

University of Paris Saclay

Master 2 Data Science

Programming Project Document Similarity

Course:

Algorithms for Data Science

by:

mohammed amine TRABZI

2020

Overview :

The objective of this project is to be able to implement a system for document similarity evaluation, using the techniques of shingling and Min-Hash and Locality Sensitive Hashing. The documents we are going to use are available in the following link : <https://www.lri.fr/~maniu/tweets.txt> . We assume that each line in file tweets.txt is considered as document, for example line zero is considered as document zero , line two is document two , and so on ...etc.

To realize this project, we will implement a python code having command line that takes : (1)_the name of the file containing the documents, (2)_the similarity threshold in [0, 1], so that the command line is: ./ < program_name > < doc_file > < similarity > .

The code output will be the pairs of documents that have a Jaccard similarity above the predefined threshold, followed by their true similarity number.

1/ Implementation analysis:

Part zero :

In the top part of the code we implement code lines to prepare the data to be processed and some functions that we will use later to get documents similarity. The code of this is consisted of:

```
#Read arguments from command line -----
if len(sys.argv)>=3:
    t=float(sys.argv[2])
    file_name=sys.argv[1]
    print('Threshold t=%f'%(t))
    print('file_name is %s'%(file_name))
else:
    print('ERR : argument 2 is missing ! ')
    print('command-line should be as follow: python < program_name > < doc_file > < similarity >')
    sys.exit(2)
```

→ **Analysis :** this code will take arguments 1 and 2 from command line and use them in the code . If we miss to write the two arguments in the command line , the code will return error message and stop the code .

```
# Tune b & r parameters: -----
b=5 # number of bands
r= round((math.log(1/b))/math.log(t)) # number of rows per band
print('r=%d , b=%d'%(r,b))
```

→ **Analysis:** here we tune the number of bands “b” and the number of rows per band “r” using the formula : $t=(1/b)^{1/r}$

```
#Open the file :-----
file= open(file_name,"r")
documents = []
print('### Print each line of the document :')
for index in file:
    documents.append(str(index.strip()).lower())
    print(str(index.strip()).lower() +'\n')
```

→ **Analysis:** we store each line in the file tweets.txt in the list documents =[] , and we lower the capital letters . We also print the docs to .

Next, the code is divided into 4 parts :

- part 1 :transform the documents into sets of k-shingles
- part 2: create the signature matrix
- part 3: Locality Sensitive Hashing
- part 4: compute the true Jaccard similarity for the document pairs

Part 1: transform the documents into sets of k-shingles

In this part we will implement code that remove space between words , then we proceed to the application of shingling. A k-shingle of each document in the documents list is then chosen to split each document into sequence of k characters. To do, we need two loops for as shown in code below.

```
k = 5 #k-shingles

offset=k+1
sh_id=0
tot_num_shingles=0

shingle_id={} # for storing shingles and there ids
unique_shingles_list=[] # We store identical shingles without repetitions
matrix=[] # for storing the id of each shingle in all documents in the same order
# as shingles are ordered

for i in documents:
    line_shingles=[]
    line_compacted=''.join(j for j in i if j.isalnum() ) # remove space between words

    for n in range(len(line_compacted)-offset): # we start shingling the line
        shingle = line_compacted[n:n + k]
        line_shingles.append(shingle)

    tot_num_shingles+=len(line_shingles) # for each line we measure len(line_shingles)
# & we add its length to tot_num_shingles
# so that we get at the end the
# total number of generated shingles

# We need to know unique shingles & shingles for each document to use them later
stock_shingle_ids=set()
for a in line_shingles:
    if a not in shingle_id:
        sh_id+=1
        shingle_id[a]=sh_id
        unique_shingles_list.append(a)
        stock_shingle_ids.add(shingle_id[a])
matrix.append(stock_shingle_ids)
```

In this loop we save shingles ids in lists. Each document has its own list of shingle ids. Shingle ids of each document are saved in matrix. We also save the unique shingles with out repetition.

The following figure represents some shingles we get and there ids:

```
print(shingle_id)
```

Figure 1

```

'bstel': 0, 'stell': 1, 'te': 2, 'ellar': 3, 'llarg': 4, 'largi': 5, 'argir': 6, 'rgirl': 7, 'girli': 8, 'irl'
il': 9, 'rילו': 10, 'liloo': 11, 'llooo': 12, 'lloooo': 13, 'ooooo': 14, 'oooov': 15, 'oovvv': 16, 'oovvvv': 17,
'ovvvv': 18, 'vvvvv': 19, 'vvvve': 20, 'vvvee': 21, 'vveee': 22, 'veeee': 23, 'eemey': 24, 'eemyk': 25, 'emyki': 2
6, 'mykin': 27, 'ykind': 28, 'kindl': 29, 'indle': 30, 'ndle2': 31, 'dle2n': 32, 'le2no': 33, 'e2not': 34, '2not'
': 35, 'noth': 36, 'ottha': 37, 'tthatt': 38, 'hatt': 39, 'hatt': 40, 'atthe': 41, 'tthed': 42, 'thedx': 43, 'h
edxi': 44, 'edxis': 45, 'dxisc': 46, 'xisco': 47, 'iscoo': 48, 'scool': 49, 'coolb': 50, 'oolbu': 51, 'olbut': 5
2, 'lbutt': 53, 'butth': 54, 'utthe': 55, 'tthe2': 56, 'the2i': 57, 'he2is': 58, 'e2isf': 59, '2isfa': 60, 'isfan
': 61, 'sfant': 62, 'fanta': 63, 'antas': 64, 'ntast': 65, 'tasti': 66, 'astic': 67, 'stici': 68, 'ticin': 69, 'i
cini': 70, 'cinit': 71, 'inits': 72, 'nitso': 73, 'itsow': 74, 'tsown': 75, 'sownr': 76, 'ownri': 77, 'wnrig': 7
8, 'bread': 79, 'readi': 80, 'eadin': 81, 'ading': 82, 'dingm': 83, 'ingmy': 84, 'ngmyk': 85, 'gmyki': 86, 'dle2l
': 87, 'le2lo': 88, 'e2lov': 89, 'Love': 90, 'lovei': 91, 'oveit': 92, 'veitl': 93, 'eitle': 94, 'itlee': 95, 't
leec': 96, 'leech': 97, 'eechi': 98, 'echil': 99, 'child': 100, 'hilds': 101, 'ildsi': 102, 'ldsis': 103, 'dsisg
': 104, 'sisgo': 105, 'isgoo': 106, 'sgood': 107, 'goodr': 108, 'oodre': 109, 'bokfi': 110, 'okfir': 111, 'kfirs
': 112, 'first': 113, 'irsta': 114, 'rstas': 115, 'stass': 116, 'tasse': 117, 'asses': 118, 'ssesm': 119, 'sesme
': 120, 'esmen': 121, 'sment': 122, 'mento': 123, 'entof': 124, 'ntoft': 125, 'toft': 126, 'ofthe': 127, 'fthek
': 128, 'theki': 129, 'hekin': 130, 'ekind': 131, 'dle2i': 132, 'le2it': 133, 'e2itf': 134, '2itfu': 135, 'itfuc
': 136, 'tfuck': 137, 'fucky': 138, 'uckin': 139, 'cking': 140, 'kingr': 141, 'ingro': 142, 'ngroc': 143, 'bkenb
': 144, 'kenbu': 145, 'enbur': 146, 'nburb': 147, 'burba': 148, 'urbar': 149, 'rbary': 150, 'baryy': 151, 'aryyo
': 152, 'ryyou': 153, 'yyoull': 154, 'youll': 155, 'oulll': 156, 'ulllo': 157, 'lllov': 158, 'lllove': 159, 'llove

```

Part 2: create the signature matrix

The signature matrix is found by applying the meanhashing :

- we define permutation vector that contains ordered unique shingles ids. For each min hash function applied we shuffle the permutation vector. If number of min hash functions is “n” , so we have to shuffle the permutation vector “n” time.
- We need the matrix that contains the shingles ids of each document
- Application of the min hash function $h(S)$ =the index of the first row (in the permuted order) in which column S has the same id with document in the matrix.
- The min hash function is applied n times on the matrix. Each time , we shuffle the permutation vector and we save the results of min hash function for each document
- At the end we get a signature matrix that contains signature of each document

The min hash function is implemented with the following code(code seen during lab1):

```
def min_hash(doc, perm):
    for d in perm:
        if d in doc: return d
```

The code below shows the code implemented to find the signature matrix

```
n=b*r # number of min hash functions
# We generate signature matrix
permutation_list = list(range(len(unique_shingles_list)))
signature_matrix = []
for i in range(n):
    signature_matrix.append([])
    random.shuffle(permutation_list)
    for doc in matrix:
        minhash = min_hash(doc,permutation_list)
        signature_matrix[i].append(minhash)
```

Part 3: Locality Sensitive Hashing

3.1. create string representation for each document, containing the signature :

We divide the signature matrix into b bands of r rows. Each band is a small signature with r min hash functions.

The following code is implemented in order to:

- divide the signature matrix into bands
- convert values in each band into string

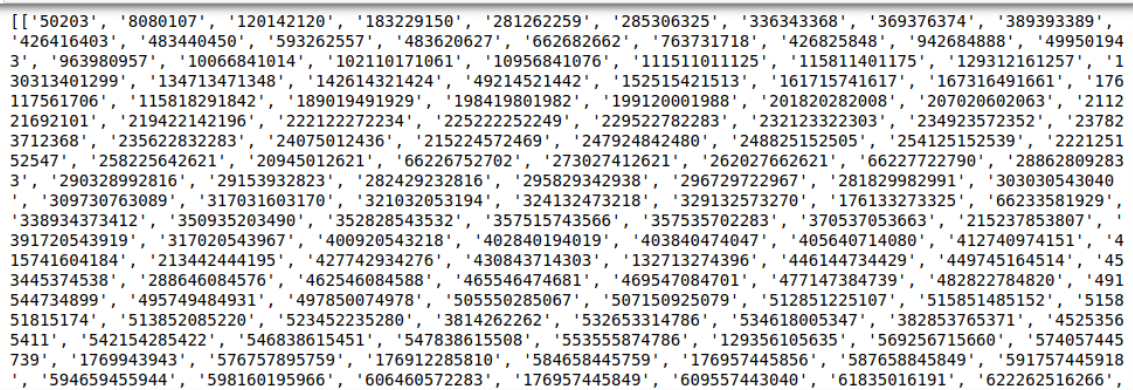
```
for k in range(b):
    x=k*r # using x is important to create bands
    vect.append([])
    for j in range(len(matrix)): # len(matrix) = number of docs
        string=[]
        for i in range(r):
            string.append(signature_matrix[i+(x)][j])
        shot_string=''.join(map(str,string))
        vect[k].append(shot_string)
```

→ The vect[] contains the signature of all bands of each document

The next figure highlights how the vect [] looks like :

print(vect)

Figure 2



3.2. Create dictionary & Finding candidat pairs:

Now we hash all values of vect[] into hash-table where we store the id of each document. If two bands are the same, they should hash to the same index in the hash-table. We store all the bands in the hash-table.

```
number_portions=len(matrix)*len(vect)
vect1=[]
bucket_size=number_portions*1000
for i in range(bucket_size):
    vect1.append([])

for i in range(b):
    for j in range(len(matrix)):
        index=hash(vect[i][j])%(bucket_size)
        if j not in vect1[index]:
            vect1[index].append(j)
        else:
            continue
```

→ **Analysis of code above:** we added if condition to avoid adding several times the same id document into the same index in the hash-table to avoid duplication of documents in the same emplacement in the hash-table.

Finding condidat pairs:

The next step is to filter vect1 [] so that we remain only indexes having more than two documents inside, and we store result in vect2 []. The following code do this task.

```
vect2=[]
for i in range(len(vect1)):
    if len(vect1[i])>=2:
        vect2.append(vect1[i])
```

Example of results found:

```
[[124, 362, 363],
 [126, 129, 130, 135],
 [107, 348],
 [127, 364],
 [124, 361, 362, 363],
 [231, 284],
 [244, 245],
 [61, 63],
 [110, 352],
 [109, 110, 352],
 [300, 83],
 [244, 245],
 [124, 368, 371],
 [109, 110, 352]]
```

Part 4: compute the true Jaccard similarity for the document pairs, and output them if above the thresh-old

In this step, we calculate true jaccard similarity between all our condidate pairs and we print only those who have jaccard similarity greater than or equal to the threshold t . ([The jaccard similarity function used here is the same we have seen during lab1](#))

```
for i in range(len(vect2)):
    for j in range(len(vect2[i])):
        for k in range(len(vect2[i])):
            if j !=k:
                j_sim = jaccard_similarity(matrix[vect2[i][j]],matrix[vect2[i][k]])
                if j_sim>=t:
                    print ('doc%d-doc%d jaccard-sym %f'%(vect2[i][j],vect2[i][k],j_sim))
                else:
                    continue
```

The figure bellow illustrates the results :

```
doc124-doc363 jaccard-sym 0.714286
doc363-doc124 jaccard-sym 0.714286
doc107-doc348 jaccard-sym 0.568627
doc348-doc107 jaccard-sym 0.568627
doc124-doc363 jaccard-sym 0.714286
doc363-doc124 jaccard-sym 0.714286
doc244-doc245 jaccard-sym 0.666667
doc245-doc244 jaccard-sym 0.666667
doc110-doc352 jaccard-sym 0.672131
doc352-doc110 jaccard-sym 0.672131
doc109-doc352 jaccard-sym 0.555556
doc110-doc352 jaccard-sym 0.672131
doc352-doc109 jaccard-sym 0.555556
doc352-doc110 jaccard-sym 0.672131
doc244-doc245 jaccard-sym 0.666667
doc245-doc244 jaccard-sym 0.666667
doc109-doc352 jaccard-sym 0.555556
doc110-doc352 jaccard-sym 0.672131
doc352-doc109 jaccard-sym 0.555556
doc352-doc110 jaccard-sym 0.672131
```

Figure 3 : True jaccard similarity of candidate pairs

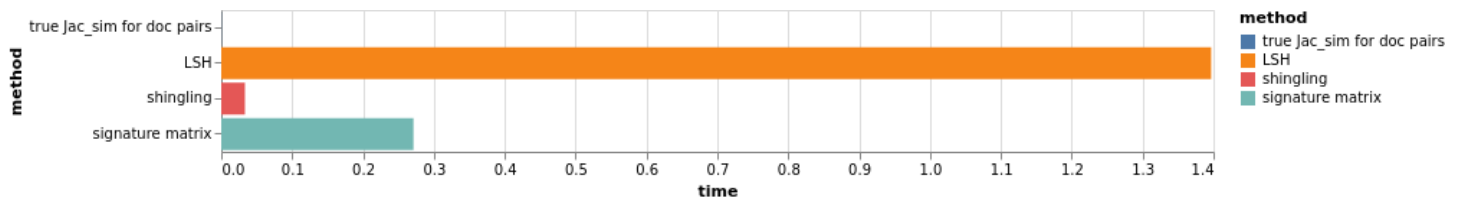
2/ Experimental evaluation:

2.1 – Time Analysis:

2.1.(1) Time execution of each part of the code:

The code has four principal parts, so for each part we measure time execution, results in figure that follows: (we can see that LSH part takes the greatest portion of execution time)

Figure 4



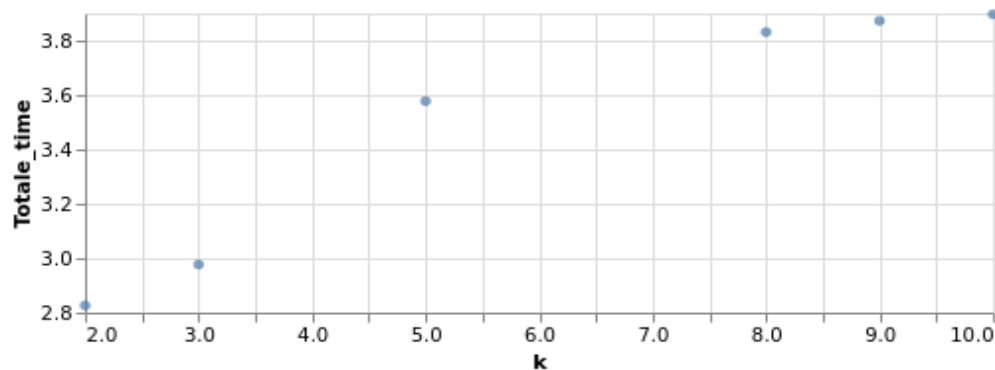
After running the code , the time measurment is displayed at the end , as follows :

Figure 5

```
Time measurements:
*****
Elapsed time to transform documents into sets of k-shingles = 0.030285
Elapsed time to create the signature matrix = 0.287049
Elapsed time to use Locality Sensitive Hashing = 1.470107
Elapsed time to compute the true Jaccard similarity for the document pairs = 0.000366
Total time = 1.787807
```

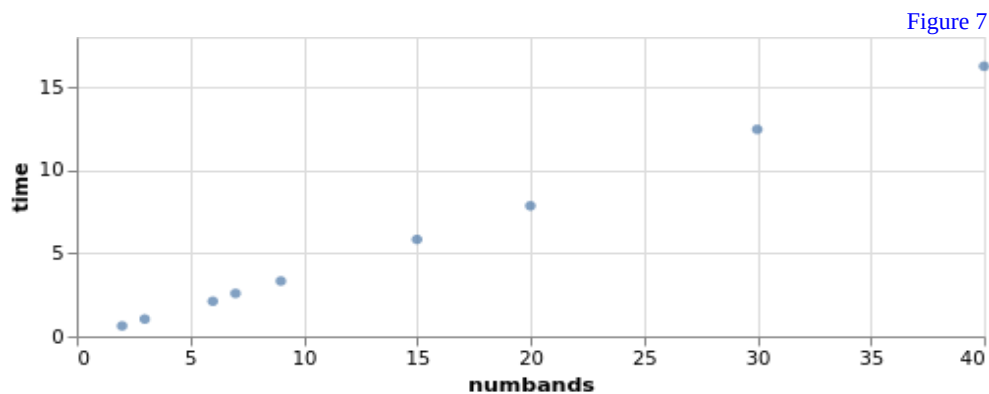
2.1.(2) Time execution of the code in function of k-shingles:

Figure 6



→ **Analysis:** The time execution increases from k=2 to k=8 (but decreases slowly when arriving to k=8), then it tends to stay stable and maybe decreases

2.1.(3) Time execution of the code in function of number of min-hash functions



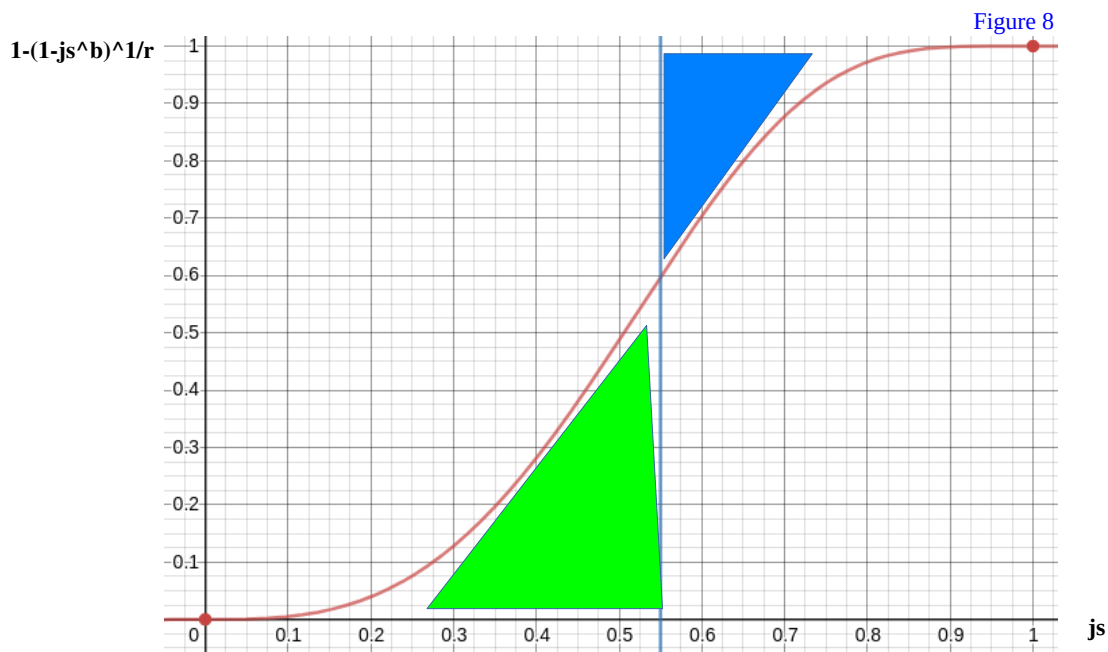
→ **Analysis:** We can clearly notice that execution time of the code increases more with number of bands

2.2_ Theoretical Analysis:

The theory says that if we use “b” band and “r” , then, the probability that a pair of documents _ having jaccard similarity js % _ to be candidate pairs is given by the following formula :

$1-(1-js^b)^{1/r}$. For example, if we choose $b=5$ and $r=3$, the threshold t is calculated by formula : $t=(1/b)^{1/r}$. In this case t is approximately $t=0.55$.

So documents having jaccard similarity at least 0.55 are likely to be true positives when using LSH technique to infer the documents similarity. However, there is always errors , for the reason that we may have false positives even though we are beyond the threshold t .



→ **Analysis :** The bleu zone corresponds to the cases of False positives , and the green zone is False negatives, and the zone beyond the Threshold (except bleu zone) is corresponding to True positive cases.