Homework 1

Mohammad Mahdi Rahimi (20208244)
mahi@kaist.ac.kr

March 17, 2021

Problem 1

Computing the Signatures

(a) Calculate the Cosine Similarity Signature

$$\begin{aligned} S1 :&[-1,-1,+1,-1] \\ S2 :&[+1,+1,-1,-1] \\ S3 :&[+1,-1,+1,-1] \\ S4 :&[-1,+1,+1,+1] \\ S5 :&[-1,+1,+1,-1] \end{aligned} \tag{1}$$

(b) Calculate the true and estimate cosine similarities of the 10 pairs

True, Estimated ⇒ True, Estimated, Diff:

$$\begin{aligned} Sim(S1,S2) :& \frac{1}{2}, -\frac{2}{4} \Rightarrow 60°, 120°, 60° \\ Sim(S1,S3) :& \frac{1}{2\sqrt{6}}, \frac{2}{4} \Rightarrow 78°, 60°, 18° \\ Sim(S1,S4) :& \frac{1}{2\sqrt{3}}, \frac{0}{4} \Rightarrow 73°, 90°, 17° \\ Sim(S1,S5) :& \frac{2}{3\sqrt{(2)}}, \frac{2}{4} \Rightarrow 62°, 60°, 2° \\ Sim(S2,S3) :& \frac{1}{2\sqrt{6}}, \frac{0}{4} \Rightarrow 78°, 90°, 12° \\ Sim(S2,S4) :& \frac{0}{2\sqrt{3}}, -\frac{2}{4} \Rightarrow 90°, 120°, 30° \\ Sim(S2,S5) :& \frac{0}{3\sqrt{2}}, \frac{0}{4} \Rightarrow 90°, 90°, 0°] \\ Sim(S3,S4) :& \frac{2}{6\sqrt{(2)}}, -\frac{2}{4} \Rightarrow 76°, 120°, 44° \\ Sim(S3,S5) :& \frac{4}{12\sqrt{(3)}}, \frac{0}{4} \Rightarrow 79°, 90°, 11° \\ Sim(S4,S5) :& \frac{6}{3\sqrt{6}}, \frac{2}{4} \Rightarrow 35°, 60°, 25° \end{aligned} \tag{2}$$

Considering Estimation can only generate $\{0, 60, 90, 120, 180\}$.

Problem 2

Locality Sensitive Hashing

(a) Detail of Implementation

Libraries.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from tqdm import tqdm

from collections import defaultdict
import itertools
import re
import time
import math
import os
import itertools
```

Shingling.

[2]:
```
def get_shingles(documents, k):
    shingles = set()
    for doc in documents:
      l = doc.split()
      [shingles.add(tuple(l[i:i+k])) for i in range(len(l) - k + 1)]

    return shingles
```

[3]:
```
def build_doc_to_shingle_dictionary(documents, shingles):
    doc_to_shingles = {}
    shingle2idx = {}
    dic = {}
    for i, doc in enumerate(documents):
      l = doc.split()
      dic[i] = [tuple(l[j:j+K]) for j in range(len(l) - K + 1)]

    shingle2idx = {}
    [shingle2idx.update({shi: i}) for i, shi in enumerate(shingles)]

    doc_to_shingles = {}
    for doc_id, shins in dic.items():
        cnt = {}
        [cnt.update({shingle2idx[shin]: 0}) for shin in shins]
        [cnt.update({shingle2idx[shin]: cnt[shingle2idx[shin]]+1}) for shin␣
 ↪in shins]

        doc_to_shingles[doc_id] = [[k, v] for k, v in cnt.items()]

    return doc_to_shingles
```

Cosine Similarity

```
[4]: def cosine_similarity(v1, v2):

         similarity = np.dot(v1, v2) / (np.linalg.norm(v1)*np.linalg.norm(v2))

         return similarity
```

Make Signatures with Random Hyper Planes

```
[5]: # random hyperplanes
     M = 256
     random_vectors = np.random.randn(M, len(shingles))
```

```
[6]: def make_signature(doc_to_shingles, random_vectors):
       C = len(doc_to_shingles)
       M, S = random_vectors.shape
       signatures = np.array(np.ones((M, C)) * np.inf, dtype = np.int)
       for doc, shin_cnt_list in doc_to_shingles.items():
         for i in range(M):
           signatures[i][doc] = 1 if sum([random_vectors[i][sc[0]] * sc[1] for sc␣
      ↪in shin_cnt_list]) >= 0.0 else -1

       return signatures
```

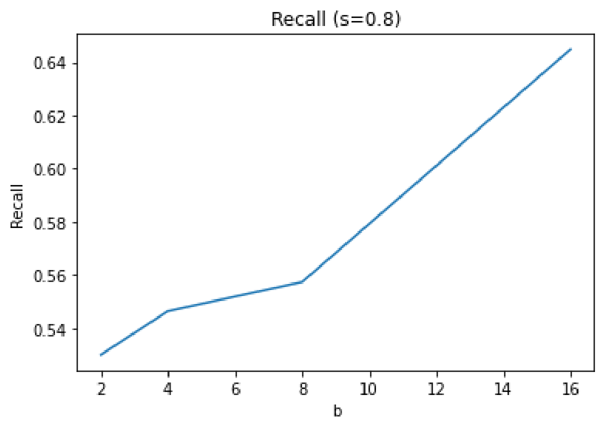
Local Sensitivity Hashing

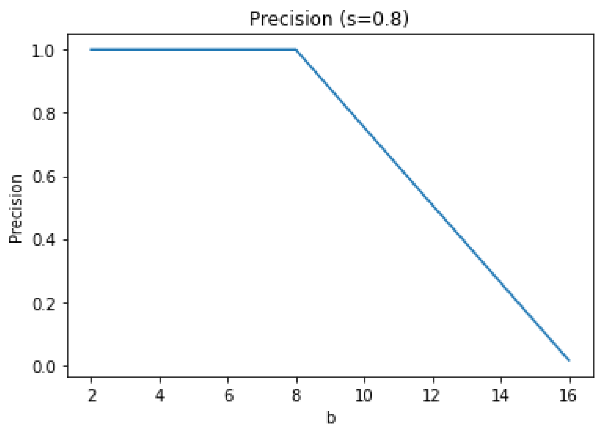
```
[7]: def lsh(signatures, b, r):

         M = signatures.shape[0]   # The number of min-hash functions
         C = signatures.shape[1]   # The number of documents

         assert M == b * r
         r = M // b
         bands = [np.array(signatures[b*r:(b+1)*r]).transpose() for b in range(b)]
         buckets = [{} for band in  bands]
         for i, band in enumerate(bands):
           [buckets[i].update({tuple(col): []}) for col in band]
           [buckets[i][tuple(col)].append(j) for j, col in enumerate(band)]

         candidatePairs = set()
         [[[candidatePairs.add(pair) for pair in itertools.combinations(v, 2)]␣
      ↪for v in bucket.values() if len(v) > 1] for bucket in buckets]

         return candidatePairs
```

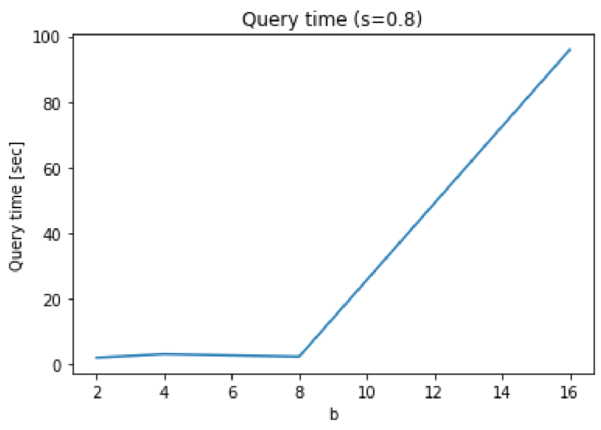
Query Analysis

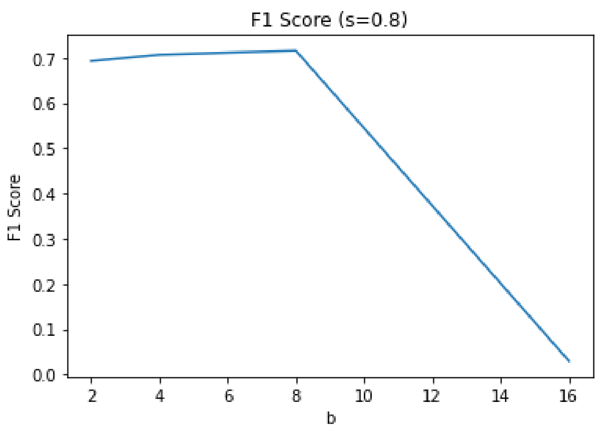
[8]:
```
def query_analysis(signatures, b, s, numConditionPositives):
    M = signatures.shape[0]  # The number of min-hash functions
    assert M % b == 0
    r = M // b
    start = time.time()
    temp = list(lsh(signatures, b, r))
    TP = len([1 for pair in temp if doc_similarity(pair[0], pair[1]) > s])
    end = time.time()
    query_time = end - start
    precision = TP / len(temp)
    recall = TP / numConditionPositives
    f1 = 2 * (precision * recall) / (precision + recall)

    return query_time, precision, recall, f1
```

(b) Result of Implementation With Number of Bands $B \in \{2, 4, 8, 16\}$







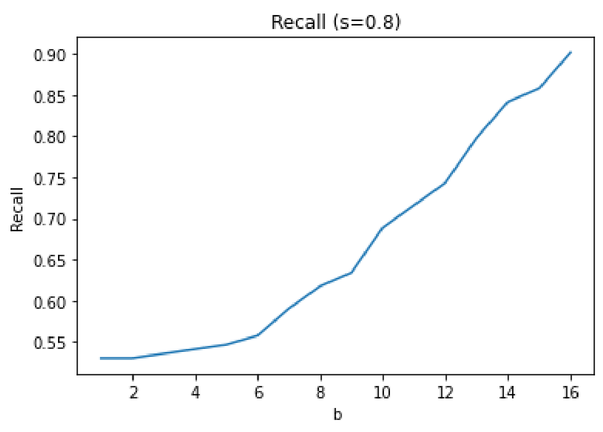


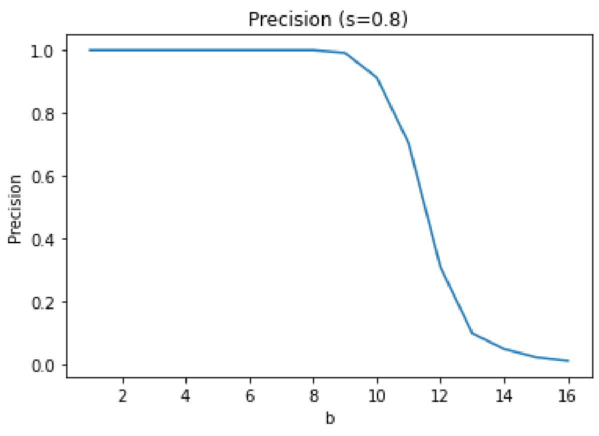
(c) Additional Result With Number of Bands $B \in [1, 16]$

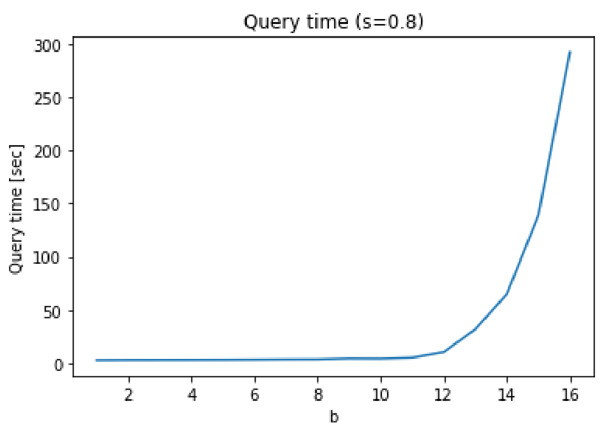
lsh Implementation changed to accept and handle any number of band.

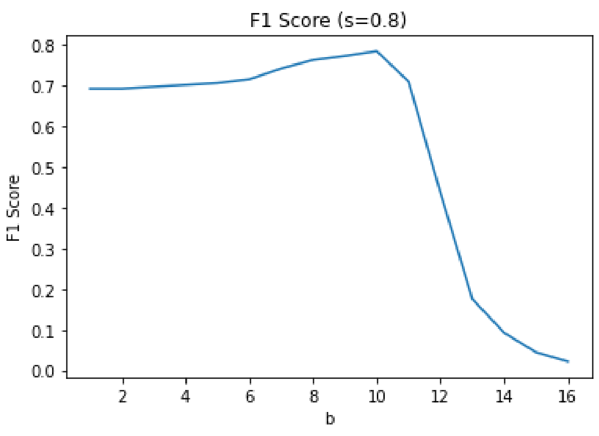
```
[7]: def lsh(signatures, b, r):

         M = signatures.shape[0]  # The number of min-hash functions
         C = signatures.shape[1]  # The number of documents

         r = M // b
         bands=[np.array(signatures[b*r:(b+1)*r]).transpose() for b in range(b-1)]
         bands.append(np.array(signatures[(b-1)*r:]).transpose())
         for i, band in enumerate(bands):
             [buckets[i].update({tuple(col): []}) for col in band]
             [buckets[i][tuple(col)].append(j) for j, col in enumerate(band)]

         candidatePairs = set()
         [[[candidatePairs.add(pair) for pair in itertools.combinations(v, 2)]␣
     ↪for v in bucket.values() if len(v) > 1] for bucket in buckets]

         return candidatePairs
```

Here are the result for $[1, 16]$.


Recall (s=0.8)
Recall
b


Precision (s=0.8)
Precision
b


Query time (s=0.8)
Query time [sec]
b


F1 Score (s=0.8)
F1 Score
b