LAB.1

Homework 1: Finding Similar Items: Textually Similar Documents

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')

import os
import sys
sys.path.append('/content/drive/MyDrive/DM_Labs/dataset')

import pandas as pd
import numpy as np
```
```
Mounted at /content/drive
```

The aim of the project was to find textually similar documents based on Jaccard similarity using the shingling, minhashing, and locality-sensitive hashing (LSH) techniques. We have taken the MeDAL dataset (Medical Dataset for Abbreviation Disambiguation for Natural Language Understanding), that is a large medical text dataset curated for abbreviation disambiguation.

In [2]:
```
dataset = pd.read_csv('/content/drive/MyDrive/DM_Labs/dataset/data.csv', sep='\t', on_bad_lines='skip')
```

Visualizing loaded rows out of 1M rows:

In [4]:
```
dataset
```

Out[4]:

	Unnamed: 0	TEXT	LOCATION	LABEL
0	0	alphabisabolol has a primary antipeptic action...	56	substrate
1	1	a report is given on the recent discovery of o...	24|49|68|113|137|172	carcinosarcoma|recovery|reference|recovery|aft...
2	2	the virostatic compound nndiethyloxotetradecyl...	55	substrate
3	3	rmi rmi and rmi are newly synthetized nrdibenz...	25|82|127|182|222	compounds|compounds|inhibitory|lethal doses|ca...
4	4	a doubleblind study with intraindividual compa...	22|26|28|77|90|144|158|203	oxazepam|placebo|oral administration|pentagast...
...	...	...	...	...
999995	999995	the human gene for producing alcohol dehydroge...	24|59	liver biopsy|complete
999996	999996	measurement of CS in saliva has excited intere...	2|27|64|74|76|83|118	steroids|active|serum|sensitive|specific|assay...
999997	999997	a timeresolved fluoroimmunoassay trfia for unc...	14|15|37|60|173	serum|albumin|antiserum|after|vitro fertilization
999998	999998	porphyria cutanea tarda pct results from a met...	59|64|205	groups|study|degradation
999999	999999	recent advances in methodology allow the mass ...	62	electrophoresis

1000000 rows × 4 columns

We take only a slice of the dataset and drop all the columns that are different from **TEXT**.

In [5]:
```
small_dataset = dataset[:1000]
small_dataset = small_dataset['TEXT']
```

In [6]:
```
small_dataset[0]
```

Out[6]:
```
'alphabisabolol has a primary antipeptic action depending on dosage which is not caused by an alteration of the phval
ue the proteolytic activity of pepsin is reduced by percent through addition of bisabolol in the ratio of the antipep
tic action of bisabolol only occurs in case of direct contact in case of a previous contact with the ATP the inhibiti
ng effect is lost'
```

Shingling

Let's define a function that constructs k–shingles of a given length (k=5) from a given document, computes a hash value for each unique shingle and represents the document in the form of an ordered set of its hashed k-shingles. Defining a shingling function that uses **list comprehension**, defining a loop while iterating the list to make it faster.

In [7]:
```
def shingles(text, size):
    chars = list(text) # Splitting the text
    return [''.join(chars[i:i+size]) for i in range(len(chars) - size + 1)]
# Joining with an empty char from the i-th char to the 'i-th + size-1' char, 'len(chars) - size + 1' times
```

k = 5 for small documents.

In [8]:
```
k = 5
five_shingles = shingles(small_dataset[0], k)
five_shingles[:10] # Just to show the first 10 5-shingles of the first document
```

Out[8]:
```
['alpha',
 'lphab',
 'phabi',
 'habis',
 'abisa',
 'bisab',
 'isabo',
 'sabol',
 'abolo',
 'bolol']
```

crc32 will be used to hash the shingles.

In [9]:
```
from binascii import crc32
import time
```

In [10]:
```
def shinglify(dataset, k):

    t0 = time.time()
    docs_as_hashed_shingles = {}

    # For each document
    for i in range(len(dataset)):
        hashed_shingles = set()
        text = dataset[i]

        # For each shingle
        for shingle in set(shingles(text, k)): # Directly converts to a shingle set to avoid repetitions
            crc_hash = crc32(shingle.encode('utf-8')) & 0xffffffff
            hashed_shingles.add(crc_hash)

        docs_as_hashed_shingles[i] = hashed_shingles

    t1 = time.time()
    return docs_as_hashed_shingles, t0, t1
```

In [11]:
```
shingled_dataset, t0, t1 = shinglify(small_dataset, k) # Returns times as well
print(str(k) + '-Shingling ' + str(len(small_dataset)) + ' docs took: %.2f seconds' %(t1 - t0))
```
```
5-Shingling 1000 docs took: 0.75 seconds
```

Compare Sets

Let's define a method which computes the **Jaccard similarity** of two sets of integers: two sets of hashed shingles.

In [129]:
```
def jaccard_2docs(doc1: set, doc2: set):
  return len(doc1 & doc2)/len(doc1 | doc2)
```

In [130]:
```
def similarity_matrix(shingled_dataset):

  t0 = time.time()
  sim_matrix = np.zeros((len(small_dataset),len(small_dataset)))

  for row in range(len(small_dataset)):
    for col in range(row, len(small_dataset)):
      if(row == col): # Skipping the similarity for the same document as it would be 1
        continue
      sim_matrix[row][col] = jaccard_2docs(shingled_dataset[row], shingled_dataset[col])

  t1 = time.time()
  return sim_matrix, t0, t1
```

In [132]:
```
sim_matrix, t0, t1 = similarity_matrix(shingled_dataset)
shingling_sim_time = t1 - t0
print('double-loop Similarity Matrix computation for ' + str(len(small_dataset)) + ' docs took: %.2f seconds' %shing
ling_sim_time)
```
```
double-loop Similarity Matrix computation for 1000 docs took: 50.52 seconds
```

Saved the time for the first technique.

In [133]:
```
sim_matrix[0][:10]
```

Out[133]:
```
array([0.        , 0.05121639, 0.06199461, 0.05589226, 0.04317656,
       0.07427938, 0.05474453, 0.05051546, 0.05222734, 0.05682782])
```

Let's print the maximum of the similarity matrix and finding which are the documents with the maximum value.

In [134]:
```
print(np.max(sim_matrix))
print(np.unravel_index(np.argmax(sim_matrix, axis=None), sim_matrix.shape))
```
```
0.4558180227471566
(173, 175)
```

As we can see, the following texts have some parts in common.

In [135]:
```
small_dataset[173]
```

Out[135]:
```
'reduced coenzyme qcytochrome c reductase from bovine heart mitochondria complex iii was incorporated into phospholip
id LDV by the cholate dialysis procedure soybean phospholipids or mixtures of purified phosphatidylcholine phosphatid
ylethanolamine and cardiolipin could be used oxidation of reduced coenzyme q by the reconstituted vesicles with cytoc
hrome c as oxidant showed the following energycoupling phenomena protons were translocated outward with a coupling ra
tio he of measurements with mitochondria under similar conditions showed an he ratio of proton translocation was not
seen in the presence of uncoupling agents and was in addition to the net acidification of the medium from the overall
oxidation reaction potassium ions were taken up by the reconstituted vesicles in the presence of valinomycin in a rea
ction coupled to electron transfer the coupling ratio for k uptake ke was in the vesicles and approximately in mitoch
ondria the rate of oxidation of reduced coenzyme q by the reconstituted LDV was stimulated up to fold by uncouplers o
r by valinomycin plus nigericin and k ions addition of valinomycin CT in a k medium caused a transient stimulation of
electron transfer the results indicate that SE coupling can be observed with isolated reduced coenzyme qcytochrome c
reductase if the enzyme complex is properly incorporated into a phospholipid vesicle'
```

In [136]:
```
small_dataset[175]
```

Out[136]:
```
'nadhcoenzyme q reductase from bovine heart mitochondria complex i was incorporated into phospholipid LDV by the chol
ate dialysis procedure mixtures of purified phosphatidylcholine and phosphatidylethanolamine were required oxidation
of nadh by coenzyme q catalyzed by the reconstituted vesicles was coupled to proton translocation directed inward wit
h an he ratio greater than similar experiments measuring proton translocation in submitochondrial particles gave an h
e ratio of the proton translocation in both systems was not seen in the presence of uncoupling agents and was in addi
tion to the net proton uptake from the reduction of coenzyme q by nadh electron transfer in the reconstituted LDV als
o caused the uptake of the permeant anion tetraphenylboron the rate of electron transfer by the reconstituted vesicle
s was stimulated about fold by uncouplers or by valinomycin plus nigericin and k ions the results indicate that energ
y coupling can be observed with isolated nadhcoenzyme q reductase if the enzyme complex is properly incorporated into
a phospholipid vesicle'
```

Min Hashing

Let's define a function which builds a minHash signature (in the form of a vector or a set) of a given length n from a given set of integers (a set of hashed shingles).

First of all, we take *numHashes* independent hash functions.

In [90]:
```
# number of random hash functions
num_hashes = 100
```

After that, we find the *maxShingleID*, that will be used lately for the hash function.

In [92]:
```
t0 = time.time()

maxShingleID = 0

for doc_id in shingled_dataset:

  shingle_id_set = shingled_dataset[doc_id]
  signature = []

  for i in range(0, num_hashes):

    for shingle_id in shingle_id_set:

      if(maxShingleID < shingle_id):
        maxShingleID = shingle_id

t1 = time.time()
max_shingle_time = t1 - t0
```

In [93]:
```
print(maxShingleID)

# We need the next largest prime number after 'maxShingleID'.
# http://compoasso.free.fr/primelistweb/page/prime/liste_online_en.php
nextPrime = 4294895617
```
```
4294895493
```

In [94]:
```
import random
random.seed(42)

def random_coeffs(k):
  # k random vaues
  rand_list = []

  for i in range(k):
    # random shingle ID
    rand_idx = random.randint(0, maxShingleID)

    # uniqueness check
    while rand_idx in rand_list:
      rand_idx = random.randint(0, maxShingleID)

    rand_list.append(rand_idx)

  return rand_list
```

For each hash function, generate a different coefficient **a** and **b**.

In [95]:
```
coeff_A = random_coeffs(num_hashes)
coeff_B = random_coeffs(num_hashes)
```

Checking that they're not representing the same line (like 3x+12 and x+4).

In [96]:
```
for a, b in zip(coeff_A, coeff_B):
  assert a not in coeff_B
  assert b not in coeff_A

  for x, y in zip(coeff_A, coeff_B):
    if(a != x and b != y):
      assert a/b != x/y
```

For each of the shingles of a document, calculate its hash using i_{th} hash function.

In [97]:
```
t0 = time.time()
# docs as signatures
signatures = []

for doc_id in shingled_dataset:

  shingle_id_set = shingled_dataset[doc_id]

  signature = [] # result signature

  for i in range(num_hashes):
    min_hash = nextPrime + 1 # initialize min_hash to be greater than the nextPrime

    for shingle_id in shingle_id_set:

      hash_code = (coeff_A[i] * shingle_id + coeff_B[i]) % nextPrime # (ax + b) % c

      if hash_code < min_hash:
        min_hash = hash_code

    signature.append(min_hash) # keep the min_hash

  signatures.append(signature) # keep the entire signature of document doc_id

t1 = time.time()
min_hashing_time = t1 - t0
min_hashing_time
```

Out[97]:
```
32.12442111968994
```

In [98]:
```
num_docs = len(shingled_dataset) # 1000
```

Time to compare the signatures (task 4):

In [111]:
```
def compare_signatures():
  t0 = time.time()
  matrix = np.zeros((len(shingled_dataset), len(shingled_dataset)))

  for i in range(num_docs):

    doc1_signature = signatures[i]

    # For each of the other test documents...
    for j in range(i, num_docs):

      if(i == j):
        continue
      doc2_signature = signatures[j]

      count = 0
      for k in range(num_hashes):
        agree = 0
        if(doc1_signature[k] == doc2_signature[k]):
          agree = 1
        count = count + agree  # on how many do they agree?

      matrix[i][j] = float(count) / float(num_hashes) # assigning similarity
  t1 = time.time()
  return matrix, (t1 - t0)
```

The percentuage matrix will have on the rows and on the columns all the documents.

In [112]:
```
minHash_sim_matrix, compare_time = compare_signatures()
```

In [113]:
```
minHash_tot_time = max_shingle_time + min_hashing_time + compare_time
minHash_tot_time
```

Out[113]:
```
46.102508544921875
```

In [104]:
```
threshold = 0.3
```

In [114]:
```
for i in range(num_docs):
  for j in range(i, num_docs):

    if(i == j):
      continue

    if minHash_sim_matrix[i][j] > threshold:
      print (minHash_sim_matrix[i][j])
```
```
0.4
0.32
0.44
0.35
0.31
0.32
```

The documents number 408 and 410 have the highest percentuge of same hash values.

In [115]:
```
print(np.max(minHash_sim_matrix))
print(np.unravel_index(np.argmax(minHash_sim_matrix, axis=None), minHash_sim_matrix.shape))
```
```
0.44
(408, 410)
```

Let's check values of before:

In [116]:
```
print(minHash_sim_matrix[173, 175]) # still high similarity on these two
```
```
0.4
```

In [117]:
```
small_dataset[408]
```

Out[117]:
```
'the nadpspecific glu dehydrogenase of neurospora crassa was digested with trypsin and peptides accounting for out of
the residues of the polypeptide chain were isolated and substantially sequenced additional experimental detail has be
en deposited as supplementary publication sup pages with the british library lending division boston spa wetherby w y
orkshire ls bq uk from whom copies may be obtained under the terms given in biochem j'
```

In [118]:
```
small_dataset[410]
```

Out[118]:
```
'peptic and chymotryptic peptides were isolated form the nadpspecific glutamate dehydrogenase of neurospora crassa an
d substantially sequenced out of residues in the polypeptide chain were recovered in the peptic and in the chymotrypt
ic peptides together with the tryptic peptides wootton j c taylor j g jackson a a chambers g k fincham j r s biochem
j these establish the CR CS of the chain including the acid and NH assignments except for seven places where overlaps
are inadequate these remaining alignments are deduced from information on the cnbr fragments obtained in another labo
ratory blumenthal k m moon k smith e l j biol chem further information has been deposited as supplementary publicatio
n sup pages with the british library lending division boston spa wetherby w yorkshire ls bq uk from whom copies may b
e obtained under the terms given in biochem j'
```

Same words: boston, biochem, j, ls, bq, uk, dehydrogenase, neurospora, polypeptide, etc...

LSH

A class LSH that implements the LSH technique: given a collection of minhash signatures (integer vectors) and a similarity threshold t, the LSH class (using **banding** and hashing) finds candidate pairs of signatures agreeing on at least a fraction t of their components.

$\begin{cases}(1/b)^{1/r} \sim s \\ b\,r = n\end{cases}$ remember from the slides.

In [205]:
```
bands = 20 # between 1 and the number of hashes 100
rows_per_band = num_hashes / bands
lsh_threshold = 1/bands ** (1/float(rows_per_band))
lsh_threshold
```

Out[205]:
```
0.5492802716530589
```

Defining a function that given a signature and number of bands, returns a band_hash_list containing the cycled mod_bands buuckets of the hashes of the signature.

In [259]:
```
'''
def band_hash(bands, minHash_signature):
  band_hash_list = []
  band_hash = 0
  for i in range(len(minHash_signature)): # for every hash in the signature

        if i % bands == 0: # number of bands modular cycle (#bands different classes)

            if i > 0:
              band_hash_list.append(band_hash)
            band_hash = 0

        band_hash = hash(minHash_signature[i]) # hashing i-th value of the signature

  return band_hash_list
  '''
```

Defining the LSH function.

In [178]:
```
import itertools
import time
```

In [260]:
```
'''
def LSH(signatures, threshold, bands, num_hashes):
  t0 = time.time()

  final_result = {} # dict: (pair of similar documents) -> similarity (> threshold)

  lsh_signatures = {}
  band_hashes = {}
  doc_id = 0

  for signature in signatures:

    lsh_signatures[doc_id] = signature
    minhash_sig = signature
    new_hashes = band_hash(bands, minhash_sig)

    # Time to construct the dictionary that links signatures and buckets

    for i in range(len(new_hashes)): # for every element of the new hashes

      if i not in band_hashes: # if not already added
        band_hashes[i] = {} # create instance in the dictionary

      if new_hashes[i] not in band_hashes[i]:
        band_hashes[i][new_hashes[i]] = [] # insert the first element
        band_hashes[i][new_hashes[i]].append(doc_id)
      else:
        band_hashes[i][new_hashes[i]].append(doc_id)

    doc_id = doc_id + 1

  found_similarities = set()
  # Compare each bucket signatures
  for h in band_hashes:
    for bucket in band_hashes[h]:

      if len(band_hashes[h][bucket]) > 1: # if at least two elements...
        for two_docs in itertools.combinations(band_hashes[h][bucket], r=2):
          if two_docs not in found_similarities:
            found_similarities.add(two_docs)

            sim = jaccard_2docs(set(lsh_signatures[two_docs[0]]), set(lsh_signatures[two_docs[1]]))
            if(float(sim) > lsh_threshold):
              print(sim)
              final_result[two_docs] = sim

  t1 = time.time()
  return final_result, (t1 - t0)
'''
```

Couldn't get the last part to work, some fixes are needed.

In [ ]:
```
!jupyter
```