

# IERG4300/ESTR4300 Fall 2020 Homework 3

Due date: Nov 23, 2020 11:59 pm

*The solution will be posted right after the deadline, so no late homework will be accepted!*

Every Student **MUST** include the following statement, together with his/her signature in the submitted homework.

*I declare that the assignment submitted on the 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/>.*

Signed (Student ) Date: 23-11-2020

Name Chan Kai Yan SID 1155124923

## Submission notice:

- Submit your report in a single PDF document on Elearning

## General homework policies:

A student may discuss the problems with others. However, the work a student turns in must be created **COMPLETELY** by oneself **ALONE**. A student may not share **ANY** written work or pictures, nor may one copy answers from any source other than one's own brain.

Each student **MUST LIST** on the homework paper the **name of every person he/she has discussed or worked with**. If the answer includes content from any other source, the student **MUST STATE THE SOURCE**. Failure to do so is cheating and will result in sanctions. Copying answers from someone else is cheating even if one lists their name(s) on the homework.

If there is information you need to solve a problem but the information is not stated in the problem, try to find the data somewhere. If you cannot find it, state what data you need, make a reasonable estimate of its value, and justify any assumptions you make. You will be graded not only on whether your answer is correct, but also on whether you have done an intelligent analysis.

Name: Chan Kai Yin

SID: 1155124983

Q1)

a.)

$\text{Sim}(C1, C2) \geq T1$

Probability C1, C2 identical in at least 1 of B  $\geq 1 - (1 - T1^r)^B \geq P1$

$1 - (1 - T1^r)^B \geq P1$

$\text{Sim}(C1, C2) < T2$

Probability C1, C2 identical in at least 1 of B  $< 1 - (1 - T2^r)^B < P2$

$P2 > 1 - (1 - T2^r)^B$

b.)

$1 - (1 - 0.85^r)^B \geq 0.99$

$0.01 > 1 - (1 - 0.5^r)^B$

$B = 150, r = 20$

Q2)

a.)

it should be 37. Cause in my view, give a same letter, there are quit a lot of differences between the uppercase and lowercase in shape, like Q and q, B and b. but there is 15 lowercase letters is similar to its uppercase which is C, I, J, K, L, M, O, P, S, U, V, W, X, Y and Z.

so we need to distinguish 52 case and set  $K = 37$ .

b.)

```
bin_t1-6.py x pdf merge.py x hw3.py x bin_txt x
file_name = 'emnist-letters-train-images-idx3-ubyte'

with open("bin_t_2.txt", "a") as ft:
    with open(file_name, 'rb') as f:
        f.read(16)
        for i in range(0, 124800):
            for j in range(0, 784):
                ft.write(str(int(ord(f.read(1)))))
                ft.write(' ')
            ft.write('\n')

# content = open(file_name, 'rb').read()
# int_f = content.decode()
```

I use this program to transfer the binary file to txt file for later on caculation.

```
dic14.ie.cuhk.edu.hk - PuTTY
import random as rnd
import math
import numpy as np

def rand_cen():
    file_name = "cen_1.txt"
    with open(file_name, "a") as f:
        for i in range(0, 37):
            f.write(str(i)+'\t')
            for j in range(0, 784):
                f.write(str(rnd.randint(0, 255)))
                f.write(' ')
            f.write('\n')

rand_cen()

~
~
~
~
~
~
```

I use this to initialize 3 different random centroids.

Later on, I realize that this is not a good method cause it will generate a lot of outlier and eventually only a few or none data will assigned to that cluster (see table 1). So I update a new-version of finding the first centroids.

dic14.ie.cuhk.edu.hk - PuTTY

```
import random as rnd
import math
import numpy as np

a = dict()
centroid_list = []
blf = open("bin_t_2.txt").readlines()

def rand_cen():
    count = 0
    file_name = "cen_1.txt"
    for i in open("label_out.txt", 'r').readlines():
        i = int(i.strip())-1
        if not a.get(str(i)):
            a[str(i)] = count
        elif not a.get(str(i+26)):
            a[str(i+11)] = count
        count+=1


    with open(file_name, "w") as bf:
        for k,v in a.items():
            str_l = str(str(k) + '\t' + str((blf[int(v)])))
            bf.write(str_l)
            #bf.write('\n')

rand_cen()
~
~
~
~
~
~
~
```

Which is, use the label 0 data for the first centroid, and label 1 data for the second centroid, and so on...

I use this to run the test2 and test3, and it turns out having a better performance than the first method(table 1).

The k-means mapper:

 dic14.ie.cuhk.edu.hk - PuTTY

```
#!/usr/bin/env python
import sys
import math
import numpy as np
k = 37

cen = np.zeros((k, 784))
up_cen = np.zeros((k, 784))
arr_cen = np.zeros(k)

arr_count_cen_v = np.ones(k)
arr_count_cen = np.ones(k)

def p_c_distance(pt, index_cen):
    dist = 0
    result = pt - cen[index_cen]
    dist += np.sum((result*result))
    return dist

def read_to_cen():
    with open("cen_1.txt", "r") as f_c:
        # count = 0
        for i in f_c.readlines():
            x = []
            j = i.strip().split('\t')[0]
            k = i.strip().split('\t')[1]
            for item in k.strip().split():
                x.append(float(item))
            x = np.array(x)
            cen[int(j)] = x
            # count += 1

def tran_a(pt):
    x = []
    for i in pt.strip().split():
        x.append(float(i))
    x = np.asarray(x)
    return x

def cal_new_cen(pt, pos):
    new_v = pt
    new_v = np.array(new_v)
    # for j in range(0, 784):
    up_cen[pos] = up_cen[pos] + new_v
    arr_count_cen[pos] += 1

read_to_cen()
up_cen = cen.copy()

# sys.stdin
```

dic14ie.cuhkeduhk - PuTTY

```
def read_to_cen():
    with open("cen_1.txt", "r") as f_c:
        # count = 0
        for i in f_c.readlines():
            x = []
            j = i.strip().split('\t')[0]
            k = i.strip().split('\t')[1]
            for item in k.strip().split():
                x.append(float(item))
            x = np.array(x)
            cen[int(j)] = x
            # count += 1

def tran_a(pt):
    x = []
    for i in pt.strip().split():
        x.append(float(i))
    x = np.asarray(x)
    return x

def cal_new_cen(pt, pos):
    new_v = pt
    new_v = np.array(new_v)
    # for j in range(0, 784):
    up_cen[pos] = up_cen[pos] + new_v
    arr_count_cen[pos] += 1

read_to_cen()
up_cen = cen.copy()

# sys.stdin

for line in sys.stdin:
    min_d = 9999999999
    pos = 53
    line = tran_a(line)
    for i in range(0, k):
        dist = p_c.distance(line, i)
        if min_d > dist:
            min_d = dist
            pos = i
    arr_cen[pos] = arr_cen[pos] + min_d
    arr_count_cen_v[pos] += 1
    cal_new_cen(line, pos)

for i in range(0, k):
    print(
        '%s\t%s\t%s' % (i, arr_cen[i] / int(arr_count_cen_v[i]), ' '.join(map(str, (up_cen[i] / int(arr_count_cen[i])))))
    )
```

## The k-means reducer:

dic14ie.cuhkeduhk - PuTTY

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

import sys
import numpy as np
k = 37
wh_v = np.zeros((k, 784), dtype='float')
wh_sum = np.zeros(k)

arr_count_cen_v = np.zeros(k)
arr_count_cen = np.zeros(k)

# for line in sys.stdin:
for line in sys.stdin:
    num_clu = int(line.strip().split('\t')[0])
    part_sum = float(line.strip().split('\t')[1])
    part_v = line.strip().split('\t')[2]

    arr_part_v = []
    for i in part_v.strip().split():
        arr_part_v.append(float(i))
    arr_part_v = np.asarray(arr_part_v)

    for i in range(0, 784):
        c = wh_v[num_clu][i] + arr_part_v[i]
        wh_v[num_clu][i] = c

    arr_count_cen[num_clu] += 1
    wh_sum[num_clu] = wh_sum[num_clu] + part_sum
    arr_count_cen_v[num_clu] += 1

for i in range(0, k):
    if arr_count_cen[i] > 0:
        print('%s\t%s' % (i, ' '.join(map(str, (wh_v[i] / arr_count_cen[i])))))
    ~
    ~
    ~
    ~
    ~
```

 dic14.ie.cuhk.edu.hk - PuTTY

~

This is one of the updated centroids.

c.)

```
gen_cen.py  kmean_mapper.py  km_map_loc.py  kmean_reducer.py  check.py  pre_proc

import sys
import struct
import numpy as np

|

with open("emnist-letters-train-labels-idx1-ubyte", "rb") as lab_file:
    magic, label_num = struct.unpack(">II", lab_file.read(8))
    arr_lab = np.fromfile(lab_file, dtype=np.uint8)

with open("labels.txt", "w") as f:
    for line in arr_lab:
        f.write(str(line))
        f.write("\n")
```

I use this to extract the labels file.

And then I calculate the accuracy with this program.

```
#!/usr/bin/env python
import sys
from operator import itemgetter
from collections import Counter
from collections import OrderedDict

cen = np.zeros((37, 784))
arr_cen = np.zeros(37)
a = defaultdict(list)

with open("label_out.txt", "r") as lb:
    labs = []
    for line in lb.readlines():
        labs.append(int(line.strip()))

def read_to_cen():
    with open("cen_1.txt", "r") as f_c:
        # count = 0
        for i in f_c.readlines():
            x = []
            j = i.strip().split('\t')[0]
            k = i.strip().split('\t')[1]
            for item in k.strip().split():
                x.append(float(item))
            x = np.asarray(x)
            cen[int(j)] = x
            # count += 1

def p_c_distance(pt, index_cen):
    dist = 0
    result = pt - cen[index_cen]
    dist += np.sum(np.square(result))
    return dist

def tran_a(pt):
    x = []
    for i in pt.strip().split():
        x.append(float(i))
    x = np.asarray(x)

    for i, j in a.items()
```



```

mapper.py × km_map_loc.py × make_cen.py × fromnumeric.py × km
def tran_a(pt):
    x = []
    for i in pt.strip().split():
        x.append(float(i))
    x = np.asarray(x)
    return x

read_to_cen()

l_count = 0
# out = open('out.txt', 'w')
for line in open("bin_t.2.txt", 'r').readlines():
    min_d = 9999999999
    pos = 53
    line = tran_a(line)
    for i in range(0, 37):
        dist = p_c_distance(line, i)
        # print(dist)
        if min_d > dist:
            min_d = dist
            pos = i
    a[str(pos)].append(l_count)
    l_count += 1

a = OrderedDict(sorted(a.items(), key=lambda t: int(t[0])))

a2 = open('a_2.txt', 'w')
a3 = open('a_3.txt', 'w')
a4 = open('a_4.txt', 'w')
a5 = open('a_5.txt', 'w')

for i, j in a.items():
    corr = 0
    labels = []
    for k in list(j):
        labels.append(int(labs[int(k)]))
    # ctr = Counter(labels)
    # second_most_common_value, its_frequency = ctr.most_common(2)

    for i, j in a.items()

```

the label 24 length 3335 and the accur = 0.318944652773613106

Python Console Terminal

Day 10:33)

```

        # print(dist)
        if min_d > dist:
            min_d = dist
            pos = i
    a[str(pos)].append(l_count)
    l_count += 1

a = OrderedDict(sorted(a.items(), key=lambda t: int(t[0])))

a2 = open('a_2.txt', 'w')
a3 = open('a_3.txt', 'w')
a4 = open('a_4.txt', 'w')
a5 = open('a_5.txt', 'w')

for i, j in a.items():
    corr = 0
    labels = []
    for k in list(j):
        labels.append(int(labs[int(k)]))
    # ctr = Counter(labels)
    # second_most_common_value, its_frequency = ctr.most_common(2)
    mos_f = max(set(labels), key=labels.count)

    # a[str(i)].append(int(second_most_common_value[1]))
    a[str(i)].append(mos_f)
    # or int(labs[int(c_line)]) == a[str(i)][-2]
    for c_line in list(j):
        if int(labs[int(c_line)]) == a[str(i)][-1]:
            corr += 1
    real_len = len(a[str(i)]) - 1
    print("For the", i, "th cluster, the label ", a[str(i)][-1], ", length", real_len, "and the accur = ",
          float(corr) / real_len)

    print(real_len, file=a2)
    print(a[str(i)][-1], file=a3)
    print(corr, file=a4)
    print(float(corr) / real_len, file=a5)

```

The Accuracy of Clustering Performance with Random Seed 1:

Cluster Number	# train images belongs to the cluster	Label of the cluster	# correctly clustered images	Classification Accuracy (%)
0	4162	26	1362	0.327247
1	3108	15	1321	0.425032
2	3614	10	977	0.270338
3	2870	15	1104	0.384669
4	2513	26	2003	0.797055
5	0	0	0	0
6	5262	18	2239	0.425504
7	0	0	0	0
8	7132	24	2245	0.314778
9	4635	8	1384	0.298598
10	3227	2	856	0.265262
11	9610	9	3211	0.334131
12	5089	16	2199	0.432108
13	3427	1	760	0.221768
14	2366	7	485	0.204987
15	3440	23	2401	0.697965
16	3487	21	1407	0.403499
17	4606	2	1268	0.275293
18	4564	22	1589	0.34816
19	0	0	0	0
20	0	0	0	0
21	5904	10	1054	0.178523
22	3438	3	1755	0.510471
23	3389	1	589	0.173798
24	4480	11	822	0.183482
25	5067	23	1306	0.257746
26	3926	22	1339	0.34106
27	4064	4	1654	0.406988
28	2910	21	1092	0.375258
29	2945	5	1947	0.661121

30	3556	19	2534	0.712598
31	0	0	0	0
32	0	0	0	0
33	4928	6	1391	0.282265
34	0	0	0	0
35	4381	13	3668	0.837252
36	2700	15	1523	0.564074
Total Set	124800	N/A	47485	0.380489

The Accuracy of Clustering Performance with Random Seed 2:

Cluster Number	# train images belongs to the cluster	Label of the cluster	# correctly clustered images	Classification Accuracy (%)
0	3656	1	794	0.217177
1	2524	26	2154	0.853407
2	3219	3	998	0.310034
3	2921	2	607	0.207806
4	2835	3	785	0.276896
5	2861	6	1049	0.366655
6	2211	17	1062	0.480326
7	4523	8	1882	0.416096
8	6626	9	2310	0.348627
9	2145	10	1064	0.496037
10	3407	14	897	0.263281
11	2340	1	839	0.358547
12	3229	13	2907	0.900279
13	2658	11	661	0.248683
14	2233	15	805	0.360502
15	3651	5	2050	0.56149
16	4557	6	1601	0.351328
17	2809	17	1190	0.423638
18	2696	19	2360	0.875371
19	2542	10	752	0.29583
20	2394	10	699	0.29198

21	3166	11	2098	0.662666
22	7122	9	1600	0.224656
23	4159	13	1173	0.282039
24	2693	14	1652	0.613442
25	3221	15	2282	0.708476
26	4169	16	1995	0.478532
27	3921	4	1200	0.306044
28	3545	18	2216	0.625106
29	3151	2	1038	0.329419
30	2580	4	1005	0.389535
31	3168	21	1803	0.569129
32	2945	22	1862	0.632258
33	4090	23	3333	0.814914
34	4494	24	2568	0.571429
35	3004	22	1181	0.393142
36	3335	26	1037	0.310945
Total Set	124800	N/A	55509	0.444784

The Accuracy of Clustering Performance with Random Seed 3:

Cluster Number	# train images belongs to the cluster	Label of the cluster	# correctly clustered images	Classification Accuracy (%)
----------------	---------------------------------------	----------------------	------------------------------	-----------------------------

Cluster Number	# train images belongs to the cluster	Label of the cluster	# correctly clustered images	Classification Accuracy (%)
0	3656	<u>1</u>	794	0.217177
1	2524	26	2154	0.853407
2	3219	3	998	0.310034
3	2921	2	607	0.207806

4	2835	3	785	0.276896
5	2861	6	1049	0.366655
6	2211	17	1062	0.480326
7	4523	8	1882	0.416096
8	6626	9	2310	0.348627
9	2145	10	1064	0.496037
10	3407	14	897	0.263281
11	2340	1	839	0.358547
12	3229	13	2907	0.900279
13	2658	11	661	0.248683
14	2233	15	805	0.360502
15	3651	5	2050	0.56149
16	4557	6	1601	0.351328
17	2809	17	1190	0.423638
18	2696	19	2360	0.875371
19	2542	10	752	0.29583
20	2394	10	699	0.29198
21	3166	11	2098	0.662666
22	7122	9	1600	0.224656
23	4159	13	1173	0.282039
24	2693	14	1652	0.613442
25	3221	15	2282	0.708476
26	4169	16	1995	0.478532
27	3921	4	1200	0.306044
28	3545	18	2216	0.625106
29	3151	2	1038	0.329419
30	2580	4	1005	0.389535
31	3168	21	1803	0.569129
32	2945	22	1862	0.632258
33	4090	23	3333	0.814914
34	4494	24	2568	0.571429
35	3004	22	1181	0.393142
36	3335	26	1037	0.310945
Total Set	124800	N/A	55509	0.444478

So, the seed 2 have the best performance, I use the seed 2 in part d.

d.)

I use the same program above to generate the label file and the smaller-size data file.  
And use the check program.

0	613	1	142	0.231648
1	404	26	341	0.844059
2	526	3	171	0.325095
3	483	2	94	0.194617
4	446	7	127	0.284753
5	483	16	172	0.356108
6	350	17	172	0.491429
7	724	8	317	0.437845
8	1141	9	382	0.334794
9	381	10	179	0.469816
10	597	14	159	0.266332
11	399	1	139	0.348371
12	519	13	474	0.913295
13	459	11	110	0.239651
14	370	15	146	0.394595
15	600	5	339	0.565
16	753	6	250	0.332005
17	446	17	195	0.43722
18	450	19	403	0.895556
19	408	10	126	0.308824
20	421	10	119	0.28266
21	512	11	314	0.613281
22	1209	9	271	0.224152
23	666	13	197	0.295796
24	456	14	277	0.607456
25	527	15	376	0.713472
26	690	16	329	0.476812
27	631	4	178	0.282092
28	561	18	350	0.623886
29	519	2	193	0.371869
30	463	4	200	0.431965
31	563	21	336	0.596803

32	516	22	316	0.612403
33	704	23	591	0.839489
34	745	24	411	0.551678
35	495	22	187	0.377778
36	570	26	164	0.287719
Total Set	20800	N/A	9247	0.444567

Q3)

a.)

E step: Assign each  $x_n$  an assignment score  $\gamma(z_{n,k})$  for each cluster  $k$ .

Initialize  $\pi_k, q_k$

summ = 0

for  $j$  in 1 to  $k$

    summ +=  $\pi_k * p(x_n | q_j)$

$\gamma(z_{n,k}) = \pi_k (p(x_n | q_k)) / \text{summ}$

M step: using  $\gamma(z_{n,k})$ , output the new  $\pi_k, q_k$  for each cluster  $k$

summ = 0

summ\_2 = 0

summ\_3 = 0

For  $n$  in 1 to  $N$

    summ +=  $\gamma(z_{n,k})$

$\pi_k = \text{summ} / N$

for  $n$  in 1 to  $N$

    summ\_2 +=  $\gamma(z_{n,k})(x_n)$

    summ\_3 +=  $\gamma(z_{n,k})$

$q_k = \text{summ}_2 / \text{summ}_3$

first run the E-step, using the result pass to M step to update  $\pi_k$  and  $q_k$ . And then pass those parameter back to E-step and loop this process until the likelihood value or parameters coverage and then stop.

b.)

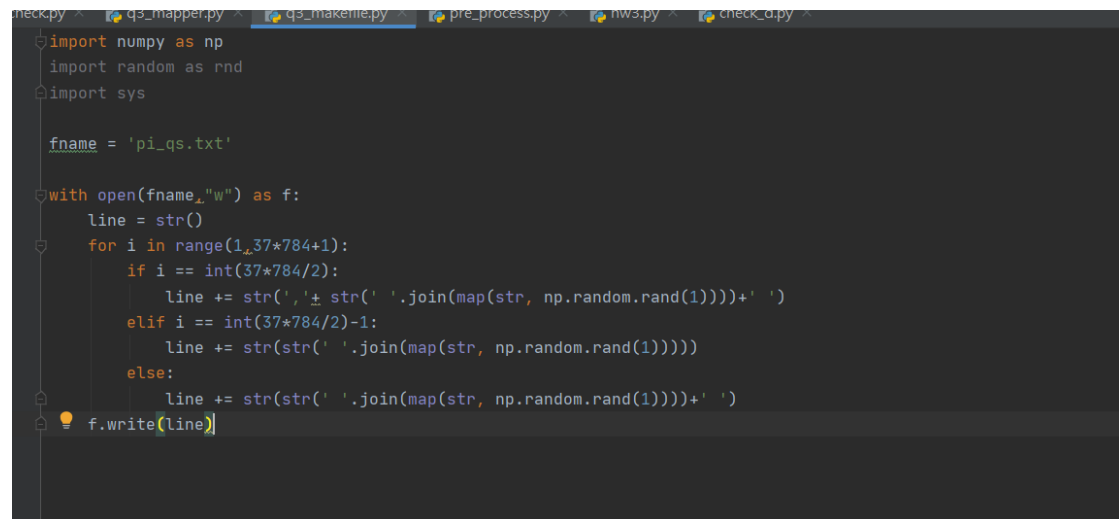
```
1 file_name = 'emnist-letters-test-images-idx3-ubyte'
2
3
4 with open("binary_file.txt","w") as ft:
5     with open(file_name, 'rb') as f:
6         f.read(16)
7         for i in range(0,20800):
8             for j in range(0, 784):
9                 if int(ord(f.read(1))) > 0:
10                     ft.write(str(1))
11                 else:
12                     ft.write(str(0))
13                 ft.write(' ')
14             ft.write('\n')
15
16
17 # content = open(file_name, 'rb').read()
18 # int_f = content.decode()
19
20
```

with open("binary\_file.txt","w"... with open(file\_name, 'rb') as f for i in range(0,20800)



I use this to generate the binarization file.

Then I use this program to Initialize  $\pi_k$ ,  $q_k$  value.

A screenshot of a code editor with a dark theme. The editor shows a Python script with the following code:

```
import numpy as np
import random as rnd
import sys

fname = 'pi_qs.txt'

with open(fname, "w") as f:
    line = str()
    for i in range(1, 37*784+1):
        if i == int(37*784/2):
            line += str(',') + str(' '.join(map(str, np.random.rand(1)))) + ' '
        elif i == int(37*784/2)-1:
            line += str(str(' '.join(map(str, np.random.rand(1)))) + ' ')
        else:
            line += str(str(' '.join(map(str, np.random.rand(1)))) + ' ')
    f.write(line)
```

The script imports numpy and random modules, sets a filename 'pi\_qs.txt', and then opens a file for writing. It iterates from 1 to 37\*784+1, building a string 'line' with random values and commas. Finally, it writes the string to the file.