IERG 4300 Spring 2023 Homework #3

Every Student MUST include the following statement, together with his/her signature in the submitted homework. I declare that the assignment submitted on Elearning system is original except for source material explicitly acknowledged, and that the same or related material has not been previously submitted for another course. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website http://www.cuhk.edu.hk/policy/academichonesty/.

Name Yoo Hyun Jun SID 1155100531

Date 31/03/2023 Signature _____

Q1. (a)

The probability of any pair of items are not similar(similarity with T1) in some row in band = $(1 - T1^r)^B$

Then, inequality will be P1 <= $1 - (1 - T1^r)^B$

If the similarity of any pair of item is below T2, then the probability of any pair of items are not similar in some row in band = $(1 - T2^r)^B$

Then, inequality will be P2 >= $1 - (1 - T2^r)^B$

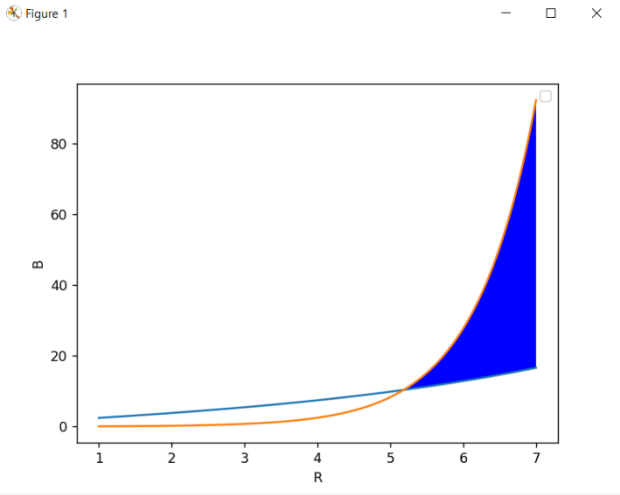
(b)

As we know T1 = 0.8, T2 = 0.3, P1 = 0.98, P2 = 0.02, we can use result in part(a) to derive a pair of values for (r, B). Then, we can change the formula from part(a) into $\log_{1-0.8^r} 0.02 \leq B \leq \log_{1-0.3^r} 0.98$

Then, use r as x-axis and B as y-axis and find the corresponding area in the graph.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import math

def rb_graph(a,b,t1,t2,p1,p2):
    r = np.linspace(a, b, 100)
    b_for_t1 = [math.log((1-p1), (1-t1**i)) for i in r]
    b_for_t2 = [math.log((1-p2), (1-t2**i)) for i in r]
    z1 = np.array(b_for_t1)
    z2 = np.array(b_for_t2)
    plt.plot(r, b_for_t1)
    plt.plot(r, b_for_t2)
    plt.fill_between(r, b_for_t1, b_for_t2,where=z2>z1, facecolor = 'blue')
    plt.xlabel('R')
    plt.ylabel('B')
    plt.legend()
    plt.show()

rb_graph(1,7, 0.8,0.3,0.98,0.02)
```

By using this python code, we can build the graph for r and b.



Graph for r and b

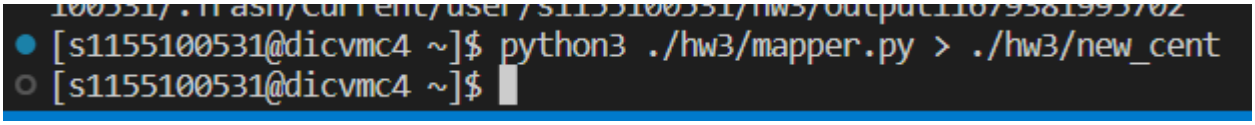
As you can see in this graph, if r and b values that do not belong to the colored area such as (4,20), (6,40) do not satisfy the requirements and r and b values that fall within are satisfied such as (6, 20), (7,40).

Q2. (a)

```
#!/usr/bin/env python

from numpy import random
import numpy as np

file = open("./hw3/MNIST/train_img", "r")
img = []
res = []
random_seed = 1000
random.seed(random_seed)
randoms = [random.randint(1,10000) for i in range(10)]
line_num = 0
sample = []

for line in file:
    line_num += 1
    line = line.strip()
    img = [int(img_item) for img_item in line.split(',')]
    if line_num in randoms:
        res.append(img)

random_img = res
cluster = [0 for i in range(10)]
result = ""
for i in range(10):
    format1 = "Centroid " + str(i) + ": "
    img = [str(res[i][j]) for j in range(784)]
    img = ' '.join(img)
    format1 = format1 + str(cluster[i]) + ", " + img
    print(format1)
```

Mapper.py

In mapper.py, it is for generating initial centroids selecting randomly from training image by using random seed.



Cmd for generating initial centroids file

```python
#!/usr/bin/env python
import sys
from math import*

def distance(x, y):
    return sqrt(sum(pow(int(a) - int(b), 2) for a, b in zip(x, y)))

file = open("new_cent", "r")

centroids = [[0 for i in range(784)] for j in range(10)]

for line in file:
    line = line.strip()
    line, img = line.split(",")
    header, count = line.split(":")
    header, index = header.split(" ")
    cent_img = img.strip().split(" ")
    centroids[int(index)] = cent_img

partial_sums = {}

for i in range(10):
    partial_sums.setdefault(i, [0 for i in range(784)] )

def sum_vector(A,B):
    return [int(x)+int(y) for x,y in zip(A, B)]


cnt = [0 for i in range(10)]

for line in sys.stdin:
    line = line.strip()
    img = [int(item) for item in line.split(",")]
    min_dis = distance(centroids[0] , img)
    # print(min_dis)
    min_idx = 0
    for i in range(1,10):
        dis = distance(centroids[i], img)
        if dis < min_dis:
            min_dis = dis
            min_idx = i

    partial_sums[min_idx] = sum_vector(partial_sums[min_idx], img)
    cnt[min_idx] += 1

i = 0
for key, value in partial_sums.items():
    print("%s\t%s|%s" %(key, str(value)[1:-1], cnt[i]))
    i +=1
```

Mapper1.py

In mapper1.py, we will get partial sum in minimum distance for each image.

```python
#!/usr/bin/env python
import sys

cur_idx = None
cluster = {}
total = 0

centroids = [0 for i in range(784)]

for line in sys.stdin:
    line = line.strip()
    idx, partial = line.split("\t")
    partial_sum, cnt = partial.split("|")
    partial_sum = partial_sum.split(",")
    for i in range(len(partial_sum)):
        partial_sum[i] = int(partial_sum[i].strip())
    idx = int(idx)
    cnt = int(cnt)
    if (cur_idx == idx):
        total += cnt
        centroids = [centroids[i] + int(partial_sum[i]) for i in range(784)]
    else:
        if cur_idx is not None:
            if total > 0:
                new_cent = [str(centroids[i]/total) for i in range(784)]
                new_cent = ' '.join(new_cent)
                total = str(total)
                print("Centroid "+ str(cur_idx) + ": " + str(total) + "," + new_cent)
                total = cnt
                centroids = [0 for i in range(784)]
                centroids = [centroids[i] + int(partial_sum[i]) for i in range(784)]

        elif cur_idx is None:
            total = cnt
            centroids = [centroids[i] + int(partial_sum[i]) for i in range(784)]


    cur_idx = idx


if total > 0:
    new_cent = [str(centroids[i]/total) for i in range(784)]
    new_cent = ' '.join(new_cent)
    total = str(total)
    print("Centroid "+ str(cur_idx) + ": " + str(total) + "," + new_cent)
```

Reducer1.py

In reducer1.py, we will aggregate the result from mappers to renewal the centroids file.

```
for i in `seq 1 10`;
do
        hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
        -D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
        -D mapred.job.name='job' \
        -D mapred.map.tasks=20 \
        -D mapred.reduce.tasks=10 \
        -D mapred.text.key.comparator.options=-k1n \
        -file ./hw3/mapper1.py -mapper mapper1.py \
        -file ./hw3/reducer1.py -reducer reducer1.py \
        -file ./hw3/new_cent \
        -input ./hw3/train_img \
        -output ./hw3/output1

        hdfs dfs -cat ./hw3/output1/* > ./hw3/new_cent
        cat ./hw3/new_cent > ./hw3/new_cent1.txt
        cat ./hw3/new_cent1.txt >> ./res.txt
        hdfs dfs -rm -r ./hw3/output1
done
```

Cmd for mapper1.py, reducer1.py

In this question, I used random seed (1000) to generate initial centroids file. And do 10 iterations.

```
Centroid 1: 6996,0 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 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
Centroid 2: 5201,0 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 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 1 4
Centroid 3: 4109,0 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 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
Centroid 4: 4326,0 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 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
Centroid 5: 4863,0 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 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Centroid 6: 5089,0 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 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
Centroid 7: 6749,0 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 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
Centroid 8: 7539,0 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 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
Centroid 9: 9595,0 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 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
Centroid 0: 5533,0 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 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
```

Result after doing 10 iteration img1

(b)

```python
#!/usr/bin/env python
import sys
from math import*

def distance(x, y):
    return sqrt(sum(pow(float(a) - float(b), 2) for a, b in zip(x, y)))

#file = open("new_cent", "r")
# file = open("cent_2023", "r")
# file = open("cent_random_seed_1", "r")
file = open("cent_1000", "r")
#file = open("./hw3/new_cent", "r")
centroids = [[0 for i in range(784)] for j in range(10)]

for line in file:
    line = line.strip()
    line, img = line.split(",")
    header, count = line.split(":")
    header, index = header.split(" ")
    cent_img = img.strip().split(" ")
    centroids[int(index)] = cent_img

cnt = [0 for i in range(10)]

for line in sys.stdin:
    line = line.strip()
    img = [int(item) for item in line.split(",")]
    min_dis = distance(centroids[0] , img)
    # print(min_dis)
    min_idx = 0
    for i in range(1,10):
        dis = distance(centroids[i], img)
        if dis < min_dis:
            min_dis = dis
            min_idx = i

    img = " ".join(str(k) for k in img)
    print("%s\t%s" %(str(min_idx), img))
```

Mapper2.py

```
[s1155100531@dicvmc4 ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
> -D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
> -D mapred.job.name='job' \
> -D mapred.map.tasks=20 \
> -file ./hw3/mapper2.py -mapper mapper2.py \
> -file ./hw3/cent_1000 \
> -input ./hw3/train_img \
> -output ./hw3/output1
23/03/21 23:43:54 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
```

Cmd for mapper2.py

```
[s1155100531@dicvmc4 ~]$ hdfs dfs -cat ./hw3/output1/* > ./hw3/ran_1000
```

Cmd for getting min_idx of each image for the centroids obtained by the method of (1)

```python
# file = open("./hw3/MNIST/train_img", "r")
file = open("./hw3/MNIST/train_label", "r")


labels = []

for line in file:
    line = line.strip()
    label = line
    labels.append(label)



file = open("./hw3/MNIST/train_img", "r")

img_list = []
i = 0
for line in file:
    line = line.strip()
    img = [int(img_item) for img_item in line.split(',')]
    img_list.append(img)
for i in range(60000):
    img = ' '.join(str(k) for k in img_list[i])

    print("%s\t%s" %(labels[i], img))
```

Join_file.py

```
python3 ./hw3/join_file.py > ./join.txt
```

Cmd for merging train_img file and label file

After generating join file and each min_idx with image file, I use the pandas module in my local machine. First, I merge two files by using image and generating a new table. Then, by using pandas I can calculate the required numbers and accuracy.

```python
import tensorflow as tf
from tensorflow import keras
import sys
import pandas as pd
from pandas import Series, DataFrame

file = open("/content/join")

train_label = []
train_img = []

for line in file:
    line = line.strip()
    label, img = line.split("\t")

    train_label.append(label)
    train_img.append(img)

df1 = DataFrame({'label':train_label, 'img':train_img})

del train_img
del train_label
file = open("/content/ran_1000")

cluster_idx = []
cluster_img = []

for line in file:
    line = line.strip()
    idx, img = line.split("\t")

    idx = idx
    cluster_img.append(img)
    cluster_idx.append(idx)

df2 = DataFrame({'index':cluster_idx, 'img':cluster_img})

del cluster_idx
del cluster_img

res = pd.merge(left = df1, right = df2, how = "left", on = "img")

print(res)
```

Python code to merge two files

```
      label                                                img index
0         5  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...     0
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 ...     3
2         4  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...     6
3         1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...     9
4         9  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...     1
...     ...                                                ...   ...
59995     8  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...     0
59996     3  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...     0
59997     5  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...     1
59998     6  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...     4
59999     8  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...     4

[60000 rows x 3 columns]
```

Result of merging join file and ran_1000 file

```
num_train_images = list(res['index'].value_counts().sort_index())
 [5462, 7088, 4989, 4047, 4283, 4946, 5028, 6865, 7497, 9795]
```

To get train images belongs to the cluster

```
res1 = res.groupby(by = ['index', 'label'], as_index = False).count()
print(res1)
print(res1.loc[res1.groupby(by = ['index'])['img'].idxmax()])
```

To get major label of the cluster and number of correctly clustered images

```
    index  label   img
0       0      0    59
1       0      1    42
2       0      2   250
3       0      3   790
4       0      4    30
..    ...    ...   ...
93      9      5   586
94      9      6   343
95      9      7   363
96      9      8   543
97      9      9   170

[98 rows x 3 columns]
    index  label    img
8       0      8   3498
19      1      9   2322
22      2      2   4123
30      3      0   3813
39      4      0   1501
55      5      6   4270
62      6      4   2978
75      7      7   3382
81      8      3   3978
89      9      1   6571
```

Result of above line

Cluster Index	The label of the cluster	# test images in the cluster	# Correctly clustered images	Classification Accuracy (%)
0	8	5462	3498	64.0424
1	9	7088	2322	32.7596
2	2	4989	4123	82.6418
3	0	4047	3813	94.2179
4	0	4283	1501	35.0455
5	6	4946	4270	86.3324
6	4	5028	2978	59.2283
7	7	6865	3382	49.2644
8	3	7497	3978	53.0612
9	1	9795	6571	67.0852
Total Set		60000	36436	60.7267

The Accuracy of Clustering Performance with Random Seed 1000

I do the same step in different centroid seed file. And generating each table.

Cluster Index	The label of the cluster	# test images in the cluster	# Correctly clustered images	Classification Accuracy (%)
0	1	4953	3093	62.4470
1	1	5957	3602	60.4667
2	7	3364	3004	89.2985
3	0	4766	4464	93.6635
4	4	7220	3278	45.4017
5	9	7321	2366	32.3180
6	3	6788	3496	51.5027
7	6	8315	4707	56.6085
8	5	5804	2470	42.5569
9	8	5512	3187	57.8193
Total Set		60000	33667	56.1117

The Accuracy of Clustering Performance with Random Seed 2023

Cluster Index	The label of the cluster	# test images in the cluster	# Correctly clustered images	Classification Accuracy (%)
0	0	4533	4272	94.2422
1	1	9692	6484	66.9005
2	2	4460	4104	92.0179
3	3	7533	3841	50.9890
4	4	5994	2276	37.9713
5	5	5509	2237	40.6063
6	6	4975	4409	88.6231
7	7	4823	4434	91.9345
8	8	4874	3413	70.0246
9	9	7607	3405	44.7614
Total Set		60000	38875	64.7917

The Accuracy of Clustering Performance with Provided Seed 1

Cluster Index	The label of the cluster	# test images in the cluster	# Correctly clustered images	Classification Accuracy (%)
0	0	4224	3984	94.3182
1	1	9816	6503	66.2490
2	2	4392	4092	93.1694
3	3	7733	3997	51.6876
4	4	5555	2352	42.3402
5	5	4996	1880	37.6301
6	6	4930	4325	87.7282
7	7	6474	3296	50.9113
8	8	4907	3464	70.5930
9	9	6973	2504	35.9010
Total Set		60000	36397	60.6617

The Accuracy of Clustering Performance with Provided Seed 2

I will choose provided seed 1 is the best random seed. Because it has the highest classification accuracy. It means that this seed selected many different labels, which means the initial centroids are far away from each other.

(c)

In this part, I selected the provided seed 1 centroid file. And do the same step as I did in (b) part by using test image and label file.

```
[s1155100531@dicvmc4 ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
> -D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
> -D mapred.job.name='job' \
> -D mapred.map.tasks=20 \
> -file ./hw3/mapper2.py -mapper mapper2.py \
> -file ./hw3/cent_random_seed_1 \
> -input ./hw3/test_img \
> -output ./hw3/output1
23/03/26 18:39:21 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
```

Cmd for running mapper2.py for test image

```
        File Input Format Counters
                Bytes Read=20759811
        File Output Format Counters
                Bytes Written=18289443
23/03/26 18:40:02 INFO streaming.StreamJob: Output directory: ./hw3/output1
[s1155100531@dicvmc4 ~]$ hdfs dfs -cat ./hw3/output1/* > ./hw3/test_seed
[s1155100531@dicvmc4 ~]$
```

Cmd for getting min_idx of each image for the centroids

Cluster Index	The label of the cluster	# test images in the cluster	# Correctly clustered images	Classification Accuracy (%)
0	0	773	722	93.4023
1	1	1586	1094	68.9786
2	2	750	689	91.8667
3	3	1333	680	51.0128
4	4	1019	391	38.3710
5	5	847	365	43.0933
6	6	830	729	87.8313
7	7	776	716	92.2680
8	8	811	561	69.1739
9	9	1275	584	45.8040
Total Set		10000	6581	65.81

The Accuracy of Clustering Performance on the Test Data

3.(a)

1. Initialize q_k, π_k for each cluster

For k in len(cluster_list):

q_k **<- random probability**

π_k **<-** $\frac{Number\ of\ points\ in\ k}{Total\ number\ of\ points}$

2. Mapper(E – step) : key = k, value = (a set of binary variables, γ)

For line in lines:

$$\gamma(z_{nk}) = \frac{\pi_k p(x_n|q_k)}{(\sum_{j=1}^{K} \pi_j p(x_n|q_k))}$$

Print (k, (set of binary variables, $\gamma(z_{nk})$)

3. Reducer(M – step) : key = k, value = (q_k, π_k)

For line in lines:

If K_{cur} = K :

 # Sum the intermediate result for the same K

$$q_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) x_n}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

$$\pi_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})}{N}$$

Else if K_{cur} != K :

Print(k, (q_k, π_k))

K_{cur} **= K**

Print(k, (q_k, π_k))