Akharaz
Majid

Project Document Similarity
Algorithme for Data Science

2020-11-09

1 Objective of the project

The objective of this programming assignment is to be able to implement a system for documentsimilarity, using the Min-Hash and Locality Sensitive Hashing methods presented in the lectures.

1.1 Implementation analysis

In this part we will discuss about the three key steps of this project.

Shingling of Documents: The first thing we do before we start studying document similarity is to present the document as a set that we get by transforming each document by the set of subtext it contains. This decomposition preserves the similarity that may exist between documents, indeed two similar documents will have two sets of equally similar subtext.

To do so, I was strongly inspired by the ”createShingle”function implemented during the course lab. This is a function that creates, among other things, a set list, each set contains all of the shingles present in a document. For example, m[0] returns the set of shingles (converted as int) present in document 0.
The strategy regarding white space was to remove it so that each document was represented by a long list of characters.

In addition to that, the function also returns a dictionary shingle_id having for key the id of the documentt and for value the shingle.

Computing Minhash Signature: Now that we have the representation of the documents as an integer set, we are going to calculate the signature matrix. This matrix has the remarkable property, according to the course, that the probability that two columns have the same values on a given row is equal to the jaccard similarity of the set corresponding to these columns.

The construction of this signature matrix is problematic indeed, swapping millions or even billions of lines is too expensive in terms of calculation. However I did not fully understand whether we should get around this problem in the statement or not. So, I got the signature matrix construction function present in the lab.

Locality-Sensitive Hashing:

First of all Then I transform the signature matrix that I have, which is in fact a list of list, into a matrix using numpy this will allow me subsequently to recover under list all at once while saving computation time.
The lsh function is the function which creates the candidate pairs.
We obtain these candidates by doing the following.
I am initializing two dictionaries:

1. dict_of_bucket: Which will simulate the bucket having for key the sub-parts of the documents and for value the list of documents having in common this sub-part of the documents;
2. Candidate: Has for key the list of peers having been found in dict of bucket and for value the number of times that these peers had in common the sublist.;

I also initialize a list which will allow to integrate each turns to store the sub-list of each document which I then place in dict of bucket.

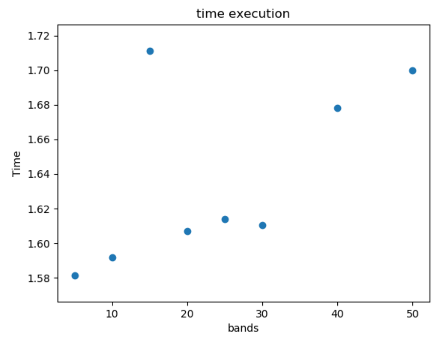
I took special care that the dictionary was reinitialized for each scan of all documents in each strip, in order to avoid false positives.

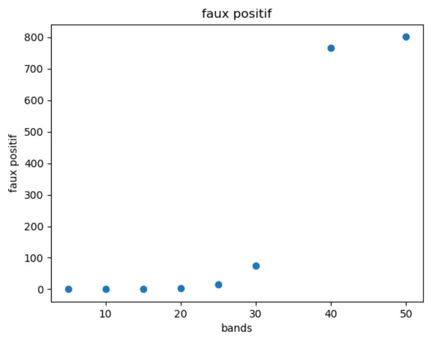
Indeed every time I enter a band, I treat all the existing peers in the dict of bucket dictionary then I reinitialize dict of bucket when I browse a new band.

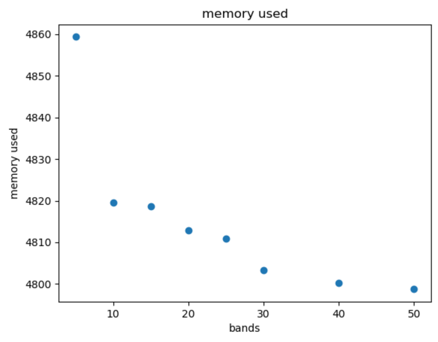
1.2 Evaluation de l'algoritme

In this section we will study the behavior of the algorithm depending on how we set the parameters and try to understand what is happening.

Size of shingle = 5, treshold = 0.5







Interpretation:

Here from left to right we have the execution time, the number of false positives and the memory consumed.

Regarding the run time, the overall trend and an increase in the latter as the number of tapes also increases. We can therefore deduce that the more we increase the number of bands, the slower the execution, it will be necessary to seek to make the right trade-off between number of bands and quality of the result.

Regarding the number of false positives, this remains roughly constant up to 25 bands and then increases very rapidly. This is due to the subdivision in mini element of the documents which make the calculation of similarity totally inappropriate which make the calculation, one concludes that it is very harmful for the results to have too high a number of bands.

And finally for the memory consumption it is obvious that its variation is opposite to the increase in bands.

These results illustrate the need to have to finely choose these parameters to optimize the performance of the algorithm. Indeed a careful look must be brought on the behavior of the number of bands chosen which in one case increases the execution time if not increases the use of the memory, a judicious choice would therefore be to take here in our example a number of bands between 20 and 30.