### CS F469

### Information Retrieval

### Assignment - 2

### Preprocessing and Extraction of dataset

The dataset we picked has a large number of news articles from various publications like New York Times, Breitbart, CNN, Business Insider, the Atlantic, Fox News, Talking Points Memo, Buzzfeed News, National Review, New York Post, the Guardian, NPR, Reuters, Vox, and the Washington Post.

import pickle  
from pathlib import Path  
import numpy as np  
import pandas as pd  
root = Path(".")

article\_1 = pd.read\_csv('articles1.csv')  
news\_content = article\_1['content'][:500]

Instead of using the raw data of the documents, we first remove all the characters that are not alphanumeric and convert the letters to lowercase which would allow documents with similar content have same set of shingles. The assumption is that Documents that have lots of shingles in common have similar text, even if the text appears in different order

documents = list()  
  
for file in news\_content:  
 ans = list()  
 for i in range(len(file)):  
 if file[i].isalnum():  
 ans.append(file[i].lower())  
 else:  
 ans.append(' ')  
   
 res = list()  
 for i in range(len(ans) - 1):  
 if ans[i] == ans[i + 1] and ans[i] == ' ':  
 continue  
 else:  
 res.append(ans[i])  
   
 if len(ans) > 0:  
 res.append(ans[-1])  
 documents.append(''.join(res))

### K - Shingles

Given the set of documents and the parameter K we apply the method of K-shingling and extract all the unique consecutive substrings of length K from the documents.

def get\_shingle\_set(documents, k):  
 shingles = set()  
 shingle\_doc\_id = dict()  
  
 for j in range(len(documents)):  
 for i in range(len(documents[j]) - k + 1):  
 cur\_shingle = documents[j][i : i + k]  
  
 shingles.add(cur\_shingle)  
   
 if shingle\_doc\_id.get(cur\_shingle) == None:  
 shingle\_doc\_id[cur\_shingle] = [j]  
 else:  
 shingle\_doc\_id[cur\_shingle].append(j)  
   
 return (shingles, shingle\_doc\_id)

### No of unique shingles vs K

for k in range(2, 11):  
 print(k, len(get\_shingle\_set(documents, k)[0]))

2 994  
3 9223  
4 47147  
5 149523  
6 358639  
7 669031  
8 1040926  
9 1432704  
10 1802365

Considering the number of shingles, we chose K = 3 for our implementation with around 47,000 unique shingles extracted from the documents

K = 3  
shingles, shingle\_doc\_id = get\_shingle\_set(documents, K)  
shingles = sorted(list(shingles))

### Construction of the incident / occurence matrix

Using the extracted K-shingles we now create the incident or the occurence matrix which tells us the presence of shingle in any document. Since this matrix is too sparse, its not used to find similar documents and thats where LSH comes into play.

incident\_matrix = np.zeros(shape=(len(shingles), len(documents)))  
incident\_matrix

array([[0., 0., 0., ..., 0., 0., 0.],  
 [0., 0., 0., ..., 0., 0., 0.],  
 [0., 0., 0., ..., 0., 0., 0.],  
 ...,  
 [0., 0., 0., ..., 0., 0., 0.],  
 [0., 0., 0., ..., 0., 0., 0.],  
 [0., 0., 0., ..., 0., 0., 0.]])

shingle\_id = dict()  
for i in range(len(shingles)):  
 shingle\_id[shingles[i]] = i  
  
for shingle, doc\_ids in shingle\_doc\_id.items():  
 for doc\_id in doc\_ids:  
 incident\_matrix[shingle\_id[shingle]][doc\_id] = 1  
  
incident\_matrix

array([[0., 0., 0., ..., 0., 0., 0.],  
 [0., 1., 0., ..., 1., 1., 0.],  
 [0., 0., 0., ..., 0., 0., 0.],  
 ...,  
 [0., 0., 0., ..., 0., 0., 0.],  
 [0., 0., 0., ..., 0., 0., 0.],  
 [0., 0., 0., ..., 0., 0., 0.]])

my\_path = root / "Pickled\_files" / "Incident\_Matrix"  
dbfile = open(my\_path, 'wb')  
pickle.dump(incident\_matrix, dbfile)   
dbfile.close()  
  
my\_path = root / "Pickled\_files" / "Shingles"  
dbfile = open(my\_path, 'wb')  
pickle.dump(shingles, dbfile)   
dbfile.close()  
  
my\_path = root / "Pickled\_files" / "Shingle\_id"  
dbfile = open(my\_path, 'wb')  
pickle.dump(shingle\_id, dbfile)   
dbfile.close()  
  
my\_path = root / "Pickled\_files" / "documents"  
dbfile = open(my\_path, 'wb')  
pickle.dump(documents, dbfile)   
dbfile.close()

for i in range(len(documents)):   
 matrix = incident\_matrix[0 : len(incident\_matrix), i]  
 print(np.count\_nonzero(matrix == 1), matrix.shape[0])

## Locality Sensitive Hashing

import pickle  
from pathlib import Path  
import numpy as np  
import pandas as pd  
import random  
from itertools import combinations  
root = Path(".")

article\_1 = pd.read\_csv('articles1.csv')

lsh\_stats = {  
 'b' : list(),  
 'r' : list(),  
 'Number of hash functions' : list(),  
 'True Positive' : list(),  
 'True Negative' : list(),  
 'False Positive' : list(),  
 'False Negative' : list(),  
 'Candidate Pair count' : list()  
}

We first load the pickled documents into RAM. The first step of LSH is the creation of the signature matrix. Since incident matrix is too sparse and dimension heavy, we create a compact signature matrix using the minhashing technique. get\_hash\_functions(hsh\_cnt, mod) generates hsh\_cnt random hash functions which will be used for minhashing. The generated hash functions are of the form %

MinHashing Technique : The signature matrix is created using the minhash technique where we apply all of the hash functions row-wise and update the minimum hash value in each document which has the current shingle. The general idea of LSH is to find a algorithm such that if we input signatures of 2 documents, it tells us that those 2 documents form a candidate pair.

The signature matrix is divided into b bands of r row each, for each band we hash its portion of the column in the corresponding bucket, Candidate column pairs are those that hash to the same bucket for at least 1 band. The results are verified by actually checking if those documents actually have similarity >= threshold. The top 3 similar documents, number of candidate pairs along with statistics depicting performance of LSH (True Positive, True Negative, False Positive, False Negative) is returned.

class LSH:  
 def \_\_init\_\_(self, b, r, hash\_cnt) -> None:  
 self.unpickle()  
 self.hash\_functions = self.get\_hash\_functions(hash\_cnt, len(self.incident\_matrix))  
 self.min\_hash()  
 self.make\_candidate\_pairs(b, r)  
  
 def get\_hash\_functions(self, hsh\_cnt, mod):   
 hash\_function = list()  
 for \_ in range(hsh\_cnt) :   
 a = random.randint(1, 100)  
 b = random.randint(1, 100)  
 hash\_function.append((a, b, mod))  
 return hash\_function  
  
 def min\_hash(self):  
 self.signature\_matrix = np.zeros(shape=(len(self.hash\_functions), len(self.documents)))  
 self.signature\_matrix.fill(10 \*\* 18)  
  
 for shingle\_row in range(len(self.incident\_matrix)):   
 cur\_hsh = 0  
 shingle\_row\_val = np.array(self.incident\_matrix[shingle\_row])  
 ids\_take = np.nonzero(shingle\_row\_val)  
   
 for (a, b, mod) in self.hash\_functions:   
 hsh\_value = ((shingle\_row + 1) \* a + b) % mod  
  
 for doc\_id in ids\_take[0]:  
 if shingle\_row\_val[doc\_id] == 1:  
 self.signature\_matrix[cur\_hsh][doc\_id] = min(self.signature\_matrix[cur\_hsh][doc\_id], hsh\_value)  
 cur\_hsh += 1  
  
 my\_path = root / "Pickled\_files" / "Signature\_Matrix"  
 dbfile = open(my\_path, 'wb')  
 pickle.dump(self.signature\_matrix, dbfile)   
 dbfile.close()  
  
 def make\_candidate\_pairs(self, b, r):   
 self.candidate\_pairs = set()  
 self.taken\_candidate\_pairs = np.zeros(shape=(len(self.documents), len(self.documents)))  
 cur\_band\_start, cur\_band\_end = 0, r - 1  
 for \_ in range(b):  
 band\_signature = dict()   
 for doc\_id in range(len(self.documents)):   
 matrix = tuple(self.signature\_matrix[cur\_band\_start : cur\_band\_end + 1, doc\_id])  
 if band\_signature.get(matrix) == None:   
 band\_signature[matrix] = [doc\_id]  
 else:   
 band\_signature[matrix].append(doc\_id)  
 for docs in band\_signature.items():   
 if len(docs[1]) > 1:   
 c\_pairs = list(combinations(docs[1], 2))  
 for (a, b) in c\_pairs:   
 self.candidate\_pairs.add((min(a, b), max(a, b)))  
 self.taken\_candidate\_pairs[a][b] = self.taken\_candidate\_pairs[b][a] = 1  
 cur\_band\_end += r  
 cur\_band\_start += r  
  
 def jaccard\_similarity(self, a, b) :  
 bits = np.bitwise\_and(a, b)  
 dif = np.bitwise\_xor(a, b)  
 return float(np.count\_nonzero(bits == 1) / (np.count\_nonzero(dif == 1) + np.count\_nonzero(bits == 1)))   
   
 def get\_similar\_documents(self, threshold):   
 similar\_docs = list()  
 self.similarity\_matrix = np.zeros(shape=(len(self.documents), len(self.documents)))  
 sims = list()  
  
 matrixes = [None for \_ in range(len(self.documents))]  
 for i in range(len(self.documents)):  
 matrixes[i] = self.incident\_matrix[:, i]  
 matrixes[i] = matrixes[i].astype(int)  
  
 for doc1 in range(len(self.documents)):   
 matrix1 = matrixes[doc1]  
  
 for doc2 in range(doc1 + 1, len(self.documents)):   
 matrix2 = matrixes[doc2]  
   
 similarity = self.jaccard\_similarity(matrix1, matrix2)  
 sims.append(similarity)  
 self.similarity\_matrix[doc1][doc2] = self.similarity\_matrix[doc2][doc1] = similarity  
 if similarity >= threshold:   
 similar\_docs.append((doc1, doc2, similarity))  
 return list(set(similar\_docs)), sims  
   
 def process\_candidate\_pairs(self, threshold):  
 false\_positive = false\_negative = 0  
 true\_positive = true\_negative = 0  
  
 for doc1 in range(len(self.documents)):  
 for doc2 in range(doc1 + 1, len(self.documents)):  
 if self.taken\_candidate\_pairs[doc1][doc2] == 1:  
 if self.similarity\_matrix[doc1][doc2] >= threshold:  
 true\_positive += 1  
 else:  
 false\_positive += 1  
 elif self.similarity\_matrix[doc1][doc2] >= threshold:  
 false\_negative += 1  
 else:  
 true\_negative += 1  
  
 print(f'''  
 true positive count = {true\_positive}  
 true negative count = {true\_negative}  
 false positive count = {false\_positive}  
 false negative count = {false\_negative}  
 ''')  
 return true\_positive, true\_negative, false\_positive, false\_negative   
  
 def unpickle(self):  
 my\_path = root / "Pickled\_files" / "Incident\_Matrix"  
 dbfile = open(my\_path, 'rb')   
 self.incident\_matrix = pickle.load(dbfile)  
 dbfile.close()  
  
 my\_path = root / "Pickled\_files" / "Shingle\_id"  
 dbfile = open(my\_path, 'rb')  
 self.shingle\_id = pickle.load(dbfile)  
 dbfile.close()  
  
 my\_path = root / "Pickled\_files" / "Shingles"  
 dbfile = open(my\_path, 'rb')  
 self.shingles = pickle.load(dbfile)  
 dbfile.close()  
  
 my\_path = root / "Pickled\_files" / "documents"  
 dbfile = open(my\_path, 'rb')  
 self.documents = pickle.load(dbfile)  
 dbfile.close()

def print\_docs(similar\_docs):  
 for x in similar\_docs:  
 print('##############################################################')  
 print(f'Doc1 is {article\_1.title[x[0]]}')  
 print(f'Doc2 is {article\_1.title[x[1]]}')  
 print(f'Similarity is {x[2] \* 100}%')

### r = 5, b = 10, hash count = 50

lsh = LSH(b=10, r=5, hash\_cnt=50)  
similar\_docs, sims = lsh.get\_similar\_documents(0.6)  
similar\_docs.sort(key=lambda x: -x[2])  
print\_docs(similar\_docs[:3])  
true\_positive, true\_negative, false\_positive, false\_negative = lsh.process\_candidate\_pairs(0.6)  
lsh\_stats['b'].append(10)  
lsh\_stats['r'].append(5)  
lsh\_stats['Number of hash functions'].append(50)  
lsh\_stats['True Positive'].append(true\_positive)  
lsh\_stats['True Negative'].append(true\_negative)  
lsh\_stats['False Positive'].append(false\_positive)  
lsh\_stats['False Negative'].append(false\_negative)  
lsh\_stats['Candidate Pair count'].append(len(lsh.candidate\_pairs))

##############################################################  
Doc1 is Transcript: President Obama on What Books Mean to Him - The New York Times  
Doc2 is Obama’s Secret to Surviving the White House Years: Books - The New York Times  
Similarity is 65.41832669322709%  
##############################################################  
Doc1 is Cyberwar for Sale - The New York Times  
Doc2 is The Perfect Weapon: How Russian Cyberpower Invaded the U.S. - The New York Times  
Similarity is 65.01605995717344%  
##############################################################  
Doc1 is President Obama’s Farewell Address: Full Video and Text - The New York Times  
Doc2 is Jolted by Deaths, Obama Found His Voice on Race - The New York Times  
Similarity is 63.88246946186861%  
  
 true positive count = 14  
 true negative count = 107149  
 false positive count = 17582  
 false negative count = 5

### r = 5, b = 20, hash count = 100

lsh = LSH(b=20, r=5, hash\_cnt=100)  
similar\_docs, sims = lsh.get\_similar\_documents(0.6)  
similar\_docs.sort(key=lambda x: -x[2])  
print\_docs(similar\_docs[:3])  
true\_positive, true\_negative, false\_positive, false\_negative = lsh.process\_candidate\_pairs(0.6)  
lsh\_stats['b'].append(20)  
lsh\_stats['r'].append(5)  
lsh\_stats['Number of hash functions'].append(100)  
lsh\_stats['True Positive'].append(true\_positive)  
lsh\_stats['True Negative'].append(true\_negative)  
lsh\_stats['False Positive'].append(false\_positive)  
lsh\_stats['False Negative'].append(false\_negative)  
lsh\_stats['Candidate Pair count'].append(len(lsh.candidate\_pairs))

##############################################################  
Doc1 is Transcript: President Obama on What Books Mean to Him - The New York Times  
Doc2 is Obama’s Secret to Surviving the White House Years: Books - The New York Times  
Similarity is 65.41832669322709%  
##############################################################  
Doc1 is Cyberwar for Sale - The New York Times  
Doc2 is The Perfect Weapon: How Russian Cyberpower Invaded the U.S. - The New York Times  
Similarity is 65.01605995717344%  
##############################################################  
Doc1 is President Obama’s Farewell Address: Full Video and Text - The New York Times  
Doc2 is Jolted by Deaths, Obama Found His Voice on Race - The New York Times  
Similarity is 63.88246946186861%  
  
 true positive count = 18  
 true negative count = 82662  
 false positive count = 42069  
 false negative count = 1

### r = 5, b = 40, hash count = 200

lsh = LSH(b=40, r=5, hash\_cnt=200)  
similar\_docs, sims = lsh.get\_similar\_documents(0.6)  
similar\_docs.sort(key=lambda x: -x[2])  
print\_docs(similar\_docs[:3])  
true\_positive, true\_negative, false\_positive, false\_negative = lsh.process\_candidate\_pairs(0.6)  
lsh\_stats['b'].append(40)  
lsh\_stats['r'].append(5)  
lsh\_stats['Number of hash functions'].append(200)  
lsh\_stats['True Positive'].append(true\_positive)  
lsh\_stats['True Negative'].append(true\_negative)  
lsh\_stats['False Positive'].append(false\_positive)  
lsh\_stats['False Negative'].append(false\_negative)  
lsh\_stats['Candidate Pair count'].append(len(lsh.candidate\_pairs))

##############################################################  
Doc1 is Transcript: President Obama on What Books Mean to Him - The New York Times  
Doc2 is Obama’s Secret to Surviving the White House Years: Books - The New York Times  
Similarity is 65.41832669322709%  
##############################################################  
Doc1 is Cyberwar for Sale - The New York Times  
Doc2 is The Perfect Weapon: How Russian Cyberpower Invaded the U.S. - The New York Times  
Similarity is 65.01605995717344%  
##############################################################  
Doc1 is President Obama’s Farewell Address: Full Video and Text - The New York Times  
Doc2 is Jolted by Deaths, Obama Found His Voice on Race - The New York Times  
Similarity is 63.88246946186861%  
  
 true positive count = 18  
 true negative count = 58597  
 false positive count = 66134  
 false negative count = 1

### r = 10, b = 20, hash count = 200

lsh = LSH(b=20, r=10, hash\_cnt=200)  
similar\_docs, sims = lsh.get\_similar\_documents(0.6)  
similar\_docs.sort(key=lambda x: -x[2])  
print\_docs(similar\_docs[:3])  
true\_positive, true\_negative, false\_positive, false\_negative = lsh.process\_candidate\_pairs(0.6)  
lsh\_stats['b'].append(20)  
lsh\_stats['r'].append(10)  
lsh\_stats['Number of hash functions'].append(200)  
lsh\_stats['True Positive'].append(true\_positive)  
lsh\_stats['True Negative'].append(true\_negative)  
lsh\_stats['False Positive'].append(false\_positive)  
lsh\_stats['False Negative'].append(false\_negative)  
lsh\_stats['Candidate Pair count'].append(len(lsh.candidate\_pairs))

##############################################################  
Doc1 is Transcript: President Obama on What Books Mean to Him - The New York Times  
Doc2 is Obama’s Secret to Surviving the White House Years: Books - The New York Times  
Similarity is 65.41832669322709%  
##############################################################  
Doc1 is Cyberwar for Sale - The New York Times  
Doc2 is The Perfect Weapon: How Russian Cyberpower Invaded the U.S. - The New York Times  
Similarity is 65.01605995717344%  
##############################################################  
Doc1 is President Obama’s Farewell Address: Full Video and Text - The New York Times  
Doc2 is Jolted by Deaths, Obama Found His Voice on Race - The New York Times  
Similarity is 63.88246946186861%  
  
 true positive count = 1  
 true negative count = 124157  
 false positive count = 574  
 false negative count = 18

### Tabulated Summary of the results

pd.DataFrame(lsh\_stats)

b r Number of hash functions True Positive True Negative \  
0 10 5 50 14 107149   
1 20 5 100 18 82662   
2 40 5 200 18 58597   
3 20 10 200 1 124157   
  
 False Positive False Negative Candidate Pair count   
0 17582 5 17596   
1 42069 1 42087   
2 66134 1 66152   
3 574 18 575

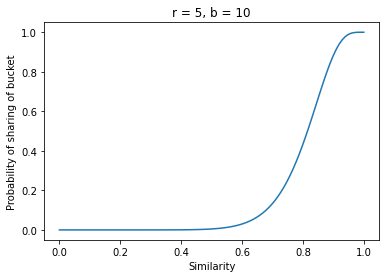
### Analysis of probability of sharing a bucket in LSH vs similarity for various given band sizes

import numpy as np  
from matplotlib import pyplot as plt

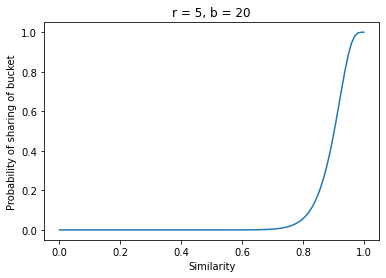
def similarity(b, r, s):  
 return 1 - (1 - s \*\* r) \*\* b

x = np.linspace(0, 1, 100)  
y1 = similarity(5, 10, x)  
y2 = similarity(5, 20, x)  
y3 = similarity(5, 40, x)

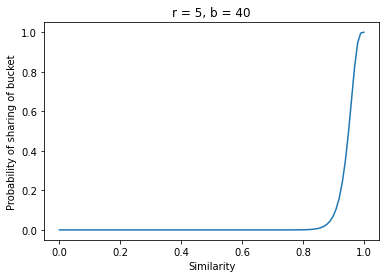
plt.plot(x, y1)  
plt.ylabel('Probability of sharing of bucket')  
plt.xlabel('Similarity')  
plt.title('r = 5, b = 10')  
plt.show()



plt.plot(x, y2)  
plt.ylabel('Probability of sharing of bucket')  
plt.xlabel('Similarity')  
plt.title('r = 5, b = 20')  
plt.show()



plt.plot(x, y3)  
plt.ylabel('Probability of sharing of bucket')  
plt.xlabel('Similarity')  
plt.title('r = 5, b = 40')  
plt.show()



### Group Members :

1. Milind Jain – 2020A7PS0153H
2. M Bhargav – 2020A7PS0025H
3. Ashwin Naveen Pugalia – 2020A7PS1080H
4. Shubh Badjate – 2020A7PS0028H