Yingxuan WU B00802170

Zheng WAN B00758336

Problem 1, 30 points: (On the Jaccard distance)

Question 1

Given the definition of the events (A), (B), and (C):

$$A = \{h(x) \neq h(y)\}$$
$$B = \{h(y) \neq h(z)\}$$
$$C = \{h(x) \neq h(z)\}$$

$$\begin{aligned} p_1 &= \mathbb{P}(\overline{A} \cap \overline{B} \cap C) \\ &= \mathbb{P}(\mathrm{h(x) = h(y), h(y) = h(z), but}\ h(x) \neq h(z)) \\ &= 0 \\ p_2 &= \mathbb{P}(\overline{A} \cap B \cap \overline{C}) \\ &= \mathbb{P}(\mathrm{h(x) = h(y), h(x) = h(z), but}\ h(y) \neq h(z)) \\ &= 0 \\ p_4 &= \mathbb{P}(A \cap \overline{B} \cap \overline{C}) \\ &= \mathbb{P}(\mathrm{h(y) = h(z), h(x) = h(z), but}\ h(x) \neq h(y)) \\ &= 0 \end{aligned}$$

Thus, based on the definitions of the events and their probabilistic interpretations, we can conclude that $p_1 = p_2 = p_4 = 0$.

Question 2

we know that from p_0 to p_7, only p_1, p_2, and p_4 are invalid.

So we have:

$$\mathbb{P}(A) = p_5 + p_6 + p_7$$
$$\mathbb{P}(B) = p_3 + p_6 + p_7$$
$$\mathbb{P}(C) = p_3 + p_5 + p_7$$

Question 3

To meet the necessary condition, we have:

$$\mathbb{P}(A) + \mathbb{P}(B) = p_3 + p_5 + 2p_6 + 2p_7$$

we know for sure that p_6 and p_7 are strictly positive, so the formular of $\mathbb{P}(A) + \mathbb{P}(B) \geq \mathbb{P}(C)$ is satisfied.

The satisfaction of the triangle inequality shows that the function d(x,y) defined as 1−sim(x,y) obeys the necessary properties and supports the idea that the Jaccard similarity possesses a locality-sensitive hashing scheme.

Problem 2

Question 1

Linearity of Expectation:

First, let's compute the expected number of false positives for a given hash function g_i

$$E[|T \cap W_i|] = \sum_{z_i \in A} P(z_i \text{ is a false positive under } g_i)$$

where

$$P(z_i \text{ is a false positive under } g_i) = P(g(z) = g(z_i) \text{ and } d(z, z_i) > c\lambda).$$

These events are not independent due to the properties of LSH functions. In LSH, the probability of collision is higher for closer points.

Thus, the probability that two distant points collide (false positive) is typically much lower than the probability that two close points collide.

Use of Markov's Inequality:

To show that the sum over all hash functions is less than $3L$ with probability at least $\frac{2}{3}$,we can use Markov's inequality:

$$P(X \geq a) \leq \frac{E[X]}{a}$$

$$X = \sum_{i=1}^{L} |T \cap W_i| \text{ and } a = 3L.$$

Assuming we find that:

$$E[X] = kL \text{ where } k < 3$$

$$P\left(\sum_{i=1}^{L} |T \cap W_i| \geq 3L\right) \leq \frac{kL}{3L} = \frac{k}{3}$$

For the probability to be less than $\frac{1}{3}$,k must be less than 1.

Given that:

$$P(g(x) = g(z)) = \frac{1}{n} \text{ for any } x \text{ such that } d(x, z) > c\lambda$$

We could easily deduce that k is less than 1 when n is bigger than 1.

Thus，the bound is improved.

Question 2

$$P(g(x) = g(z)) = \frac{1}{n^\rho} \text{ for any } x \text{ such that } d(x, z) \leq \lambda \text{ for some } \rho < 1$$

$$\text{the probability that } g_j(x^*) = g_j(z) \text{ for any particular } j \text{ is at least } \frac{1}{n^\rho}$$

$$\text{the probability that } g_j(x^*) \neq g_j(z) \text{ for any particular } j \text{ is at most } 1 - \frac{1}{n^\rho}$$

$$\text{the probability that } g_j(x^*) \neq g_j(z) \text{ for all } j \text{ is at most } (1 - \frac{1}{n^\rho})^L$$

Using $L = n^\rho$

$$P[g_j(x^*) \neq g_j(z) \, \forall \, 1 \leq j \leq L] \leq (1 - \frac{1}{L})^L < \frac{1}{e}$$

Using the limit-based definition of the exponential function, specifically:

$$e^x = \lim_{m \to \infty} (1 + \frac{x}{m})^m$$

When m is a finite value greater than or equal to 1, this expression serves as a lower boundary for the series' convergence.

Question 3

Given a set A and a distance function d, the algorithm aims to identify an element x^* such that its distance to some other point z is at most λ.

Two main ways the algorithm can fail:

1. **False Negatives**: x^* is hashed to an incorrect bucket. The probability of this event happening is bounded at e^{-1}.

$$P_{\text{false negative}} \leq e^{-1}$$

1. **False Positives**: x^* is hashed to the correct bucket but is overshadowed by too many other points, specifically more than $3L$.

The combined probability of failure, taking into account both scenarios, is:

$$P_{\text{fail}} \leq e^{-1} + \left(1 - e^{-1}\right) \times \frac{1}{3}$$

Given that $P_{\text{fail}} \approx 0.58$, there's a decent chance that the algorithm might not always return the correct point. But when it succeeds, it ensures that the returned point satisfies the (c, λ)-ANN condition.

This hashing-based approach, while probabilistic, offers a mechanism to quickly retrieve approximate nearest neighbors under certain constraints.

Problem 3, 40 points: (A similarity-matching function)

In [1]:

```
#1. Split text into elements: write a function shingles to convert the input
#text into elements of three characters
def shingles(line):
  res = set()
  for i in range(len(line) - 3 +1):
    res.add(line[i:i+3].lower())
  return res
```

In [2]:

```
from traitlets import Union
#2. Calculate Jaccard distance: create a function that accepts as imputs two
#sets and compute its Jaccard distance
```

```
def jdist(setA, setB):
  inter = len(setA.intersection(setB))
  union = len(setA.union(setB))
  if union == 0:
    return 0
  else:
    sim = inter/union
    return sim
```

In [3]:

```
#3. Test out the results by running the following code:
print(shingles("This function works perfectly"))
```

```
{'is ', ' pe', 'wor', 'rfe', 'ork', 'ect', 'nct', 'per', ' fu', ' wo', 'ks ', 'thi', 's
f', 'erf', 'tio', 's p', 'his', 'rks', 'fun', 'on ', 'tly', 'ctl', 'cti', 'fec', 'ion',
'n w', 'unc'}
```

In [4]:

```
#4. Use 1 and 3 to write the minhash function. Put the main code inside a try
# except call.

#I have included the lower() in shingles, so I did not put it in this part
def minhash(input_question, compare_question):
    try:
        input_shingles = shingles(input_question)
        compare_shingles = shingles(compare_question)
        return jdist(input_shingles, compare_shingles)
    except Exception as e:
        print(f"An error occurred: {e}")
        return 0

print(minhash("I have a cat", "I have an apple"))
print(minhash("I have a cat", "I have a dog"))
print(minhash("I have a cat", "I have a caf"))
print(minhash("I have a cat", "I have a cat"))

#Observation: We noticed thatthe similarity increases as the two texts converge
#from each other.
#This is because the number of shared shingles (3-character sequences) between
#the texts increases, thus increasing the Jaccard similarity value.
```

```
0.35294117647058826
0.5384615384615384
0.8181818181818182
1.0
```

Problem 4, 70 points: (Searching via MinHash and Locality Sensitive Hashing)

In [5]:

```
! pip install datasketch
! pip install nltk
```

```
Requirement already satisfied: datasketch in /Users/zhengwan/opt/anaconda3/lib/python3.
8/site-packages (1.6.4)
Requirement already satisfied: numpy>=1.11 in /Users/zhengwan/opt/anaconda3/lib/python
3.8/site-packages (from datasketch) (1.24.3)
Requirement already satisfied: scipy>=1.0.0 in /Users/zhengwan/opt/anaconda3/lib/python
3.8/site-packages (from datasketch) (1.10.1)
Requirement already satisfied: nltk in /Users/zhengwan/opt/anaconda3/lib/python3.8/site
-packages (3.5)
Requirement already satisfied: tqdm in /Users/zhengwan/opt/anaconda3/lib/python3.8/site
-packages (from nltk) (4.65.0)
Requirement already satisfied: joblib in /Users/zhengwan/opt/anaconda3/lib/python3.8/si
te-packages (from nltk) (1.2.0)
Requirement already satisfied: click in /Users/zhengwan/opt/anaconda3/lib/python3.8/sit
e-packages (from nltk) (7.1.2)
Requirement already satisfied: regex in /Users/zhengwan/opt/anaconda3/lib/python3.8/sit
e-packages (from nltk) (2020.10.15)
```

In [6]:

```
 # 1. You may need the folowing packges:
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read csv)
from collections import defaultdict
from tqdm import tqdm # make your loops show a smart progress meter
import nltk # Natural Language Toolkit
import datasketch # Probabilistic data structures for processing and searching very la
```

In [7]:

```
# 2. Extract the data from “train.csv” to the DataFrame qa pairs and take a look on it.
qa_pairs = pd.read_csv("train.csv")
print(qa_pairs.head())
```

```
   id  qid1  qid2                                          question1  \
0   0     1     2  What is the step by step guide to invest in sh...
1   1     3     4  What is the story of Kohinoor (Koh-i-Noor) Dia...
2   2     5     6  How can I increase the speed of my internet co...
3   3     7     8  Why am I mentally very lonely? How can I solve...
4   4     9    10  Which one dissolve in water quikly sugar, salt...

                                           question2  is_duplicate
0  What is the step by step guide to invest in sh...             0
1  What would happen if the Indian government sto...             0
2  How can Internet speed be increased by hacking...             0
3  Find the remainder when [math]23^{24}[/math] i...             0
4            Which fish would survive in salt water?             0
```

In [8]:

```
#3. Create a random sample of questions from qa pairs. For example you can use
#the following code:
sents_pairs = pd.concat([qa_pairs[qa_pairs['is_duplicate'] == 0].sample(100, random_sta
sents_pairs = sents_pairs.sample(frac=1.)
sents = pd.concat([sents_pairs['question1'], sents_pairs['question2']])
```

In [9]:

```
#4. Represent the questions as single word tokens if they are not stop words:
#• Download ‘stopwords’ from nltk package
nltk.download('stopwords')
nltk.download('punkt')

from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
print(stop_words)

#• Create ‘set_dict’ dictionary which maps question id (eg ‘m23’) to set
#representation of question.
#• Loop through each question, convert them into shingles, and, if the shingle
#isn’t a stopword, add it to a hashset which will be the value for the set dict
#dictionary.
#• Do not froget to lowcase!
#• Additionally create ‘norm dict’ dictionnary which maps question id (eg ‘m23’)
# to actual question (we may use it to evaluate the result).

set_dict = defaultdict(set)
norm_dict = {}


'''
# we define a new shingles function here to remove the words in the stopwords:
def shingles2(line):
    res = set()
    for i in range(len(line) - 3 + 1):
        shingle = line[i:i+3].lower()
        if not any(word in stop_words for word in shingle.split()):
            res.add(shingle)
    return res

for index, row in sents_pairs.iterrows():
    question1 = row['question1'] if pd.notna(row['question1']) else ""
    question2 = row['question2'] if pd.notna(row['question2']) else ""
    q1_shingles = set(shingles2(question1))
    q2_shingles = set(shingles2(question2))
    mid1 = "m" + str(row['qid1'])
    mid2 = "m" + str(row['qid2'])
    set_dict[mid1] = {'question':q1_shingles}
    set_dict[mid2] = {'question':q2_shingles}
    norm_dict[mid1] = {'question':question1}
    norm_dict[mid2] = {'question':question2}
'''

for index, row in sents_pairs.iterrows():
    words_q1 = set([word.lower() for word in nltk.word_tokenize(row['question1']) if wo
    words_q2 = set([word.lower() for word in nltk.word_tokenize(row['question2']) if wo

    mid1 = "m" + str(row['qid1'])
    mid2 = "m" + str(row['qid2'])

    set_dict[mid1] = words_q1
    set_dict[mid2] = words_q2

    norm_dict[mid1] = row['question1']
    norm_dict[mid2] = row['question2']
```

```
{'the', 'not', 'having', 'this', "mightn't", 'from', 'all', 'theirs', 'any', 'that', "s
houldn't", 'be', 'over', 'both', 'which', 'too', 'wouldn', 'hers', 'we', 'through',
'm', 'did', "you're", 'and', 'there', 'will', 'so', 'll', "wouldn't", 'it', 'what', 'd
o', "weren't", 'further', "it's", 'mustn', 'yourselves', 'wasn', 'other', 'with', 'betw
een', 'no', "isn't", 'myself', 'weren', 'under', 'yourself', 'was', 'for', 'doing', "di
dn't", "haven't", 'same', 'are', 'had', 'can', 'in', 'who', 'but', 'd', 'ma', 'were',
"won't", 'above', 'each', "hadn't", "should've", "she's", 'during', 'o', "you'll", 'cou
ldn', 'hasn', 'ourselves', 'don', "aren't", 's', 'as', "you'd", 'once', 'my', "wasn't",
'few', 'more', 'these', 'ours', 'an', 'against', 'they', 'before', 'where', 'nor', 'i',
'why', "you've", 'being', 'just', 'whom', 'by', 'down', 'has', 'because', 'at', "must
n't", 'have', 'your', 'or', 'its', 'themselves', 'he', 'some', "don't", 't', 'should',
'isn', 'his', 'y', 'only', 'into', 've', 'here', 'yours', "doesn't", 'aren', 'than', 'a
fter', 'her', 're', 'a', 'me', 'up', 'didn', 'their', 'below', 'now', 'again', 'about',
"shan't", 'mightn', 'herself', 'if', "hasn't", 'ain', 'doesn', 'shouldn', 'himself', "c
ouldn't", 'itself', "needn't", 'until', 'does', 'haven', 'am', 'you', 'those', 'then',
"that'll", 'she', 'such', 'hadn', 'shan', 'of', 'is', 'most', 'them', 'won', 'how', 'wh
ile', 'off', 'our', 'to', 'own', 'very', 'him', 'out', 'on', 'needn', 'when', 'been'}
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/zhengwan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /Users/zhengwan/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [10]:

```
#5. Create minHash signatures:
#• Fix the number of permutations for the MinHash algorithm ‘num perm’.
from datasketch import MinHash, MinHashLSH
num_perm = 128
min_dict = defaultdict()

#• Create ‘min dict’ which maps question id (eg ‘m23’) to min hash signatures.
# You can use ‘MinHash’ from ‘datasketch’ package

#• Loop through all the set representations of questions and calculate the
# signatures and store them in the ‘min dict’ dictionary
for qid, question in set_dict.items():
    m = MinHash(num_perm=num_perm)
    for word in question:
        m.update(word.encode('utf8'))
    min_dict[qid] = m
```

In [11]:

```
#6. LSH can be used with MinHash to achieve sub-linear query cost. Create LSH
#index using
#‘MinHashLSH’ from ‘datasketch’ package
#• Set the Jaccard similarity threshold (e.g. =0.4) as a parameter in MinHashLSH.
#• Loop through the signatures or keys in the ‘min dict’ dictionary and store
#them. Datasketch stores these in a dictionary format, where the key is a
#question and the values are all the questions deemed similar based on the
```

```
#threshold.

# Create LSH index
lsh = MinHashLSH(threshold=0.7, num_perm=128)
for qid, question in min_dict.items():
    lsh.insert(qid, question)
```

In [12]:

```
#7. Giving the MinHash of the query set, retrieve the keys (m1, m2 etc.) that reference
#approximate Jaccard similarities using the following code:
big_list = []
for query in min_dict.keys():
    big_list.append(lsh.query(min_dict[query]))
```

In [13]:

```
# 8. Check some of the resulting pairs.
print(big_list[:12])

for similar_qids in big_list[:12]:
    if len(similar_qids) > 1:
        print("Similar Questions:")
        for qid in similar_qids:
            print(f"{qid}: {norm_dict[qid]}")
        print("\n")
```

```
[['m37699'], ['m37700'], ['m95929'], ['m95930'], ['m202142'], ['m128309'], ['m63529'],
['m63530'], ['m18328'], ['m46707'], ['m323845', 'm5107'], ['m323845', 'm5107']]
Similar Questions:
m323845: What should I do to enjoy my life?
m5107: How do I enjoy the life?


Similar Questions:
m323845: What should I do to enjoy my life?
m5107: How do I enjoy the life?
```