MASTER 2 DATA SCIENCE

Cours : Algorithms for Data Science.

Établissement : UNIVERSITE PARIS SACLAY

Enseignant : SILVIU MANIU

Sujet : Document Similarity - 25/10/2020

Etudiant : Panongbene Jean Mouhamed Sawadogo.

SOMMAIRE :

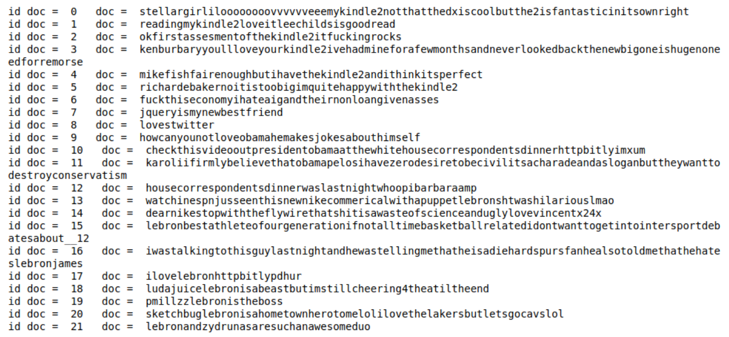
I. INTRODUCTION

In this work, we have implemented an algorithm that takes a set of documents and determines if there is a similarity between the two-to-two documents with a certain probability which is calculated using the jaccard similarity. To do this, we used the Jupyter language (therefore python) the description of how to run this program is given in the Readme.rd. For the methodology used to determine the similarity between two documents, we will give a description of our implementation choices throughout this report.

Format of documents used and parameters to be given : For the algorithm to work correctly, the documents to be compared must be in a single .txt file and the different documents are separated by a single line break.
We must also give our algorithm the threshold of comparability, that is to say the minimum probability for two documents to be considered equal.

II. DOCUMENTS PREPROCESSING

The documents are loaded into an array where the subscripts are the document identifiers and the values are the content of the documents, i.e. the text. Then, we remove the spaces between the documents so that each document becomes a long string of characters without spaces, that is to say a succession of alphanumeric characters. Thus, at the end the different documents will be represented by an array where the indices are the ids of the elements and the values are the strings of the documents without spaces.



```
id doc =  0   doc =  stellargirlilooooooovvvvvveeemykindle2notthatthedxiscoolbutthe2isfantasticinitsownright
id doc =  1   doc =  readingmykindle2loveitleechildsisgoodread
id doc =  2   doc =  okfirstassesmentofthekindle2itfuckingrocks
id doc =  3   doc =  kenburbaryyoullloveyourkindle2ivehadmineforafewmonthsandneverlookedbackthenewbigoneishugenone
edforremorse
id doc =  4   doc =  mikefishfairenoughbutihavethekindle2andithinkitsperfect
id doc =  5   doc =  richardebakernoitistoobigimquitehappywiththekindle2
id doc =  6   doc =  fuckthiseconomyihateaigandtheirnonloangivenasses
id doc =  7   doc =  jqueryismynewbestfriend
id doc =  8   doc =  lovestwitter
id doc =  9   doc =  howcanyounotloveobamahemakesjokesabouthimself
id doc =  10   doc =  checkthisvideooutpresidentobamaatthewhitehousecorrespondentsdinnerhttpbitlyimxum
id doc =  11   doc =  karoliifirmlybelievethatobamapelosihavezerodesiretobecivilitsacharadeandasloganbuttheywantto
destroyconservatism
id doc =  12   doc =  housecorrespondentsdinnerwaslastnightwhoopibarbaraamp
id doc =  13   doc =  watchinespnjusseenthisnewnikecommericalwithapuppetlebronshtwashilariouslmao
id doc =  14   doc =  dearnikestopwiththeflywirethatshitisawasteofscienceanduglylovevincentx24x
id doc =  15   doc =  lebronbestathleteofourgenerationifnotalltimebasketballrelatedidontwanttogetintointersportdeb
atesabout__12
id doc =  16   doc =  iwastalkingtothisguylastnightandhewastellingmethatheisadiehardspursfanhealsotoldmethathehate
slebronjames
id doc =  17   doc =  ilovelebronhttpbitlypdhur
id doc =  18   doc =  ludajuicelebronisabeastbutimstillcheering4theatiltheend
id doc =  19   doc =  pmillzzlebronistheboss
id doc =  20   doc =  sketchbuglebronisahometownherotomelolilovethelakersbutletsgocavslol
id doc =  21   doc =  lebronandzydrunasaresuchanawesomeduo
```

FIGURE 1 – table obtained after preprocessing the documents contained in the tweets.txt file

III. THE K-SHINGLE

After creating the document array, we now move on to building the k-shingles. To optimize the work, we implemented both the construction of the k-shingles of the documents and the construction of the vector containing the different k-shingles dictated on all the documents. We will need this vector for to build of the hash functions and the signature matrix.

We have noticed that the construction execution time of k shingles and the number of different k shingles increase exponentially with the size of the shingles k. As we can see from the pictures below.

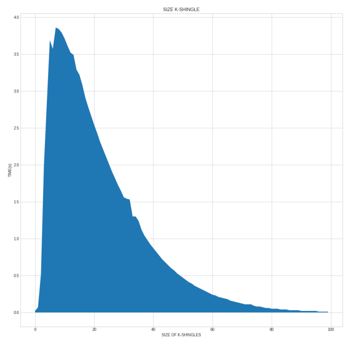


Figure 2 – This figure gives the execution time depending on the size of the k-shingles

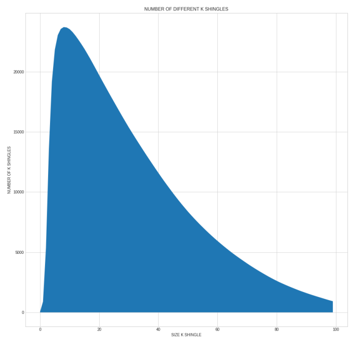
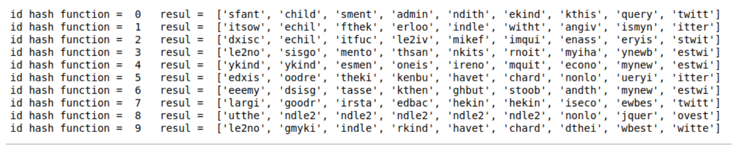


Figure 3 – This number gives the number of unique k-shingles based on the size of the k-shingles

IV. TO BUILD MINIHASHING MATRIX

To build the minihashing matrix, we create n hash functions. The creation of these hash functions is done simply by permutation of the different k-shingles generated during the previous step. We will use these hash functions to build the minihashing matrix which will then be of size number_of_hash_function * number_of_documents.



```
id hash function =  0   resul =  ['sfant', 'child', 'sment', 'admin', 'ndith', 'ekind', 'kthis', 'query', 'twitt']
id hash function =  1   resul =  ['itsow', 'echil', 'fthek', 'erloo', 'indle', 'witht', 'angiv', 'ismyn', 'itter']
id hash function =  2   resul =  ['dxisc', 'echil', 'itfuc', 'le2iv', 'mikef', 'imqui', 'enass', 'eryis', 'stwit']
id hash function =  3   resul =  ['le2no', 'sisgo', 'mento', 'thsan', 'nkits', 'rnoit', 'myiha', 'ynewb', 'estwi']
id hash function =  4   resul =  ['ykind', 'ykind', 'esmen', 'oneis', 'ireno', 'mquit', 'econo', 'mynew', 'estwi']
id hash function =  5   resul =  ['edxis', 'oodre', 'theki', 'kenbu', 'havet', 'chard', 'nonlo', 'ueryi', 'itter']
id hash function =  6   resul =  ['eeemy', 'dsisg', 'tasse', 'kthen', 'ghbut', 'stoob', 'andth', 'mynew', 'estwi']
id hash function =  7   resul =  ['largi', 'goodr', 'irsta', 'edbac', 'hekin', 'hekin', 'iseco', 'ewbes', 'twitt']
id hash function =  8   resul =  ['utthe', 'ndle2', 'ndle2', 'ndle2', 'ndle2', 'ndle2', 'nonlo', 'jquer', 'ovest']
id hash function =  9   resul =  ['le2no', 'gmyki', 'indle', 'rkind', 'havet', 'chard', 'dthei', 'wbest', 'witte']
```

FIGURE 4 – We can see in this image a small snippet of the minihashing matrix after using 10 hash functions

Note : We can notice that in our minhashing matrix we have words with numeric links. Indeed, we have used the words of the k-shingle for the sake of simplicity. We could also have used numbers but that doesn't have much effect on that.

V. TO BUILD SIMILARITY MATRIX

Once the mini hash matrix is build, we can easily to build the similarity matrix by calculating the jaccard similarity of the columns of the mini hash matrix. From this matrix and by defining a similarity threshold, we can say that two documents are similarity or not.
In the image below we can see the number of similariates according to the threshold

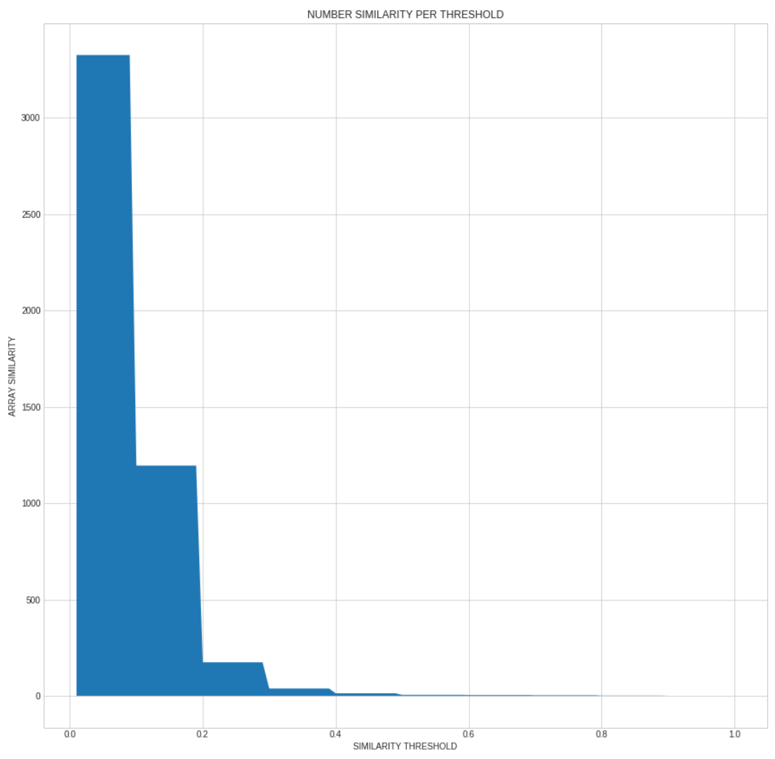


FIGURE 5 – Evolution of the number of similarities as a function of the threshold

VI. LOCALITY SENSITIVE HASHING

In this part, we have implemented the locality sensitive hash function. You can find a description of the version of Locality Sensitive Hashing implemented through this link.

For LOCALITY SENSITIVE HASHING, performance will vary depending on the similarity threshold and the number of bands we define.

On the images below we can see the evolution of the performance of the locality function as a function of the fixed threshold by comparing it to the similarity matrix

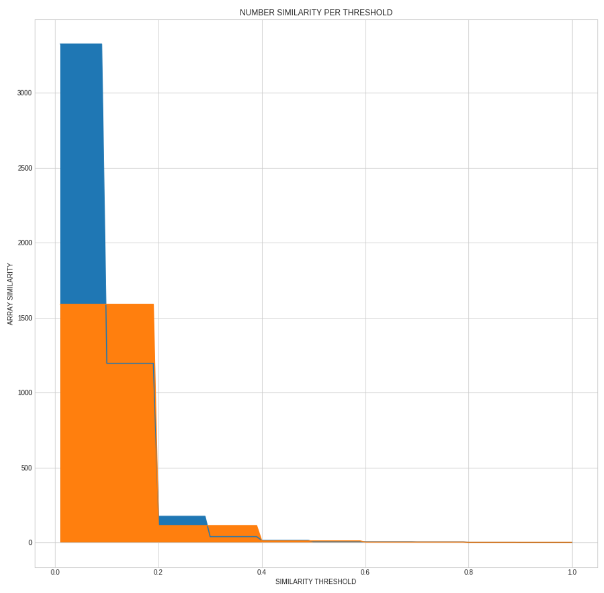


FIGURE 6 – Evolution of the number of similarities as a function of the threshold

In this image, we can clearly see that the locality function only works correctly for thresholds between 0.1 and 0.2 and greater than 0.3. In these cases, we observe empirically that the false positive rate is rather acceptable.