, they are considered to be similar. Optionally, jaccard similarity can be applied to those pairs in order to filter out any false positives. The step with the jaccard similarity is done in the *main.py* as the finale step, after we get the output with the candidate pairs.

3 Evaluation

3.1 Data set

The data set contains ten articles, where two pairs suffer from plagiarism. The initial data set belongs to an online tutorial for MinHashing. (https://mccormickml.com/2015/06/12/minhash-tutorial-with-python-code/). We extracted ten articles out of this data set and modified some of the entries for testing.

3.2 Tests

In the first test we changed the number of hashfunctions, while $k = 10$ and the threshold being 0.6. The results of version1 can be found in Tab.3 and of version2 in Tab.4. In each cell, we wrote down the value of the jaccard for both similar pairs, that we know exist, if there is no value in the table that means the similar pairs were not detected. The more hashfunction one uses the more accurate the statements are, which means they get closer to the values gotten using Shingling and jaccard similarity, which can be observed in the results.

In the second test we changed the value of the shingle size, while the threshold and the number of hash functions were set to 0.6 and 8 respectably. The results found in Tab.1 and Tab.2, which show how many pairs were found when using different k. When using $k = 2$ alot of documents were predicted as pairs using minhash. This is expected because two character after eachother can normally appear alot in most documents. The larger k is, the more accurate until we get to a point, where we accuracy drop and this was observe we used $k = 20$. The difference between 10 and 20 in similarity score can be seen in Tab.6 for the algorithm shingling.

Finally, for the following values: $k = 10$, $t = 0.6$ and the number of hash-functions 8, we measured the time taken to execute each function. The results can be found in Tab.5.

Note: In all the tables LSH with FP stands for using LSH algorithm then filtering out the false positives in the candidates.

Jaccard Similarity	$k = 2$	$k = 4$	$k = 6$	$k = 10$	$k = 20$
Shingling	2	2	2	2	2
Minhash	42	2	2	2	2
LSH with FP	2	2	2	2	2

Table 1: Results of found pairs with varying k and using version 1

Jaccard Similarity	$k = 2$	$k = 4$	$k = 6$	$k = 10$	$k = 20$
Shingling	2	3	2	2	2
Minhash	20	4	2	2	2
LSH with FP	2	2	2	2	2

Table 2: Results of found pairs with varying k and using version 2

Jaccard Similarity	$h = 2$	$h = 4$	$h = 6$	$h = 8$	$h = 8$
Shingling	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)
Minhash	(0.75,0.63)	(1.0,1.0)	(1.0,1.0)	(0.88,0.63)	(0.83,0.67)
LSH with FP	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)	(0.82,-)

Table 3: Results of the similarity value for the different algorithms with varying number of hash functions and using version 1

Algorithm	$h = 2$	$h = 4$	$h = 6$	$h = 8$	$h = 8$
Shingling	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)
Minhash	(0.88,0.5)	(1.0,1.0)	(1.0,83)	(0.88,0.63)	(0.83,0.83)
LSH with FP	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)	(0.82,0.76)	(-,-)

Table 4: Results of the similarity value for the different algorithms with varying number of hashfunctions and using version

Time	Version1	Version2
Shingling	0.0219	0.0368
Minhash	0.1400	0.1621
LSH	0.1528	0.1647
LSH without FP	0.1560	0.1697

Table 5: Runtime results of the different algorithms

Jaccard Similarity	Version1	Version2
$k = 10$	(0.82,0.76)	(0.82,0.76)
$k = 20$	(0.8,0.73)	(0.8,0.73)

Table 6: Jaccarcy similarity results of two different shingle sizes

4 Instructions

Here we will go through the instructions on how to run our implemented function.The only file needed to be run is the *main.py* using the following command:

python main.py

Since we have two version for doing the Shingling, the user has to enter either 1 or 2 as an input depending on the version they want to test. The user has to enter the number when this statement appears in the terminal:

Enter the version to be performed

The default value for k is 10, if the user wants to change it, they can enter it after this statement:

Enter the shingle size

If the statement is left blank by clicking enter, the default value will be used.

Furthermore, the default value for the number of used hash function is 8 and can also be change by the user, by entering it after the statement:

Enter the number of hash functions

If again left blank, the default value be used here.

Finally, the user can also change the threshold or leave it as its default value, which is 0.6. The new threshold can be entered after this statement:

Enter a threshold

As an output, you will get the index or IDs (this depends on the version on shingling used) two pairs of documents, which are most similar as a tuple, once using the jaccard similarity and once using the signature similarity then the signature matrix is also printed and finally the lhslist.

5 Conclusions

As can be seen observed, in the results the first method using jaccard similarity provides the most accurate results, which is expected. The more power and memory we want to save by using the two other methods (minHash, LSH), the more inaccurate the results get. However, in all three used methods, depending on how the set threshold, the most similar documents are correctly predicted. While we do not see a memory and power advantage in the usage of the two other methods, this is due to the number of documents used, which are very few. If more than 100 documents were utilized, then we are sure the time advantage will be felt and observed in the results.