Name: Chan Kai Yin
SID: 1155124983

Q1
a.)

V^T = [[-2.72892220e-01 -5.11655370e-01 -3.65997772e-01 -2.18288336e-01
  -4.34381463e-01 -3.54970449e-01 -4.09198211e-01]
 [6.94767979e-01 -4.45527084e-01 4.19050700e-01 -3.70815387e-01
  -3.30113929e-02 -6.71226587e-02 1.00164115e-02]
 [-4.71915547e-01 7.92010632e-02 7.22885353e-01 -1.76991435e-01
  -3.69690775e-01 2.39586242e-01 -1.51858060e-01]
 [-7.26004179e-02 1.71285014e-01 1.72998791e-01 -1.68478339e-01
   6.31512574e-01 -2.85638770e-01 -6.53207053e-01]
 [-3.76000996e-01 -2.79589227e-01 -1.45935518e-01 -6.05475565e-01
   4.24833065e-01 2.46807177e-01 3.88790900e-01]
 [2.10084204e-01 6.41844491e-01 -2.43855394e-01 -6.22519677e-01
  -3.07018643e-01 -2.47808353e-02 -4.50503897e-02]
 [-1.71289418e-01 1.18188612e-01 2.39139495e-01 -3.16831951e-02
  -1.08660778e-04 -8.17906830e-01 4.79091087e-01]]

U = [[-0.30575453 -0.28084225 -0.17900878 -0.57665451 -0.64311746 -0.22241214]
 [-0.27136687 0.21692604 0.7668416   0.17114917 -0.42910594
0.27898591]
 [-0.2314542   0.47006359 -0.01477209 0.27240655 -0.05702831 -0.80485614]
 [-0.35742914 -0.46995441 -0.30288063 0.71228838 -0.21029095 0.08985302]
 [-0.70262853 -0.2263871   0.2179558 -0.2271184   0.59426503 -0.05313762]
 [-0.39731688 0.61725415 -0.49035722 -0.07095566 -0.04052885 0.4626112]]

Sigma = [[24.61024449 0.   0.   0.   0.   0.
   0.   ]
 [0.   11.44513576 0.   0.   0.   0.
   0.   ]
 [0.   0.   9.21913422 0.   0.   0.
   0.   ]
 [0.   0.   0.   7.66940571 0.   0.
   0.   ]
 [0.   0.   0.   0.   4.81782202 0.

```
      0.              ]
  [ 0.                0.                0.                0.                0.
1.82238996
      0.              ]]
```

And the = U*Sigma*V^T

```
[[ 2.00000000e+00    5.00000000e+00    2.24820162e-15    6.00000000e+00
     3.53883589e-15    3.00000000e+00    5.00000000e+00]
  [ 1.00000000e+00    4.00000000e+00    9.00000000e+00    4.05231404e-15
     7.21644966e-16    3.00000000e+00    4.74967288e-15]
  [ 5.00000000e+00    5.55111512e-16    5.00000000e+00    9.99200722e-16
     4.00000000e+00    1.00000000e+00    1.00000000e+00]
  [-7.84095011e-16    8.00000000e+00 -3.54577478e-15    4.00000000e+00
     8.00000000e+00    1.00000000e+00 -4.55364912e-16]
  [ 1.00000000e+00    9.00000000e+00    6.00000000e+00    3.00000000e+00
     7.00000000e+00    8.00000000e+00    9.00000000e+00]
  [ 1.00000000e+01    2.00000000e+00    3.00000000e+00 -1.22124533e-15
     5.00000000e+00    2.00000000e+00    5.00000000e+00]]
```

Which identical to A. Transform it to a 2-D space.

The Whole Doc:

```
[[ -7.52469371    -3.2142777 ]
  [ -6.678405       2.48274797]
  [ -5.69614447     5.37994158]
  [ -8.79641846    -5.37869202]
  [-17.29185993    -2.59103105]
  [ -9.77806553     7.0645575 ]]
```

Program:

```
from numpy import array
import numpy as np
from numpy import diag
from numpy import zeros
from scipy.linalg import svd

A = array([
    [2,5,0,6,0,3,5],
    [1,4,9,0,0,3,0],
    [5,0,5,0,4,1,1],
    [0,8,0,4,8,1,0],
    [1,9,6,3,7,8,9],
    [10,2,3,0,5,2,5],
])
U, s, VT = svd(A)

a_0 = A.shape[0]
a_1 = A.shape[1]
sigma = zeros((a_0, a_1))
sigma[:a_0, :a_0] = diag(s)

n = 2

sigma = sigma[:, :n]
VT = VT[:n, :]

doc7 = array([[0,5,4,0,3,2,2]])
# print(doc7.dot(VT.T))

# print(U)
# print(sigma)
# print(VT)
print(A.dot(VT.T))
```

b.)1

Doc7:

[[-6.85374965 -0.76467929]]

2.

Word space: 0.8633266263033915

Concept space: 0.9914060814075807

Program:

```
print(VT)

doc3 = A[2]
dco6 = A[5]
doc3_2d = doc3.dot(VT.T)
doc6_2d = dco6.dot(VT.T)
result = 1 - spatial.distance.cosine(doc3, dco6)
print(result)

result = 1 - spatial.distance.cosine(doc3_2d, doc6_2d)
print(result)
```

It extends from the above program.

2.)

```
from numpy import array
import numpy as np
from numpy import diag
from numpy import zeros
from scipy.linalg import svd
from scipy import spatial
from sklearn.decomposition import PCA

pca = PCA(25)

file = "bin_t_2.txt"

line_ls = []
with open(file, 'r') as f:
    for line in f.readlines():
        line_ls.append(line.strip().split())
A = np.array(line_ls)
line_ls = []
A = pca.fit_transform(A)

with open("test_bin_2d.txt", 'w') as f:
    for line in A:
        mat_str = " ".join(map(str, line))
        f.write(mat_str)

A = A - A.mean(axis=0)
# print(A)
```

I use this one to do the dimension reduction of the original training data set. So as the label file and testing files.

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.)

I use this program to transform those data set file.

```python
from numpy import array
import numpy as np
from numpy import diag
from numpy import zeros
from scipy.linalg import svd
from scipy import spatial
from sklearn.decomposition import PCA

pca = PCA(25)

file = "bin_t_2.txt"

line_ls = []
with open(file, 'r') as f:
    for line in f.readlines():
        line_ls.append(line.strip().split())
A = np.array(line_ls)
line_ls = []
A = pca.fit_transform(A)

with open("test_bin_2d.txt", 'w') as f:
    for line in A:
        mat_str = " ".join(map(str,line))
        f.write(mat_str)

A = A - A.mean(axis=0)
# print(A)
```

I use the previous k-mean MapReduce program (attached below), with some fine tune, including change all 784 to 25.

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
```

dic14.ie.cuhk.edu.hk - 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 = 99999999999
    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:

dic14.ie.cuhk.edu.hk - 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])))))
~
~
~
~
~
~
~
~
~
```

I use this .sh file to run the k-means. For each caculation, I run 10 times to converage the k-means.

dic14.ie.cuhk.edu.hk - PuTTY

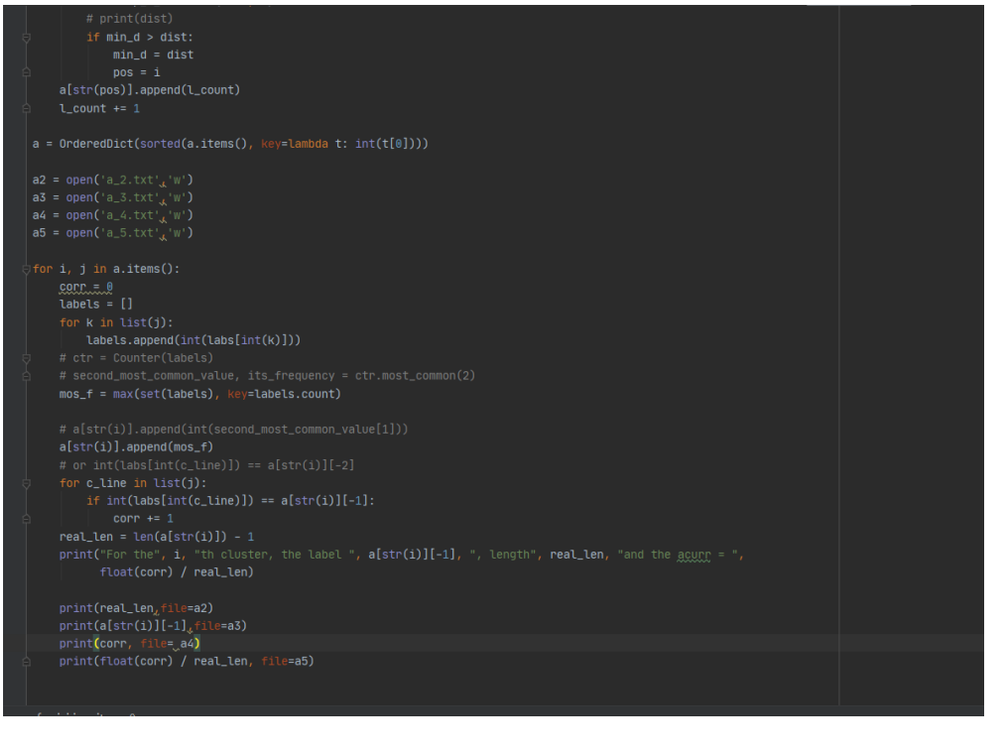
```
#!/bin/bash

hdfs dfs -rm -r ./k_output_1_37

hadoop jar /usr/hdp/2.4.2.0-258/hadoop-mapreduce/hadoop-streaming.jar \
-D mapred.map.tasks=20 \
-D mapred.reduce.tasks=20 \
-file cen_1.txt \
-input ./kmeans_input/bin_t_2.txt \
-file kmeans_mapper_37.py -mapper kmeans_mapper_37.py \
-file kmeans_reducer_37.py -reducer kmeans_reducer_37.py \
-output k_output_1_37

rm cen_1.txt

hdfs dfs -cat ./k_output_1_37/part-* > cen_1.txt


~
~
~
~
~
~
~
```

And then I calculate the accuracy with this program, same as the previous homework. Again with some fine tune, including change all 784 to 25.

```
#!/usr/bin/env python
import ...
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
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 = 99999999999
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()
Python Console
Terminal
lay 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 acurr = ",
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	3723	1	821	0.220521
1	2774	26	2214	0.798125
2	3104	3	753	0.24259
3	3556	8	994	0.279528
4	3116	7	711	0.228177
5	2643	6	1023	0.38706
6	2294	17	1123	0.489538
7	3634	8	1442	0.396808
8	5171	9	1753	0.339006
9	3863	6	1581	0.409267
10	4002	14	1094	0.273363
11	2653	15	724	0.272899
12	3694	13	3312	0.896589
13	2503	1	898	0.358769
14	2836	3	1296	0.456982
15	2663	5	1833	0.688321
16	2345	18	916	0.390618
17	3821	17	1089	0.285004
18	3433	19	2547	0.741917
19	3752	20	561	0.14952
20	4307	10	1945	0.45159
21	3027	11	2046	0.675917
22	5485	9	1667	0.30392
23	4433	23	965	0.217686
24	2499	14	1596	0.638655
25	3136	15	2214	0.705995
26	3576	16	1815	0.50755
27	3781	4	2018	0.533721
28	2981	18	1730	0.580342
29	3350	2	1036	0.309254

30	4708	25	1372	0.291419
31	2819	21	1873	0.66442
32	2548	22	1721	0.675432
33	3671	23	2946	0.802506
34	3073	24	2296	0.747153
35	2515	22	1544	0.613917
36	3311	26	997	0.301117
Total Set	124800	N/A	56466	0.452452

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	3725	1	817	0.219329
1	2835	26	2273	0.801764
2	3133	3	829	0.264603
3	3816	8	1134	0.29717
4	3181	7	731	0.229802
5	2614	6	1009	0.385998
6	2299	17	1066	0.46368
7	3751	8	1443	0.384697
8	5059	9	1738	0.343546
9	4056	6	1563	0.385355
10	4126	14	1154	0.27969
11	3380	15	1912	0.56568
12	3714	13	3313	0.89203
13	2418	1	881	0.364351
14	2638	3	1480	0.561031
15	2800	5	1933	0.690357
16	2380	18	1015	0.426471
17	3441	17	1024	0.297588
18	3069	19	2404	0.783317

19	4009	25	787	0.196308
20	4260	10	2049	0.480986
21	2948	11	1990	0.675034
22	5334	9	1661	0.311399
23	4419	23	959	0.217017
24	2640	14	1575	0.596591
25	2630	15	1733	0.658935
26	3564	16	1787	0.501403
27	3987	4	1855	0.465262
28	3066	18	1632	0.53229
29	3209	19	951	0.296354
30	4407	25	1267	0.287497
31	2717	21	1859	0.684211
32	2620	22	1685	0.64313
33	3665	23	2917	0.795907
34	3070	24	2259	0.735831
35	2479	22	1517	0.61194
36	3341	26	1048	0.313679
Total Set	124800	N/A	57250	0.458734

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 (%)
0	3876	1	767	0.197884
1	2847	26	2281	0.801194
2	3126	3	822	0.262956
3	3816	8	1115	0.292191
4	3160	7	730	0.231013
5	2652	6	1025	0.386501
6	2172	17	1031	0.474678
7	3747	8	1430	0.381639
8	5385	9	1883	0.349675

9	4324	6	1630	0.376966
10	4192	14	1178	0.281011
11	3441	15	1915	0.556524
12	3734	13	3319	0.888859
13	2413	1	880	0.364691
14	2644	3	1487	0.562405
15	2832	5	1944	0.686441
16	2683	18	979	0.36489
17	2958	17	894	0.302231
18	3038	19	2412	0.793943
19	4648	25	1171	0.251936
20	4269	10	2072	0.48536
21	2973	11	1995	0.671039
22	5340	9	1666	0.311985
23	4539	23	937	0.206433
24	2717	14	1641	0.603975
25	2605	15	1730	0.664107
26	3576	16	1831	0.512025
27	3970	4	1867	0.470277
28	3002	18	1658	0.552298
29	3152	19	916	0.290609
30	3229	25	1183	0.366367
31	2641	21	1851	0.700871
32	2734	22	1762	0.644477
33	3707	23	2940	0.793094
34	3077	24	2267	0.736757
35	2213	22	1412	0.638048
36	3368	26	1040	0.308789
Total Set	124800	N/A	57661	0.462027

So, the seed 3 have the best performance, I use the seed 3 in part d.

d.)

For the test files:

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

	to the cluster		images	
0	653	1	135	0.206738
1	446	26	357	0.800448
2	478	3	136	0.284519
3	632	8	200	0.316456
4	505	7	129	0.255446
5	453	6	161	0.355408
6	362	17	178	0.491713
7	598	8	232	0.38796
8	922	9	303	0.328633
9	716	6	258	0.360335
10	736	14	209	0.283967
11	556	15	314	0.564748
12	595	13	538	0.904202
13	420	1	156	0.371429
14	459	3	249	0.542484
15	448	5	304	0.678571
16	477	18	169	0.354298
17	490	17	143	0.291837
18	506	19	403	0.796443
19	788	25	197	0.25
20	724	10	337	0.46547
21	494	11	314	0.635628
22	871	9	292	0.335247
23	751	23	139	0.185087
24	446	14	277	0.621076
25	441	15	299	0.678005
26	601	16	311	0.517471
27	659	4	322	0.488619
28	471	18	265	0.562633
29	572	2	180	0.314685
30	537	25	197	0.366853
31	474	21	341	0.719409
32	487	22	307	0.63039
33	627	23	515	0.821372
34	511	24	365	0.714286

35	344	22	221	0.642442
36	550	26	165	0.3
Total Set	20800	N/A	9618	0.462403

Compare to the HW3 result, using this method have higher accuracy. More than that, I realize that the memory consumption of the program is much lesser because of each data's dimesion has been dectucted to 25 from 784. And the speed of map reduce and checking program is faster too.

Q3.)

a.)

i.) Item-item CF:

(2, 5) 0.2721655269759087
(2, 4) 0.17588161767036214
(2, 3) 0.09860132971832693
(2, 2) 1.0
(2, 1) -0.8571593913749194
(2, 0) -0.13608276348795437

So, set the k = 2. And the most similar users for item C (2) is 5 and 4. So the answer is 4.39.

ii)

User-User CF:

(1, 2) 0.23134069792952236
(1, 4) -0.35000000000000003
(1, 1) 1.0
(1, 0) 0.35355339059327373

So, set the k = 2. And the most similar users for user II (1) is 0 and 2. So the answer is 4.60.

b.)

[[1.60181186 1.10155771 4.70852404 4.02325968 3.46382091 3.24415151]
 [2.99738067 2.54936675 4.95571807 3.78009147 4.36990693 3.60615196]
 [4.60012762 4.47794519 3.13993082 1.45924377 4.26033711 2.67962585]
 [2.69458465 2.24724505 4.80723373 3.74062535 4.12135617 3.46697337]
 [3.9587518 4.01563034 1.42240116 0.01145553 2.96531543 1.48791261]]

The predict rate is 4.95571807, far from both result from part a.

c.)

The update rule is incorrect:

```
P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
```

This will not give the fastest descent direction.

The correct version:

```
p_tmp = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
q_tmp = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
P[i][k] = p_tmp
Q[k][j] = q_tmp
```

Predict:

[[1.6546088 1.34485331 4.41669398 4.27765114 3.41018211 3.43295484]
 [3.43277886 2.81838621 4.59216079 3.43970718 4.35709812 3.66860648]
 [4.85539918 4.00646431 3.24683611 1.00278926 4.23128113 2.73461114]
 [2.41277102 1.97116285 4.8092776 4.29820692 4.0028651 3.77352135]
 [4.5342318 3.7506164 1.54903192 -0.82689138 3.06958918 1.43321501]]

The result is 4.59216079, close to the result of part a.

Q4.)

a.)

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

k = 100
store = dict()
file = 'shakespeare-basket'

for line in open(file).readlines():
    words = line.strip().split()
    for word in words:
        if (not store.get(str(word))) and len(store) < k-1:
            store[str(word)] = 1
        elif store.get(str(word)):
            store[str(word)] += 1
        elif not store.get(str(word)):
            for ele in list(store.keys()):
                store[ele] -= 1
                if store[ele] <= 0:
                    del store[ele]
            # print("test")

store = dict(sorted(store.items(), key=lambda item: item[1]))

for ele, count in store.items():
    print(ele, count)

# print(len(store))
```

Q4_data_st_loc

```
pg 18
works 27
but 32
donations 32
for 44
any 84
work 102
tm 116
this 132
project 243
her 252
or 263
gutenberg 316
said 2048
you 2528
with 2955
as 4516
his 11767
by 11876
is 23153
it 26331
he 31653
that 35513
i 59556
was 59656
in 223821
a 261914
to 339629
and 439567
of 576275
the 1007946
```

DDO ▶ 4: Run 5: Debug Python Console Terminal

m 2020.1.5 available: // Update... (today 18:57)

b.)

If all distant items' frequency are blow n/k which in here refer the 1 % of total number of words, then the 1-pass algorithm will produce the wrong answer.

Or like the example in lecture slide p36, in this scenario the last one is in-correct classify as heavy hitter.

c.)

```
eyes    5
crab    1
that    35498
with    2941
sister  5
sourestnatured  1
lives   2
herself 1
he      31638
look    4
hour    21
this    117
work    87
cat     6
project 228
lucetta 235
and     439553
is      23138
in      223806
it      26317
states  1
howling 6
as      4502
troilus 590
seen    12
any     70
sir     372
write   37
you     2514
nay     31
mother  3
grandam 7
gutenberg       301
but     17
letter  1
parting 17
a       261900
thy     23
i       59541
dog     78
think   1
writ    129
enter   22
the     1007932
```

Mapper:

dic14.ie.cuhk.edu.hk - PuTTY

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

k = 100
store = dict()


for line in sys.stdin:
    words = line.strip().split()
    for word in words:
        if (not store.get(str(word))) and len(store) < k-1:
            store[str(word)] = 1
        elif store.get(str(word)):
            store[str(word)] += 1
        elif not store.get(str(word)):
            for ele in list(store.keys()):
                store[ele] -= 1
                if store[ele] <= 0:
                    del store[ele]

# store = dict(sorted(store.items(), key=lambda item: item[1]))

for ele, count in store.items():
    print('%s\t%s' % (ele, count))
```

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

k = 100
store = dict()

for line in sys.stdin:
    word, count = line.strip().split('\t')
    if store.get(str(word)):
        store[str(word)] += int(count)
    else:
        store[str(word)] = int(count)

if len(store) > k - 1:
    store_x = int(sorted(store.items(), key=operator.itemgetter(1))[k-1][1])
    # kth_ele = int(store[str(list(store.keys())[k])])
    for ele in list(store.keys()):
        store[ele] -= store_x
        if store[ele] <= 0:
            del store[ele]

store = dict(sorted(store.items(), key=lambda item: item[1]))
for ele, count in store.items():
    print('%s\t%s' % (ele, count))
```

Result:

```
windsor 31
how     7
sins    24
which   51
you     29037
woodman 18
shall   147
why     1
enter   67
beast   16
falstaff        1192
gutenberg       308
sir     3321
thunder 1
sides   14
a       263028
off     6
for     8469
i       74062
jove    53
doe     14
green   2
bribe   9
sleeves 3
the     1009060
~
```