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/.

Signed (Student Cao Yuhang) Date: 04/22/2018
Name Cao Yuhang SID 1155092180

Uncompress:

Convert original data to “csv” format. Each row is a 784 dimension data, each column is a pixel feature. And use PCA to reduct 784 dimension to 25 dimension. (I use PCA package of sklearn)

```python
import os
import struct
import numpy as np
from sklearn.decomposition import PCA


def load_mnist(path, kind='train'):
    if kind == 'train':
        labels_path = './ori_data/train-labels-idx1-ubyte'
        images_path = './ori_data/train-images-idx3-ubyte'
    elif kind == 'test':
        images_path = './ori_data/t10k-images-idx3-ubyte'
        labels_path = './ori_data/t10k-labels-idx1-ubyte'

    with open(labels_path, 'rb') as lbpath:
        magic, n = struct.unpack('>II', lbpath.read(8))
        labels = np.fromfile(lbpath, dtype=np.uint8)

    with open(images_path, 'rb') as imgpath:
        magic, num, rows, cols = struct.unpack('>IIII', imgpath.read(16))
        images = np.fromfile(imgpath, dtype=np.uint8).reshape(len(labels), 784)

    return images, labels


pca = PCA(n_components=25)
X_train, y_train = load_mnist("./")
X_train = pca.fit_transform(X_train)
X_test, y_test = load_mnist("./", kind='test')
X_test = pca.transform(X_test)

np.savetxt('data/train_img.csv', X_train, fmt='%f', delimiter=',')
np.savetxt('data/train_labels.csv', y_train, fmt='%f', delimiter=',')
np.savetxt('data/test_img.csv', X_test, fmt='%f', delimiter=',')
np.savetxt('data/test_labels.csv', y_test, fmt='%f', delimiter=',')
```

Get Initial Centroids:

1. Choose first centroid randomly.
2. Select next initial centroids in such a way that the Euclidean distance of that point is maximum from other selected initial centroids.
3. Repeat step 2 until we get k initial centroids.
4. The program will ensure initial points contain at least 7 different labels.

The result is stored in “cluster_id \t centroid \t cluster_number” format, cluster_number represents how many images in this cluster, at beginning, this is just a dump number, just for the consistent in later processing.

```python
import numpy as np


np.random.seed(0)

X_train = np.genfromtxt('../data/train_img.csv', dtype=float, delimiter=',')
X_label = np.genfromtxt('../data/train_labels.csv', dtype=float, delimiter=',')

N, D = X_train.shape
K = 10

while True:
    # initialize
    initial_centriods = np.zeros((K, D))
    labels = np.zeros(K)
    current_indexes = []
    current_centriods = []

    index = np.random.randint(X_train.shape[0], size=1)[0]  # initial point 1

    current_indexes.append(index)
    current_centriods.append(X_train[index])
    labels[0] = X_label[index]
    print ("{}th initial point index: {}, distance is: {}, \
            label is: {}".format(0, index, 0, X_label[index]))

    for i in range(9):
        max_distance = -np.inf
        max_index = None
        for j in range(X_train.shape[0]):
            distance = 0
            for k in range(len(current_centriods)):
                distance += np.sum((X_train[j] - current_centriods[k])**2)
            if distance > max_distance and j not in current_indexes:
                max_distance = distance
                max_index = j
        print ("{}th initial point index: {}, distance is: {}, \
                label is: {}".format(i+1, max_index, max_distance / len(
                    current_centriods), X_label[max_index]))
        current_indexes.append(max_index)
        current_centriods.append(X_train[max_index])
        labels[i+1] = X_label[max_index]

    for i in range(10):
        initial_centriods[i] = current_centriods[i]

    print np.unique(labels)
    if len(np.unique(labels)) >= 7:
        break

with open('initial_point.txt', 'w') as f:
    for i in range(10):
        centroid = initial_centriods[i]
        centroid = ','.join([str(v) for v in centroid])
        f.write("{}\t{}\t{}\n".format(i, centroid, 0))
```

Q1 (a):

Mapper:
input: split of data
output: “cluster_id \t partial sum of this cluster \t image number of this cluster”

Reducer:
Merge partial sum and image number of same cluster_id, divided sum by number to get mean of this cluster_id

Code and Result:

- mapper1. py: mapper
- reducer1.py: reducer
- format_output.py: get desire output format
- cal_error.py: calculate error between consecutive iterations
- run.sh: run the program in an iterative manner
- ori_centroid_points.txt, old_centroid_points.txt, new_centroid_points.txt: tmp files
- res_a: result of part_a

Code:
Mapper:

```
#!/usr/bin/env python
import sys
import numpy as np


def zero():
    return 0


def main():

    # read cluster points
    centroids = np.zeros((10, 25))
    with open('./old_centroid_points.txt', 'r') as f:
        i = 0
        for line in f:
            index, centroid, counts = line.strip().split("\t")
            index = int(float(index))
            centroid = np.fromstring(centroid, sep=',')
            centroids[index] = centroid
            i += 1

    local_centroid = np.zeros((10, 25))
    local_centroid_counts = np.zeros((10, 1))

    for line in sys.stdin:
        p = np.fromstring(line, sep=',')
        distances = np.sum((centroids - p) ** 2, axis=1)
        min_index = np.argmin(distances)

        local_centroid[min_index] += p
        local_centroid_counts[min_index] += 1

    for i in range(len(local_centroid_counts)):
        centroid = ','.join([str(v) for v in local_centroid[i]])
        print str(i) + '\t' + centroid +  '\t' + str(local_centroid_counts[i, 0])


if __name__ == '__main__':
    main()
```

Reducer:

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

import sys
import numpy as np


def main():
    current_centroid_index = None
    current_counts = 0.

    for line in sys.stdin:
        centroid_index, local_centroid, local_counts = line.strip().split("\t")

        centroid_index = int(centroid_index)
        local_counts = float(local_counts)
        local_centroid = np.fromstring(local_centroid, sep=',')

        if current_centroid_index is None:
            current_centroid_index = centroid_index
            current_counts = local_counts
            current_centroid = local_centroid
        elif centroid_index != current_centroid_index:
            if current_counts != 0:
                current_centroid = current_centroid / current_counts
            p = ','.join([str(v) for v in current_centroid])
            print "{}\t{}\t{}".format(current_centroid_index, p, current_counts)

            current_counts = local_counts
            current_centroid_index = centroid_index
            current_centroid = local_centroid
        else:
            current_centroid += local_centroid
            current_counts += local_counts

    if current_counts != 0:
        current_centroid = current_centroid / current_counts
    p = ','.join([str(v) for v in current_centroid])
    print "{}\t{}\t{}".format(current_centroid_index, p, current_counts)

if __name__ == "__main__":
    main()
```

run.sh:

```
hadoop fs -rm -r output

input='data/train_img.csv'
output="output"

m=4
r=2

error=999999
i=0
cp ./ori_centroid_points.txt ./old_centroid_points.txt
cp ./ori_centroid_points.txt ./new_centroid_points.txt
rm error.txt

while [ $error -gt 1000 ]; do
# while [ $i -lt 20 ]; do
    hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar \
        -D mapred.map.tasks=$m \
        -D mapred.reduce.tasks=$r \
        -D mapred.compress.map.output=ture \
        -file mapper1.py -mapper mapper1.py  \
        -file reducer1.py -reducer reducer1.py  \
        -file old_centroid_points.txt \
        -input $input \
        -output output \

    rm new_centroid_points.txt
    hadoop fs -get output ./
    hadoop fs -rm -r output
    cat output/* >> ./new_centroid_points.txt
    error="$(python cal_error.py)"
    echo "$iteration ${i}, error is ${error}" >> error.txt
    cp new_centroid_points.txt old_centroid_points.txt

    rm -rf output
    i=$(( i + 1 ))
done
```

Result:

Centroid 0:
[39.098,-202.311,86.814,-509.701,328.529,57.122,-118.289,-170.412,-141.802,146.666,44.002,-0.444,142.038,-7.217,-18.595,21.43,46.062,34.573,2.671,25.999,-7.996,-36.65,-71.651,20.985,14.178], 6381.0
Centroid 1:
[1001.953,-145.828,-30.583,-140.166,-203.167,-188.409,-13.516,-137.782,-128.233,-82.745,-260.335,-260.696,-99.179,-200.734,-303.228,-54.648,-75.731,103.984,-89.466,-91.555,6.927,54.482,98.558,40.438,-12.722], 1899.0
Centroid 2:
[-728.741,-342.396,84.25,87.61,-212.078,3.882,29.828,-78.454,85.055,57.866,20.667,40.185,23.894,27.256,-87.069,-7.772,-31.282,-3.056,-31.904,-14.565,-5.359,10.608,39.571,-3.212,0.932], 10811.0
Centroid 3:
[208.513,-451.691,-663.203,-142.049,116.269,-83.284,-51.154,115.844,32.3,-224.471,95.342,-41.478,-22.331,1.806,47.806,22.654,-58.104,-15.879,13.804,43.004,10.236,7.217,-16.707,-41.664,-33.641], 7030.0
Centroid 4:
[181.043,-312.186,330.151,444.678,333.283,362.061,81.247,229.695,-242.15,136.705,53.821,28.084,-121.201,-65.784,128.796,1.581,-47.235,-13.808,-6.587,-51.998,15.257,45.316,63.174,-32.752,-46.824], 4677.0
Centroid 5:
[-25.763,617.166,-343.7,157.581,27.156,-19.613,-186.878,127.584,82.277,199.264,-35.616,-7.8,10.359,22.717,-14.812,-15.475,-14.496,-52.154,-14.55,-12.648,45.469,-9.675,19.684,18.468,46.067], 9306.0
Centroid 6:
[1117.632,-238.883,-199.075,77.391,-764.98,391.652,181.496,-312.997,76.624,185.866,-40.765,145.35,-26.054,-57.58,96.035,19.269,152.765,-48.261,-6.48,-54.034,-69.783,-32.562,-132.53,-31.117,48.274], 2558.0
Centroid 7:
[-259.708,565.308,224.865,-283.488,-37.417,168.487,134.155,-34.995,-37.841,-265.069,-12.737,4.268,-64.959,-23.227,23.124,13.92,26.796,1.048,39.631,-8.488,-30.238,35.495,-34.916,-2.827,-14.41], 8898.0
Centroid 8:
[666.067,-164.625,697.475,-294.534,-345.818,-232.316,-111.582,294.202,212.304,70.262,7.858,71.844,-71.377,81.464,98.495,31.854,-51.657,43.813,-25.795,46.164,22.84,-27.15,73.936,21.708,9.031], 2907.0
Centroid 9:
[211.3,39.973,251.362,579.169,225.634,-506.005,152.375,-112.409,-19.489,-115.874,-73.306,-58.259,90.957,60.237,17.611,-42.894,85.547,47.582,55.28,54.861,-16.171,-56.125,-15.214,25.283,-12.851], 5533.0

Q1 (b):

code:

get_label.py: calculate the accuracy

```python
import numpy as np


centroids = np.zeros((10, 25))

i = 0
with open('./centroid_points.txt', 'r') as f:
    for line in f:
        centroid = np.fromstring(line.strip().split("\t")[1], sep=',')
        centroids[i] = centroid

        i += 1

predictions = [[] for i in range(10)]

with open('../data/test_img.csv', 'r') as f:
    i = 0
    for line in f:
        p = np.fromstring(line, sep=',')
        distances = np.sum((centroids - p) ** 2, axis=1)
        min_index = np.argmin(distances)
        min_distances = distances[min_index]
        predictions[min_index].append((i, min_distances))
        i += 1


nums = np.zeros((10, 1))
for i in range(10):
    predictions[i] = sorted(predictions[i], key=lambda x: x[1])
    nums[i] = len(predictions[i])


labels = np.genfromtxt('../data/test_labels.csv', delimiter=',')

for s in [0.05, 0.1, 0.5, 1]:
    print("threshold is: {}".format(s))

    correctly_clustered_images = np.zeros(10)
    accuracies = np.zeros(10)

    res = np.zeros(10000)

    for i in range(10):
        tmp = np.zeros(10)
        threshold = s * nums[i]
        above = False

        for p in predictions[i]:
            index = p[0]
            label = int(labels[index])
            tmp[label] += 1
            if tmp[label] >= threshold:
                above = True

                # calculate accuracy
                correct_num = 0
                for pair in predictions[i]:
                    j = pair[0]
                    res[j] = label
                    if res[j] == labels[j]:
                        correct_num += 1
                accuracy = correct_num / float(len(predictions[i]))
                print "cluster: {}, num: {}, threshold: {}, true label: {}, correctly_clustered_images: {}, accuracy: {}".format(
                    i, nums[i], threshold, label, nums[i]*accuracy, np.round(accuracy, 3))
                correctly_clustered_images[i] = nums[i]*accuracy
                accuracies[i] = accuracy
                break

        if not above:
            label = np.argmax(tmp)

            # calculate accuracy
            correct_num = 0

            for pair in predictions[i]:
                j = pair[0]
                res[j] = label
                if res[j] == labels[j]:
                    correct_num += 1

            accuracy = correct_num / float(len(predictions[i]))

            print "cluster id: {}, num: {}, threshold: {}, true label: {}, correctly_clustered_images: {}, accuracy: {}".format(
                    i, nums[i], threshold, label, nums[i]*accuracy, np.round(accuracy, 3))

            correctly_clustered_images[i] = nums[i]*accuracy
            accuracies[i] = accuracy

    print "correctly_clustered_images: {}".format(np.sum(correctly_clustered_images))
    print "overall accuracy is: {}".format(np.sum(res==labels) / float(len(labels)))
    print
```

Table. 1.	The Accuracy	of Clustering	Performance	with	x = 5%
Cluster Number	# images in the entire cluster	# of images considered (m) when determining the cluster label	Major Label of central images	# correctly clustered images	Classification Accuracy (%)
0	381	19	0	322	0.845
1	1220	61	3	694	0.569
2	1595	80	9	558	0.35
3	1436	72	9	376	0.262
4	905	45	6	785	0.867
5	1052	53	8	589	0.56
6	1710	85	1	1104	0.646
7	816	41	2	707	0.866
8	410	20	0	369	0.9
9	475	24	0	259	0.545
Total Set	10000	500	NA	5763	0.5763

Table. 2.	The Accuracy	of Clustering	Performance	with	x = 10%
Cluster Number	# images in the entire cluster	# of images considered (m) when determining the cluster label	Major Label of central images	# correctly clustered images	Classification Accuracy (%)
0	381	38	0	322	0.845
1	1220	122	3	694	0.569
2	1595	160	9	558	0.35
3	1436	144	7	604	0.421
4	905	90	6	785	0.867
5	1052	105	8	589	0.56
6	1710	171	1	1104	0.646
7	816	82	2	707	0.866
8	410	41	0	369	0.9
9	475	47	0	259	0.545
Total Set	10000	1000	NA	5991	0.5991

Table. 3.	The Accuracy	of Clustering	Performance	with	x = 50%
Cluster Number	# images in the entire cluster	# of images considered (m) when determining the cluster label	Major Label of central images	# correctly clustered images	Classification Accuracy (%)
0	381	190	0	322	0.845
1	1220	610	3	694	0.569
2	1595	798	4	561	0.352
3	1436	718	7	604	0.421
4	905	452	6	785	0.867
5	1052	526	8	589	0.56
6	1710	855	1	1104	0.646
7	816	408	2	707	0.866
8	410	205	0	369	0.9
9	475	238	0	259	0.545
Total Set	10000	5000	NA	5994	0.5994

Table. 4.	The Accuracy	of Clustering	Performance	with	x = 100%
Cluster Number	# images in the entire cluster	# of images considered (m) when determining the cluster label	Major Label of central images	# correctly clustered images	Classification Accuracy (%)
0	381	381	0	322	0.845
1	1220	1220	3	694	0.569
2	1595	1595	4	561	0.352
3	1436	1436	7	604	0.421
4	905	905	6	785	0.867
5	1052	1052	8	589	0.56
6	1710	1710	1	1104	0.646
7	816	816	2	707	0.866
8	410	410	0	369	0.9
9	475	475	0	259	0.545
Total Set	10000	10000	NA	5994	0.5994

Best x: 50% and 100%

Explain:

Firstly observe that as x increases, accuracy also increases, if x is small, we rely more on the points that has smaller distance, in some sense we can reduce the noise (points that far away from centroid) influence. However, why accuracy keep decreasing? In fact this is the phenomenon of "curse of dimensionality", as dimension increase, the data points will concentrated more and more in a thin-shell near the surface. Let's look at cluster 3, in cluster 5%, we can find the true label is 9, but the #correctly cluster image is only 376; in cluster 10%, we can find the true label is 7, and the #correctly cluster image is 604, although label 9 is more similar to centroid, in the cluster 3 it is minor; although label 7 is far away from centroid, in cluster 3 it is majority, this is why accuracy increase.

Q1 (c):

- mapper1. py: mapper
- reducer1.py: reducer
- get_data.py: get desire data folders
- cal_error.py: calculate error between consecutive iterations
- run.sh: run the program in an iterative manner
- ori_centroid_points.txt, old_centroid_points.txt, new_centroid_points.txt: tmp files
- res: result of part 5 folders

Notice: (5_folder and 10_folder are almost same, just modify 5 to 10)

mapper and reducer are same with Q1 (a)

get_data.py: split total data (mixture with training and testing data) into 5 folder, loop over 5 folder, every time choose 1 folder as test set and 4 for train set

```python
import numpy as np
import os


np.random.seed(0)

X_train = np.genfromtxt('../data/mixture_img.csv', delimiter=',')
X_label = np.genfromtxt('../data/mixture_labels.csv', delimiter=',')

n_folder = 5
interval = X_train.shape[0] / n_folder

# shuffle
indices = np.random.permutation(X_train.shape[0])
X_train = X_train[indices]
X_label = X_label[indices]

if os.path.isdir('5_folder_input'):
    os.system("rm -rf 5_folder_input")
os.system("mkdir 5_folder_input")

# split into n folders
for i in range(n_folder):
    # test folder
    test_folder_num = i
    test_indices = range(test_folder_num * interval,
                         (test_folder_num + 1) * interval)
    test_image = X_train[test_indices]
    test_label = X_label[test_indices]

    # train folder
    train_image = np.delete(X_train, test_indices, axis=0)
    train_label = np.delete(X_label, test_indices)

    if os.path.isdir('5_folder_input/iter_{}'.format(i)):
        os.system("rm -rf 5_folder_input/iter_{}".format(i))
    os.mkdir('5_folder_input/iter_{}'.format(i))

    np.savetxt('5_folder_input/iter_{}/train_image.csv'.format(i),
               train_image, fmt='%f', delimiter=',')
    np.savetxt('5_folder_input/iter_{}/train_label.csv'.format(i),
               train_label, fmt='%f', delimiter=',')
    np.savetxt('5_folder_input/iter_{}/test_image.csv'.format(i),
               test_image, fmt='%f', delimiter=',')
    np.savetxt('5_folder_input/iter_{}/test_label.csv'.format(i),
               test_label, fmt='%f', delimiter=',')
```

run.sh: run 5 experiments for the corresponding 5 dataset (different test and train folder combination)

```
m=4
r=2

n=0

while [ $n -lt 5 ]; do
    hadoop fs -rm -r output
    rm -rf output
    input="5_folder_input/iter_${n}/train_image.csv"
    output="output"
    cp ./ori_centroid_points.txt ./old_centroid_points.txt
    cp ./ori_centroid_points.txt ./new_centroid_points.txt
    error=999999
    rm -rf "iter_${n}_error.txt"
    i=0

    while [ $error -gt 1000 ]; do
        hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar \
            -D mapred.map.tasks=$m \
            -D mapred.reduce.tasks=$r \
            -D mapred.compress.map.output=ture \
            -file mapper1.py -mapper mapper1.py  \
            -file reducer1.py -reducer reducer1.py  \
            -file old_centroid_points.txt \
            -input $input \
            -output output \

        rm new_centroid_points.txt
        hadoop fs -get output ./
        hadoop fs -rm -r output
        cat output/* >> ./new_centroid_points.txt
        error="$(python cal_error.py)"
        echo "$iteration ${i}, error is ${error}" >> "iter_${n}_error.txt"
        cp new_centroid_points.txt old_centroid_points.txt

        rm -rf output
        i=$(( i + 1 ))
    done
    cp new_centroid_points.txt "iter_${n}_centroid_points.txt"
    n=$(( n + 1 ))
done
```

The result is tested on x=100%, which is best x in part (b)

5-folder results:

Testing set	Classification Accuracy
Part 1	0.5931
Part 2	0.5898
Part 3	0.597
Part 4	0.594
Part 5	0.5894
Average	0.59266

10-folder results:

Testing set	Classification Accuracy
Part 1	0.5957
Part 2	0.5893
Part 3	0.5983
Part 4	0.5893
Part 5	0.5947
Part 6	0.6034
Part 7	0.598
Part 8	0.5901
Part 9	0.5849
Part 10	0.5953
Average	0.5939

Comparison between with PCA and without PCA:

Compared with without PCA in homework3, the accuracy is almost the same (all about 0.59~0.6), just lower a little. PCA represents each 28x28-pixel image of handwritten digit as the linear combination of 25 principal “eigen-digits”, it loses some information, this maybe the reason why accuracy decrease a little, but the performance is greatly improved.

Q1 (b):

Code files

- mapper1. py: mapper1
- reducer1.py: reducer1
- mapper2. py: mapper1
- reducer2.py: reducer1
- get_initial_parameters.py
- run.sh: run the program in an iterative manner
- res: result of part 5 folders

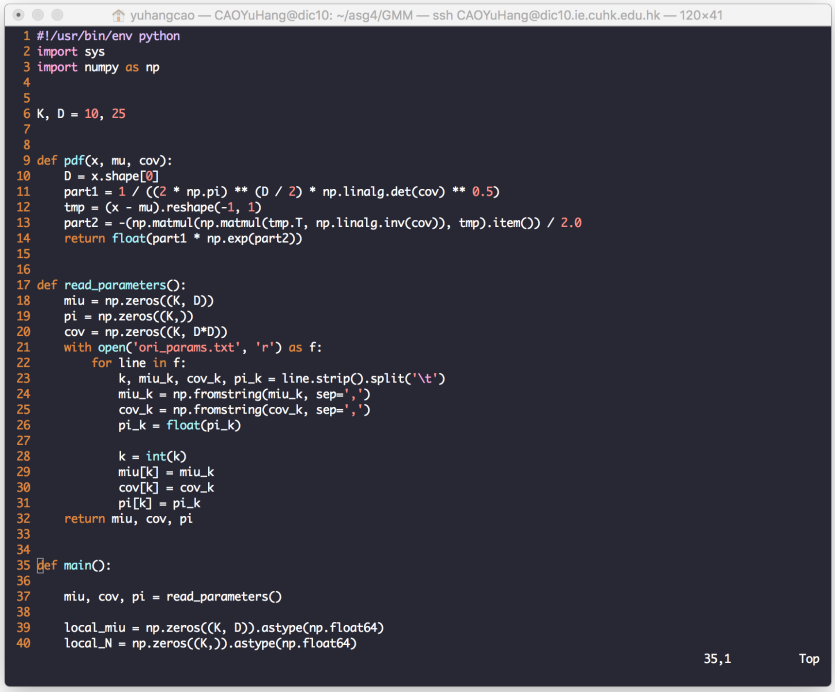
get_initial_parameters.py:

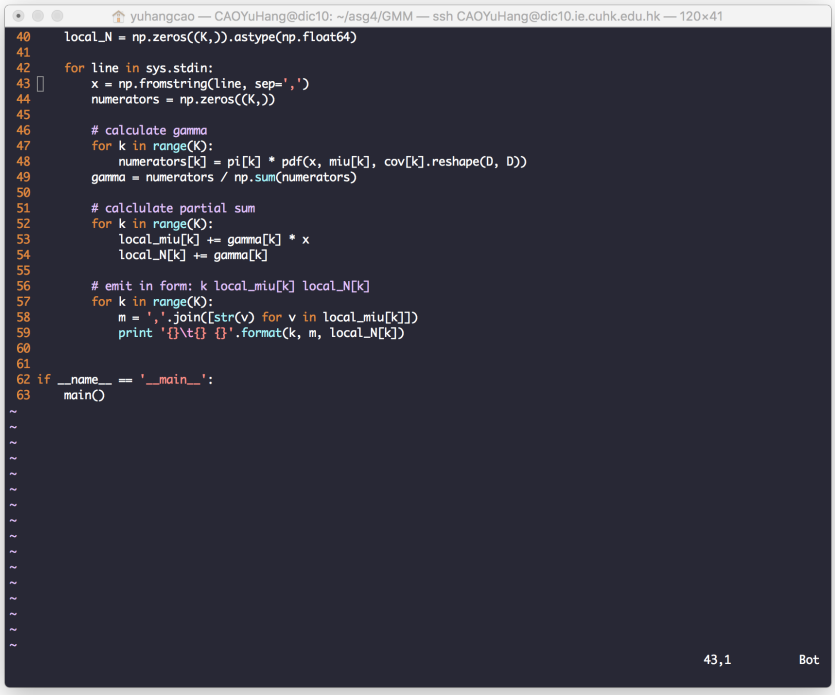
- get initial parameter, I just compute the mean, cov of original data, pi is all initialized to be 0.1

```
import numpy as np


K, D = 10, 25

X = np.genfromtxt('./data/train_img.csv', delimiter=',')
N = X.shape[0]
interval = N / K

tmp = np.mean(X, axis=0)
mu = np.zeros((K, D))
cov = np.zeros((K, D*D))

for k in range(K):
    x = X[k * interval: (k+1) * interval]
    mu[k] = np.mean(x, axis=0).reshape(-1)
    cov[k] = np.cov(x.T).reshape(-1)

pi = np.ones(10,)
pi = pi / np.sum(pi)

with open('ori_params.txt', 'w') as f:
    for k in range(K):
        m = ','.join([str(v) for v in mu[k]])
        c = ','.join([str(v) for v in cov[k]])
        f.write('{}\t{}\t{}\t{}\n'.format(k, m, c, pi[k]))
~
~
```

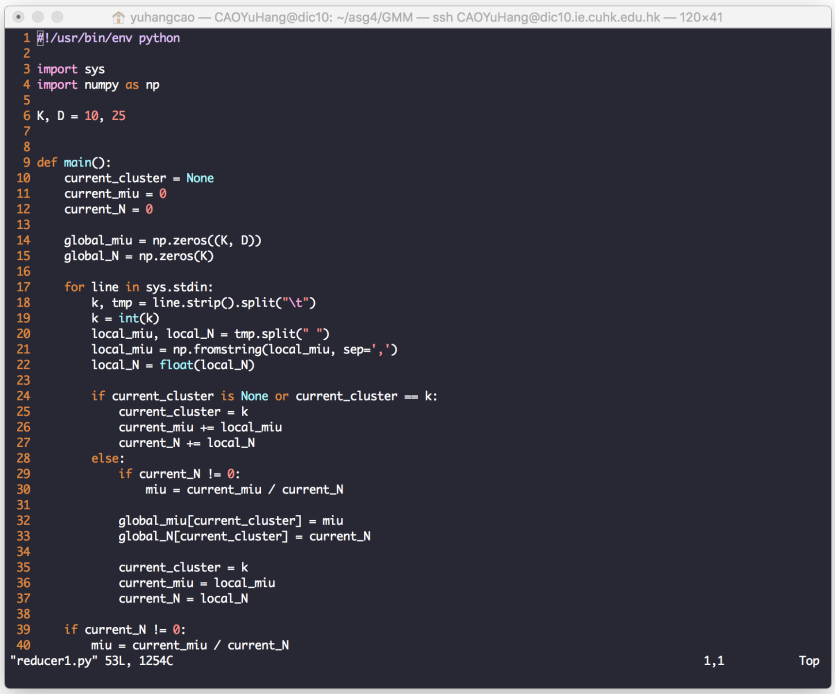
First job is to calculate μ_{new}

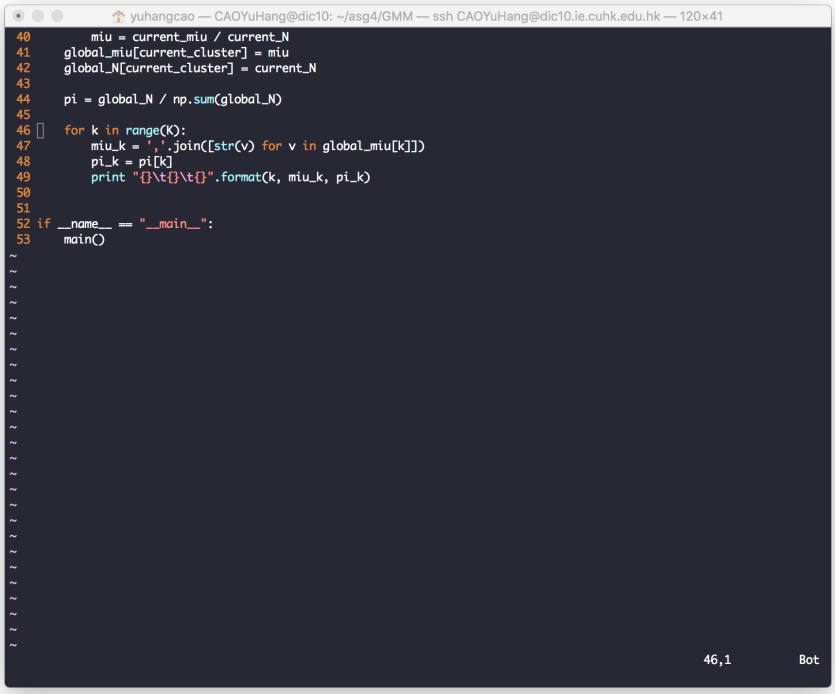
mapper1.py:





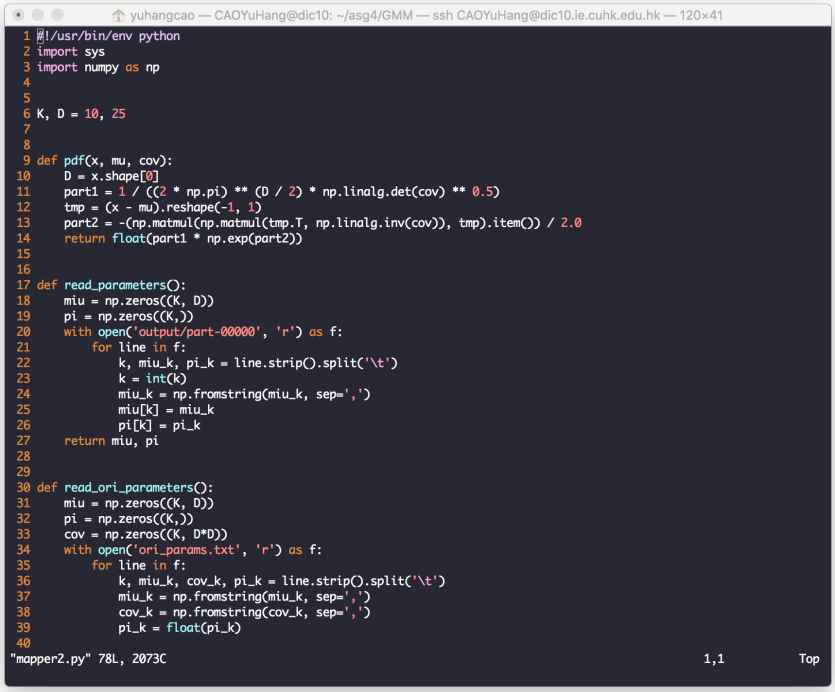
reducer1.py:

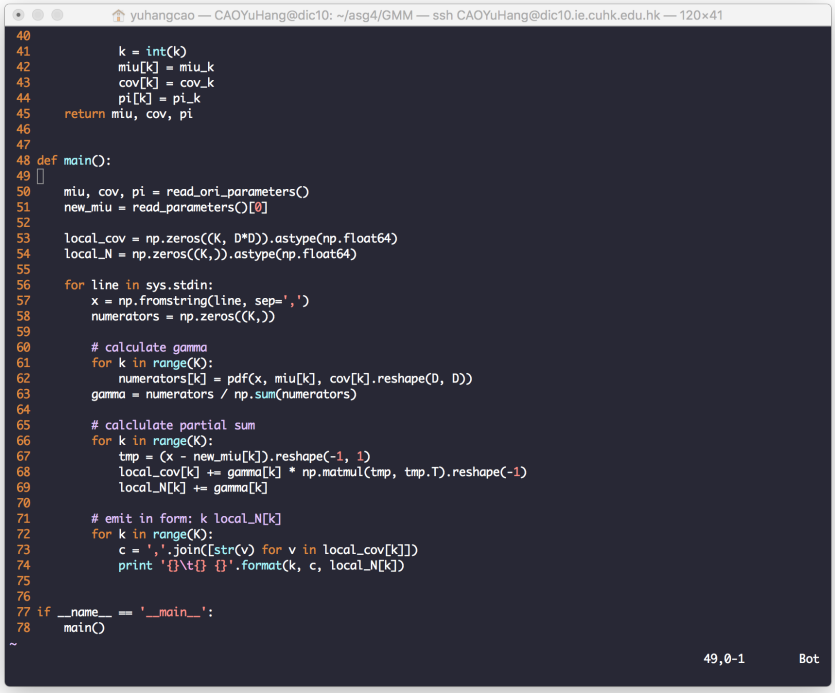

yuhangcao — CAOYuHang@dic10: ~/asg4/GMM — ssh CAOYuHang@dic10.ie.cuhk.edu.hk — 120×41
#!/usr/bin/env python
import sys
import numpy as np
K, D = 10, 25
def main():
current_cluster = None
current_miu = 0
current_N = 0
global_miu = np.zeros((K, D))
global_N = np.zeros(K)
for line in sys.stdin:
k, tmp = line.strip().split("\t")
k = int(k)
local_miu, local_N = tmp.split(" ")
local_miu = np.fromstring(local_miu, sep=',')
local_N = float(local_N)
if current_cluster is None or current_cluster == k:
current_cluster = k
current_miu += local_miu
current_N += local_N
else:
if current_N != 0:
miu = current_miu / current_N
global_miu[current_cluster] = miu
global_N[current_cluster] = current_N
current_cluster = k
current_miu = local_miu
current_N = local_N
if current_N != 0:
miu = current_miu / current_N
"reducer1.py" 53L, 1254C
1,1
Top


yuhangcao — CAOYuHang@dic10: ~/asg4/GMM — ssh CAOYuHang@dic10.ie.cuhk.edu.hk — 120×41
miu = current_miu / current_N
global_miu[current_cluster] = miu
global_N[current_cluster] = current_N
pi = global_N / np.sum(global_N)
for k in range(K):
miu_k = ','.join([str(v) for v in global_miu[k]])
pi_k = pi[k]
print "{}\t{}\t{}".format(k, miu_k, pi_k)
if __name__ == "__main__":
main()
46,1
Bot

second job is used to calculate cov and pi

mapper2.py:





reducer2.py:

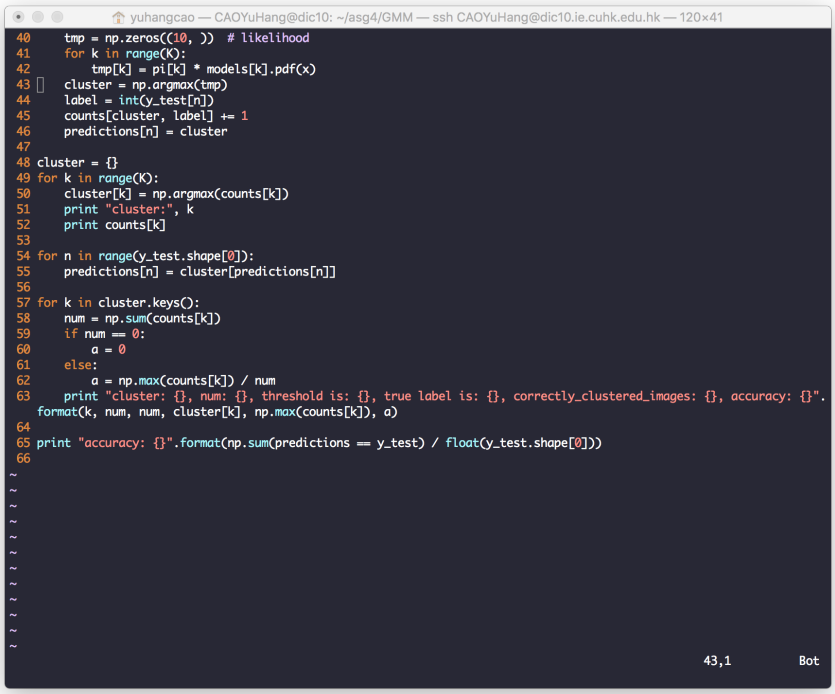
yuhangcao — CAOYuHang@dic10: ~/asg4/GMM — ssh CAOYuHang@dic10.ie.cuhk.edu.hk — 120×41

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

import sys
import numpy as np

K, D = 10, 25


def read_parameters():
    miu = np.zeros((K, D))
    pi = np.zeros((K,))
    with open('output/part-00000', 'r') as f:
        for line in f:
            k, miu_k, pi_k = line.strip().split('\t')
            k = int(k)
            miu_k = np.fromstring(miu_k, sep=',')
            miu[k] = miu_k
            pi[k] = pi_k
    return miu, pi


def main():
    current_cluster = None
    current_cov = 0
    current_N = 0

    global_miu, pi = read_parameters()
    global_N = np.zeros(K)
    global_cov = np.zeros((K, D*D))

    for line in sys.stdin:
        k, tmp = line.strip().split("\t")
        local_cov, local_N = tmp.split(" ")

        k = int(k)
        local_cov = np.fromstring(local_cov, sep=',')
        local_N = float(local_N)

        if current_cluster is None or current_cluster == k:
            current_cluster = k
"reducer2.py" 68L, 1716C                                                1,1           Top
```

yuhangcao — CAOYuHang@dic10: ~/asg4/GMM — ssh CAOYuHang@dic10.ie.cuhk.edu.hk — 120×41

```
            current_cluster = k
            current_cov += local_cov
            current_N += local_N
        else:
            if current_N != 0:
                current_cov = current_cov / current_N

            global_cov[current_cluster] = current_cov
            global_N[current_cluster] = current_N

            current_cluster = k
            current_cov = local_cov
            current_N = local_N

    if current_N != 0:
        current_cov = current_cov / current_N

    global_cov[current_cluster] = current_cov
    global_N[current_cluster] = current_N

    for k in range(K):
        miu = ','.join([str(v) for v in global_miu[k]])
        cov = ','.join([str(v) for v in global_cov[k]])
        pi = global_N[k] / np.sum(global_N)
        print "{}\t{}\t{}\t{}".format(k, miu, cov, pi)


if __name__ == "__main__":
    main()
~
~
~
~
~
~
~
~
~
~
                                                                        43,1          Bot
```

get_label.py: calculate accuracy, determine label using majority method



Result after training for 20 iterations using GMM:

Table. 1.	**The Accuracy**	**of Clustering**	**Performance**	**with**	**x = 100%**
Cluster Number	# images in the entire cluster	# of images considered (m) when determining the cluster label	Major Label of central images	# correctly clustered images	Classification Accuracy (%)
0	941	941	2	415	0.441
1	1261	1261	4	438	0.347
2	1186	1186	8	734	0.619
3	905	905	0	652	0.720
4	768	768	6	763	0.993
5	986	986	1	984	0.998
6	601	601	5	517	0.860
7	1197	1197	4	464	0.388
8	1160	1160	3	775	0.668
9	995	995	7	504	0.507
Total Set	10000	10000	NA	6246	0.6246

Q2

(a)

Additional parameters:
When predict the desired rating, using the top 2 most similar item

How to handle negative effect of missing ratings:
For a movie m, if missing rating of user u, then fill r(m, u) with the average rating of movie m.

1. Calculate mean of each Movie

	User 1	User 2	User 3	User 4	User 5	Mean
Movie A	1		6	2	4	3.25
Movie B	1	3		1	4	2.25
Movie C	6			4	2	4
Movie D	4	4	2	5		3.75
Movie E	4	5	4	0	3	4
Movie F		4	4	5	1	3.5

2. Normalize each movie by its mean, (filling missing rating with its average, so after subtract the mean, the missing rating is 0)

	User 1	User 2	User 3	User 4	User 5
Movie A	-2.25	0	2.75	-1.25	0.75
Movie B	-1.25	0.75	0	-1.25	1.75
Movie C	2	0	0	0	-2
Movie D	0.25	0.25	-1.75	1.25	0
Movie E	0	1	0	0	-1
Movie F	0	0.5	0.5	1.5	-2.5

3. Calculate sim (C, m) by formula $sim(x,y)=\cos(r_x,r_y)=\frac{r_x \bullet r_y}{||r_x|| \bullet ||r_y||}$

	User 1	User 2	User 3	User 4	User 5	sim (C, m)
Movie A	-2.25	0	2.75	-1.25	0.75	-0.55
Movie B	-1.25	0.75	0	-1.25	1.75	-0.82
Movie C	2	0	0	0	-2	1.0
Movie D	0.25	0.25	-1.75	1.25	0	0.08
Movie E	0	1	0	0	-1	0.5
Movie F	0	0.5	0.5	1.5	-2.5	0.59

4. Calculate predict rating according top 2 similar movie using formula $r_{xi}=\frac{\sum_{y \in N} s_{xy} \bullet r_{yi}}{\sum_{y \in N} s_{xy}}$

The top 2 similar Movie of C is E and F, so

$$r_{C2}=\frac{sim_{CE} \times r_{E2}+sim_{CF} \times r_{F2}}{sim_{CE}+sim_{CF}}=\frac{0.5 \times 5+0.59 \times 4}{0.5+0.59}=4.46$$

(b)

Additional parameters:
When predict the desired rating, using the top 2 most similar item

1. Calculate mean of each User

	Movie A	Movie B	Movie C	Movie D	Movie E	Movie F	Mean
User 1	1	1	6	4	4		3.2
User 2		3		4	5	4	4
User 3	6			2	4	4	4
User 4	2	1	4	5		5	3.4
User 5	4	4	2		3	1	2.8

2. Normalize each movie by its mean, (filling missing rating with its average, so after subtract the mean, the missing rating is 0)

	Movie A	Movie B	Movie C	Movie D	Movie E	Movie F
User 1	-2.2	-2.2	2.8	0.8	0.8	0
User 2	0	-1	0	0	1	0
User 3	2	0	0	-2	0	0
User 4	-1.4	-2.4	0.6	1.6	0	1.6
User 5	1.2	1.2	-0.8	0	0.2	-1.8

3. Calculate sim (2, m) by formula $sim(x,y)=\cos(r_x,r_y)=\frac{r_x \bullet r_y}{\|r_x\| \bullet \|r_y\|}$

	Movie A	Movie B	Movie C	Movie D	Movie E	Movie F	sim (2, m)
User 1	-2.2	-2.2	2.8	0.8	0.8	0	0.49
User 2	0	-1	0	0	1	0	1
User 3	2	0	0	-2	0	0	0
User 4	-1.4	-2.4	0.6	1.6	0	1.6	0.47
User 5	1.2	1.2	-0.8	0	0.2	-1.8	-0.27

4. Calculate predict rating according top 2 similar user using formula $r_{xi}=\frac{\sum_{y \in N} s_{xy} \bullet r_{yi}}{\sum_{y \in N} s_{xy}}$

The top 2 similar User of User2 is User1 and User4, so

$$r_{C2}=\frac{sim_{21} \times r_{1C}+sim_{24} \times r_{4C}}{sim_{21}+sim_{24}}=\frac{0.49 \times 6+0.47 \times 4}{0.49+0.47}=5.02$$

(c)

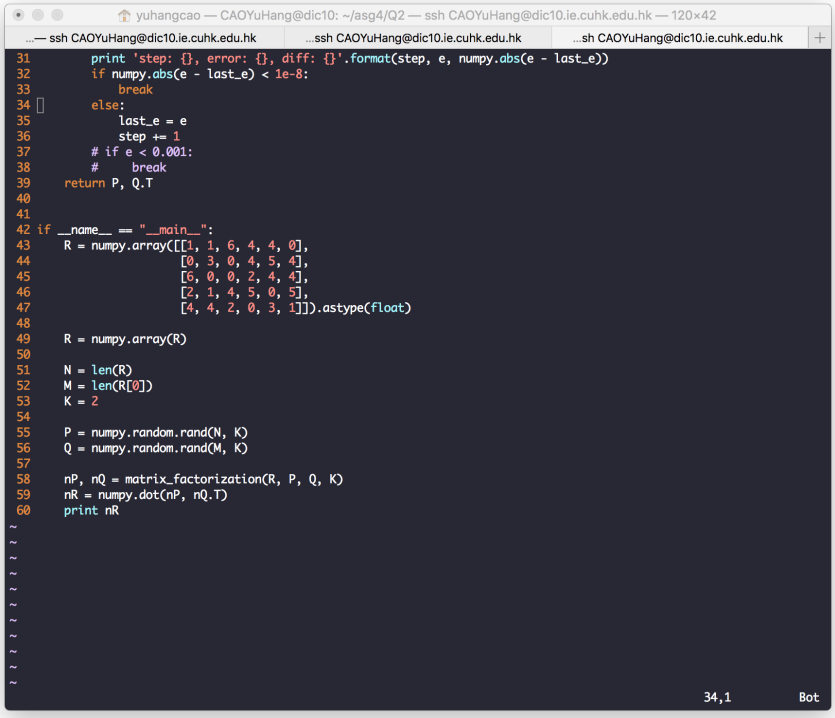
Code:

- new converge condition, error change is smaller than 1e-8
- Prediction rating of User 2 on Movie C: 4.94633897
- Iteration times: 49403

(d)

Mistake: when update Q[k][j], the P[i][k] has been updated, it should use the original P[i][k]

Correct code:



Result in next page

- Prediction rating of User 2 on Movie C after modify: 4.9459001
- Iteration times: 52880

- Comparison: The object value is a little lower than the false code, but takes more iterations to converge.
- Reason I guess: The original code uses the updated p[i][k], this will lead update more for 1 step compare with using original p[i][k], so this will lead fewer iterations.