Locality Sensitive Hashing

In this assignment you will implement localitiy sensitive hashing step-by-step and apply that on detecting almost duplicate names. You are required to implement the functions as instructed. Do not change the signatures of the functions.

Submission instruction

Submit a text file with extension .py containing the functions you have implemented. You can simply use the "download as" option from Jupyter notebook to export the notebook as a .py file. The filename should be LSH_{YOUR_ROLL_NO}.py where {YOUR_ROLL_NO} is your 6 letter roll number. For example, a sample file name would be LSH_CS1702.py. Please take this seriously and name the file correctly.

Important: do not change the function signatures

You need to implement the functions given in this notebook. You may define more functions at your will if you need to, but make sure you implement the functions given and do not change their signatures. Before submitting, remove unnecessary print / collect statements in order to avoid too much console output.

Setup environment

We need to setup pyspark and pydrive.

In [2]:
```
!pip install pyspark
!pip install -U -q PyDrive
!apt install openjdk-8-jdk-headless -qq
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.6/dist-packag
es (3.0.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.6/dist-p
ackages (from pyspark) (0.10.9)
openjdk-8-jdk-headless is already the newest version (8u272-b10-0ubuntu1~18.0
4).
0 upgraded, 0 newly installed, 0 to remove and 11 not upgraded.
```

Follow the interactive instructions for accessing the file in Google drive.

In [3]:
```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

Load the appropriate file

The files are shared on Google drive, with the following ids given. Choose to keep the appropriate line, use the corresponding filename and comment the others. After executing, you should be able to see the file in the files section on the left panel.

You may face troubles trying to run your code on the full data, so it is totally okay to run it on one

of the smaller files.

In [4]:
```
#id='1aoZykfz5GLGGw3lRA86ogd7yCsgkgHoT' # for titles-small.txt
id='1IPssUa3m-zfWmvVbLtI-lKhJbeZjIgBg' # for titles-10k.txt (with 10k title
# id='1RzTA4iOfH3bOiyG2kDaxweUuj4AEcMda' #
downloaded = drive.CreateFile({'id': id})
downloaded.GetContentFile('titles-10k.txt')
```

Parameters

Let us set our parameters. You are free to change these around and experiment. However, do not hardcode them anywhere else in the code.

In [5]:
```
#The following tunable parameters are considered for the assignment
# Number of hash functions
n = 20
# size of each shingle
k = 3
# Number of bands
b = 4
```

Note: Please open the notebook in Google colab otherwise it might show indentation error in local sytem because here 2 whitespace works for indentation but in local system, jupyter notebook takes 4 whitespaces or tab for indentation

1. Converting a string to shingles (of characters) [Marks: 5]

In our setup, the items we would compare are movie titles (strings). You would have to convert names to sets of k-shingles (of characters), *after removing the whitespaces* and *converting to lowercase*. For example, the 3-shingling of the name `'Die hard'` would be `['die', 'ieh', 'eha', 'har', 'ard']`.

Implement the following function which takes a string and k as input and outputs the list of unique shingles generated from the string.

In [6]:
```
def string_to_shingles(name, k):
  shingles=[]
  name=name.lower() # it converts given string to all lower characters stri
  string="".join(name.split()) # it removes the whitepaces from the string
  #print(str)
  for s in range(len(string)-k+1):
    shingles.append(string[s:s+k]) # taking k-length shingles
  return list(set(shingles)) # to get unique shingles, first convert list t
  # returns list of unique shingles generated from the string
```

Also test your code.

In [7]:
```
print(string_to_shingles('Die hard', 3))
```

```
['har', 'die', 'ard', 'ieh', 'eha']
```

2. Generating the shingle - item matrix from the whole data [Marks: 20]

For each title, first give it an ID. You can do it by the `zipWithIndex()` operation on Spark RDDs. Try testing it first.

Then, for each title, get the list of shingles in map and for each unique shingle, get the list of IDs using reduce.

Tips: when your input RDD is a list of tuples of the following form:
`[('Die hard', 0), ('Die another day', 1), ('Tomorrow never dies', 2), ('Chup ke chup ke', 3), .... ]`

for each tuple `t`, you may use `t[0], t[1]` etc to access the first, second (and so on) elements of the

The "titles-10k.txt" file has total 10k movie titles which are loaded in the titles RDD.

In [8]:
```
from pyspark import SparkContext, SparkConf
sc = SparkContext.getOrCreate()

# Use appropriate file path here
titles = sc.textFile("titles-10k.txt")

# Get the number of titles
N = titles.count()  # N=10000 = total number of movie titles
itemsByShingles = titles.zipWithIndex() # associate a unique id for each ti
```

Here, we convert each title to a 3-shingles and keyval is a list of key-value pair in which key has title and value is a list of 3-shingles for corresponding title.

In [9]:
```
keyval=titles.map(lambda x: (x,string_to_shingles(x,k)))
#keyval.takeSample(False,5) # taking a sample of size 5 without replacement
```

Here, keyval1 returns the key-value pair where key is a list of 3-shingles for a particular title and corresponding movie id.

In [14]:
```
keyval1=itemsByShingles.map(lambda x: (string_to_shingles(x[0],k),x[1]))
#keyval1.takeSample(False,5)
```

From the list of " ([list of 3-shingles] , movie-id) ", I make a pair of a particular shingle and movie id.

In [15]:
```
key_val=keyval1.flatMap(lambda n: [(x, n[1]) for x in n[0]])
#key_val.takeSample(False,5)
```

In [16]:
```
#print(key_val.count())
```

Here, result returns a key-value pair where key is a shingle and and value is the list of movie id associated with a particular shingle which is the objective of this part.

In [24]:
```
result=key_val.combineByKey(lambda v:[v],lambda x,y:x+[y],lambda x,y:x+y)
result.takeSample(False,5)
```

Out[24]:
```
[('ulw', [988, 1451, 5609, 6125, 6898, 9356, 9504, 9602]),
 ('chä', [7889]),
 ('llw', [1054, 1951, 6024, 8702, 9575, 9732]),
 ('ayu', [2739, 3707]),
 ('m,t', [4006, 6058, 9482])]
```

In [25]:
```
#result.count() # returns total number of unique shingles
```

At the end of this step, the `itemsByShingles` RDD should have the list of (shingle, [list of movie ids]).

If you display your output, it should look like:

[(shingle, [list of movie ids]), (shingle, [list of movie ids]), ...]

However, do not keep collect or large print statements in your code at the time of submitting.

3. Computation of min-hash signature matrix [Marks: 10 + 10 = 20]

Instead of using random permutations, you will implement min-hash function using Murmurhash (v3) functions, as discussed in the class. The input to your function should be a *title* (which corresponds to a row) and the output should be a number (hash value).

Recall the outline of the min-hash signature matrix algorithm:

```
1. For each row r BEGIN
2.  Compute h_1(r), h_2(r), ..., h_n(r)
3.   For each column j BEGIN
4.     If the j-th column has 1 in row r
5.     For each i = 1, 2, ..., n BEGIN
6.       set SIG_{i,j} = min(SIG_{i,j}, h_i(r))
7.     END
8.    END IF
9.   END
10. END
```

For each shingle, we have the list of movie IDs which contain the shingle. So, for each shingle (each row), we can perform the actions in lines 5-7 only for the movie IDs that are present in the list. For any other j, the j-th column does not have a 1 in row r.

Do not create a SIG matrix and try to update it. Instead, note that for every 1-entry in the shingle-id matrix (that is, every id in the list corresponding to a shingle), the corresponding (i, id)-th entry of the signature matrix may get potentially updated. Simply output the tuple ((i, id), h_i(text)) in another map. The final value of the (i,id)-th entry of the signature matrix is the minimum of all such h_i(text) values obtained in this process. That minimum can be computed by another reduce process.

(a) The map [Marks: 10 out of 20]

To make it easier, you may implement an `update_signature` function which takes the shingle (text), the list of ids associated with the shingle and the total number of shingles (for computing the hash values). It should simply return the tules `((i, id), h_i(text))` for all i = 1, ..., n.

Note: There has been a correction here. Instead of N (the number of movie ids), we need to pass the number of Shingles here.

In [27]:
```
!pip3 install mmh3
```

```
Requirement already satisfied: mmh3 in /usr/local/lib/python3.6/dist-packages
(2.5.1)
```

Here, mmh3 returns the key-value pair where key is a tuple of hash-function index and movie id and value is hash-value. Actually, here, key is $(i, j)^{th}$ entry in signature matrix and value is value in that $(i, j)^{th}$ entry for the signature matrix.

In [29]:
```
import mmh3
def update_signature(text, ids, numOfShingles):

    # h = [mmh3.hash(text,i) % (total number of shingles) for i in range(n)

  min_sig = []
  for i in range(n):
    for j in range(len(ids)):
      temp=[]
      temp2=[]
      temp.append(i)
      temp.append(ids[j])
      temp2.append(tuple(temp))
      temp2.append(mmh3.hash(text,i) % (numOfShingles))
      min_sig.append(tuple(temp2))
```

```
16
17
18    return min_sig
```

(b) The reduce [Marks: 10 out of 20]

Now that you have all `((i, id), h_i(shingle))` tuples output by the map, use reduce to compute the minimum of `h_i(shingle)` for every `(i,id)` key. This would produce the signature matrix in a sparse matrix representation.

You may actually implement the map function above and call map and reduce together later as well.

Here, new_key_val returns the list of key-value pair where where key is a pair of hash-function id $i = 0$ to $(n - 1)$ and a movie-id. Value in the key-value pair is value of the hash-function where input is a unique 3-shingle. Here, key-value pairs are corresponding to the unique 3-shingle and associated list of movie-ids.

In [31]:
```
numOfShingles = result.count() # Calculate, you need to pass this to the up

# You should use map with the update_signature function and a reduce
#new_key_val=result.flatMap(lambda n: [(x, n[1]) for x in n[1]])

new_key_val=result.map(lambda x: update_signature(x[0],x[1],numOfShingles)
#new_key_val.takeSample(False,2)
new_key_val = new_key_val.flatMap(lambda list: list)
```

In [32]:
```
#print(result.count(),new_key_val.count())
```

Now, signature returns the signature matrix which is our objective for this part.It is in the form of $((i, j), k)$ where, (i, j) represents the $(i, j)^{th}$ entry of the signature matrix and k is the value of the hash function. **Note:** Here, size of the signature matrix is 199960 , not $20 \times 10000 = 200000$. The reason is, in the original dataset "titles-10k.txt", there are 2 movie titles "99" and "S1" for which I can't generate 3-shingle. So, size of the signature matrix will be $20 * (10000 - 2) = 199960$.

In [33]:
```
signature = new_key_val.reduceByKey(min)
signature.takeSample(False,5)
```

Out[33]:
```
[((15, 7758), 1104),
 ((18, 6315), 899),
 ((4, 7382), 346),
 ((6, 4133), 525),
 ((17, 9220), 1166)]
```

In [34]:
```
#signature.count()
```

The RDD signature should be of the following form:
`[((0, 1), 5), ((0, 3), 56), ... ]`

where each tuple `((i,j),v)` represents the (i, j)-th entry of the signature matrix with value v.

4. Implement the banding [Marks: 15]

Now configure your number of bands b (a divisor of number of hash functions n) and implement the candidate pair computation. Any pair of movies which agree completely in their signature on at least one band should be output as candidate pairs.

Here, signature matrix is divided into 4 parts by rows since there are 4 bands and number of rows in each band is 5.

In [35]:
```
#r=n/b = 20/4 = 5

band1=signature.filter(lambda x : x[0][0] >=0 and x[0][0]<=4)
band2=signature.filter(lambda x : x[0][0] >=5 and x[0][0]<=9)
band3=signature.filter(lambda x : x[0][0] >=10 and x[0][0]<=14)
band4=signature.filter(lambda x : x[0][0] >=15 and x[0][0]<=19)

candidate_pairs = []
```

Since, I get the candidate pairs in band1 where pair of movies completely agree in their signature, so I will not check the other bands. The logic to find the candidate pair which I have used is: first for a band1, I listed the repeating hash-values in the signature matrix along with their entry means to which row and column they belong to. After that, pair-wise columns in which hash-values are same, I compare the other 3 hash-values for those pair-wise coulumns. If all values are matches, then I return the those column pairs.
Note: Here, I return only the candidate pairs of movie ids. Later, I will return candidate pairs by movie titles.

In [36]:
```
#band1

temp01= band1.map(lambda x :(x[1],x[0]))
temp101=temp01.groupByKey().map(lambda x : (x[0], list(x[1])))
#print(temp101.takeSample(False,2))
def ls(x):
  f2=[]
  for i in range(len(x[1])):
    for j in range(i+1,len(x[1])):
      f1=[]
      if x[1][i][0]== x[1][j][0]:
        l1=[]
        l1.append(x[1][i][1])
        l1.append(x[1][j][1])
        f1.append(tuple(l1))
        f1.append(x[1][j][0])
        #f1.append(x[0])
        f2.append(tuple(f1))
  return f2
filt1=temp101.flatMap(lambda x : [ls(x)])
#filt1.take(2)
filt1=filt1.flatMap(lambda l: l)
#filt1.count()
fin1=filt1.groupByKey().map(lambda x : (x[0], list(x[1])))
#print(fin1.takeSample(False,2))
#fin1.count()
def check(x):
  if len(x[1])==4:
    return x[0]
res1=fin1.map(lambda x: check(x))
res1=res1.filter(lambda x: x is not None)
res1.collect()
```

Out[36]:

```
[(7456, 9558),
 (3374, 988),
 (3374, 4628)
```

Now, instead of candidate pairs as movie ids, I will return the candidate pairs by movie titles. I used the above code (just copy-pasted) and instead of movie-id, I replaced the movie-titles.

In [37]:

```
keyval_modified=keyval.map(lambda x: (string_to_shingles(x[0],k),x[0])) #it
#list of 3-shingles for a particular title and corresponding movie id
#keyval_modified.takeSample(False,5)
key_val_modified=keyval_modified.flatMap(lambda n: [(x, n[1]) for x in n[0]
# a pair of a particular shingle and movie id.
#key_val_modified.takeSample(False,5)
result_modified=key_val_modified.combineByKey(lambda v:[v],lambda x,y:x+[y]
# where key is a shingle and and value is the list of movie id associated w
#result_modified.takeSample(False,5)
#numOfShingles = result.count() # Calculate, you need to pass this to the u

#signature = # Implement the rest here

# You should use map with the update_signature function and a reduce
#new_key_val=result.flatMap(lambda n: [(x, n[1]) for x in n[1]])

new_key_val_modified=result_modified.map(lambda x: update_signature(x[0],x[
#new_key_val_modified.takeSample(False,2)
#signature.take(5)
new_key_val_modified = new_key_val_modified.flatMap(lambda list: list)
signature_modified = new_key_val_modified.reduceByKey(min)
#r=n/b = 20/4 = 5
#x = sc.parallelize([1,2,4,5])
##max_val = x.reduce(lambda a, b: a if a > b else b)
#print(max_val)
band1_modified=signature_modified.filter(lambda x : x[0][0] >=0 and x[0][0]
band2_modified=signature_modified.filter(lambda x : x[0][0] >=5 and x[0][0]
band3_modified=signature_modified.filter(lambda x : x[0][0] >=10 and x[0][0
band4_modified=signature_modified.filter(lambda x : x[0][0] >=15 and x[0][0

candidate_pairs = []
#band1_modified

temp01_modified= band1_modified.map(lambda x :(x[1],x[0]))
temp101_modified=temp01_modified.groupByKey().map(lambda x : (x[0], list(x[
#print(temp101.takeSample(False,2))
def ls(x):
  f2=[]
  for i in range(len(x[1])):
    for j in range(i+1,len(x[1])):
      f1=[]
      if x[1][i][0]== x[1][j][0]:
        l1=[]
        l1.append(x[1][i][1])
        l1.append(x[1][j][1])
        f1.append(tuple(l1))
        f1.append(x[1][j][0])
        #f1.append(x[0])
        f2.append(tuple(f1))
  return f2
filt1_modified=temp101_modified.flatMap(lambda x : [ls(x)])
#filt1.take(2)
filt1_modified=filt1_modified.flatMap(lambda l: l)
#filt1.count()
fin1_modified=filt1_modified.groupByKey().map(lambda x : (x[0], list(x[1])))
#print(fin1.takeSample(False,2))
#fin1.count()
def check(x):
  if len(x[1])==4:
    return x[0]
res1_modified=fin1_modified.map(lambda x: check(x))
```

```
62 res1_modified=res1_modified.filter(lambda x: x is not None)
63 res1_modified.collect()
```

Out[37]:
```
[('Statistinnen des Lebens', "Paul Wang's Destiny"),
 ('Statistinnen des Lebens', 'The Thread of Destiny'),
 ("Paul Wang's Destiny", 'The Thread of Destiny'),
 ('Caught in the Rain', 'Caught in a Park'),
 ('The Fighting Brothers', 'The Fighting Hope'),
 ('The Fighting Brothers', 'Fighting Mad'),
 ('The Fighting Brothers', 'The Fighting Trail'),
 ('The Fighting Hope', 'Fighting Mad'),
 ('Fighting Mad', 'The Fighting Trail'),
 ('Threads of Destiny', 'Destiny'),
 ("What's Sauce for the Goose", "That's Him"),
 ('His Ups and Downs', 'Ups and Downs'),
 ('Hands Down', 'Pines Up and Palms Down'),
 ("The Widow's Secret", "The Widow's Kids"),
 ('In the Window Recess', 'The Face at the Window'),
 ("The Governor's Ghost", "The Governor's Lady"),
 ('The Joneses Have Amateur Theatricals', 'In the Theatrical Business'),
 ('His Prehistoric Past', 'Prehistoric Poultry'),
 ('The Price He Paid', 'The Price She Paid'),
 ("Paul J. Rainey's African Hunt", "Rainey's African Hunt")]
```

The candidate pairs should be of the form

`[(movie1, movie2), (movie1, movie2), ... ]`

5. Optional:

Optionally, you may also want to compute the pairwise actual Jaccard similarities for each candidate pair and test the number of false positives and negatives for a similarity threshold s.

In [ ]:
```
1
```