ENGG 4030 Homework_3

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 柳一村) Date: 09/04/2018

Name LIU Yicun SID 1155092202

Q1 K Means Clustering

(a) To begin with, I firstly pre-process the given 'ubyte' file into a decimal file. Because the first 16 offsets of the image dataset are not pixels (they are general info about the dataset), I simply omit them and combine the rest pixels and labels into one file, where the pixels are presented in decimal between 0 and 255.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns

Fig 1: first 16 offsets are not pixels

```
def convert(img_file, label_file, txt_file, n_images, Tformat):
  lbl_f = open(label_file, "rb")    # MNIST has labels (digits)
  img_f = open(img_file, "rb")      # and pixel vals separate
  txt_f = open(txt_file, "w")       # output file to write to

  img_f.read(16)    # discard header info
  lbl_f.read(8)     # discard header info

  for i in range(n_images):   # number images requested
    lbl = ord(lbl_f.read(1))  # get label (unicode, one byte)
    txt_f.write("digit " + str(lbl))
    txt_f.write(":")
    for j in range(784):  # get 784 vals from the image file
      val = int(ord(img_f.read(1)))
      if(Tformat == "decimal"):
        txt_f.write(str(val) + " ")
      else:
        if(Tformat == "binary"):
          if(val > 200):
            txt_f.write(str(1) + " ")
          else:
            txt_f.write(str(0) + " ") # will leave a trailing space
    txt_f.write("\n")  # next image

  img_f.close(); txt_f.close(); lbl_f.close()
```

Fig 2: conversion function to convert 'ubyte' into decimal or binary

I convert both the image and label file of the training and testing set and pack the image and label into one file like:

```
digit 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 18 18 18 126 136 175 26 166 255 247 127 0 0 0 0 0 0 0 0 0 0 0 0 30 36 94 154 170 253 253 2
0 0 49 238 253 253 253 253 253 253 253 253 251 93 82 82 56 39 0 0 0 0 0 0 0 0 0 0 0 0 18 219 253 253
0 0 0 0 0 0 0 80 156 107 253 253 205 11 0 43 154 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 14 1 154 253
0 139 253 190 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 190 253 70 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 81 240 253 253 119 25 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 93 252 253 187 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 249
0 46 130 183 253 253 207 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 39 148 229 253 253 253 250 182 0 0
253 253 201 78 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 23 66 213 253 253 253 253 198 81 2 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 55 172 226 253 253 253 253 244 133 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Fig 3: sample of one digit in the converted file

For the implementation of K-Means in Mapreduce framework, the mapper should read the result of the previous centroids and assign the digit image into the nearest cluster, while the reducer should receive digit image emitted from the mapper (with assigned cluster) and re-count the centroid. Each MapReduce task stands for one iteration of the K-means.

If it is the first iteration, it will randomly assign centroid points and run the first mapper.

```
if os.path.exists(fname):
        if os.path.getsize(fname) == 0:
            with open(fname,"a") as f:
                for o in range(0,10):
                    w_str = 'Centroid ' + str(o) + ':'
                    for k in range(0, 28*28):
                        w_str += str(rnd.randint(0,255)) + ' '
                    w_str = w_str[:-1]
                    w_str += ',0\n'
                    f.write(w_str)
            f.close()
            exit()
```

Fig 4: in the first iteration, the centroid is randomly chosen

The mapper first read the previous centroid from an external file and store in a numpy matrix.

```
fname = 'centroid.txt'

centroid = np.zeros((10,28,28))
```

Fig 5: the centroid is read from outside and stored in a numpy matrix

```
with open(fname) as f:
    lines = [line.rstrip('\n') for line in open(fname)]
    for line in lines:
        line = line.strip()
        line, count = line.split(',')
        header, cents = line.split(':')
        header, index = header.split()
        cents = cents.split()
        for r in range(0,28):
            for c in range(0,28):
                tot = r * 28 + c
                if(tot % 2 == 0):
                    centroid[int(index)][r][c] = float(cents[tot])
                else:
                    centroid[int(index)][r][c] = float(cents[tot])
```

Fig 6: the mapper first read the previous centroid

For each digit image passed to the mapper, the mapper calculates the Euclidean distance from the digit (784 dimension) to all 10 cluster centroids. Then it assigns the digit to the cluster with the smallest Euclidean distance.

```
for line in sys.stdin:
    line = line.strip()
    header, line = line.split(':')
    header, digit = header.split()
    pixels = line.split()

    close_distance = distance(pixels,0)
    close_index = 0
    for i in range(1,10):
        new_dist = distance(pixels,i)
        if(close_distance > new_dist):
            close_distance = new_dist
            close_index = i

    str_pass = ""
    for pixel in pixels:
        str_pass = str_pass + str(pixel) + " "
    str_pass = str_pass[:-1]

    print '%s\t%s' % (str(close_index), str_pass)
```

Fig 7: calculating the distance and emit the tuple

In the last, the mapper emits the assigned cluster number to the reducer, along with the 784 pixels in the digit image.

For one reducer, it receives all digit images assigned to the same cluster in the mapping phase and re-count the centroids of the 10 clusters.

```
current_index = None

centroid = np.zeros(28*28)
num_member = 0

for line in sys.stdin:
    index, pixels = line.strip().split('\t')
    pixels = pixels.split()

    if current_index:
        if(index == current_index):
            for r in range(0,28):
                for c in range(0,28):
                    centroid[r*28+c] += int(pixels[r*28+c])
            num_member += 1
        else:
            w_str = 'Centroid ' + current_index + ':'
            for i in range(0, 28*28):
                w_str += str(centroid[i]/num_member) + " "
            w_str = w_str[:-1]
            w_str += "," + str(num_member) + "\n"
            print(w_str)
            centroid = np.zeros(28*28)
            num_member = 0
    current_index = index

if(current_index>1):
    w_str = 'Centroid ' + str(current_index) + ':'
    for i in range(0, 28*28):
        w_str += str(centroid[i]/num_member) + " "
    w_str = w_str[:-1]
    w_str += "," + str(num_member)
    print(w_str)
```

Fig 8: implementation of reducer in K-Means

Considering in actual case, a perfect ‘convergence’ case is hard to reach (may take many iterations), here I simply run the MapReduce task for 10 to 20 iterations, until the centroid points are steady.

```
chmod +x mapper_q1.py reducer_q1.py

for i in `seq 1 10`;
do
    hdfs dfs -rm -r /user/ly116/output_q1

    hdfs dfs -mkdir /user/ly116/mnist_train
    hdfs dfs -put ./dataset/mnist_train.txt /user/ly116/mnist_train

    hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar \
    -D mapreduce.map.memory.mb=2048 \
    -D mapreduce.reduce.memory.mb=1024 \
    -D mapred.map.tasks=20 \
    -D mapred.reduce.tasks=10 \
    -input /user/ly116/mnist_train/* \
    -output output_q1 \
    -file mapper_q1.py -mapper mapper_q1.py \
    -file reducer_q1.py -reducer reducer_q1.py \
    -file centroid.txt

    hdfs dfs -cat /user/ly116/output_q1/* > centroid_new.txt

    cat centroid_new.txt > centroid.txt
    cat centroid_new.txt >> centroid_all.txt

    hdfs dfs -rm -r /user/ly116/output_q1
done
```

Fig 9: bash file of running MapReduce

The running time of each iteration is about 20 seconds, with 20 mappers and 10 reducers.

Elapsed:	19sec
Diagnostics:	
Average Map Time	11sec
Average Shuffle Time	3sec
Average Merge Time	0sec
Average Reduce Time	0sec

I format the output of centroids as the final result, as requested:

```
Centroid 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 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.016316015457277802 0.03535136682410191 0.10834406755402891 0.037641339058251 0.0166022613424932 0.027908973
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0001431229426077 0.0178903678259625 0.21482753685415773
0.32632030914555604 0.2749391727493917 0.2853871475597538 0.207528266781165 0.08573064262201231 0.002146844139
0.0007156147130385 0.0 0.0528123658222413 0.044797481036210104 0.0254758837841706 0.0267639902676399 0.0 0.0 0
0.013596679547731501 0.019894089022470304 0.15128095033633893 0.46715328467153283 0.6849864033204522 0.8897953
0.9745241162158294 0.7054529841133533 0.39129812508945183 0.18720480893087163 0.27450980392156865 0.4086160011
0.29898382710748533 0.4010304851867754 0.44940603978817806 0.31830542435952486 0.18219550593960213 0.110634034
0.0 0.0 0.0 0.07170459424645771 0.35938170888793475 0.6889938457134679 1.3194504079003864 2.2531844854730214 3
3.36653785601832 3.152569056819808 2.2028052096751107 1.7402318591670245 2.1027622727923285 3.4788893659653644
4.399313009875483 3.581651638757693 2.6991555746386147 1.5332760841562902 0.7047373694003148 0.175039358809217
0.0 0.0 0.0 0.0 0.0280520967511092 0.3943037068842135 1.4084728782023759 3.364104765993989 5.647631315299843 8
15.215400028624588 15.454129096894231 14.619436095606126 12.498354086160012 11.359954200658365 12.183197366537
20.039931300987547 16.270359238585947 12.74423930156004 8.47674252182625 5.160870187491055 2.5875196794046085
0.06469157005868041 0.0 0.0 0.0 0.07556891369686561 0.3829969944182052 1.5083726921425504 4.112351509947045 8.
23.513525118076426 35.283669672248465 45.61557177615572 51.38700443681122 55.92171175039359 57.5985401459854 5
54.71146414770288 51.12322885358523 44.66094174896236 34.306426220123086 24.130385000715616 16.22355803635323
```

(b) In (a), I have the final centroids, here I use a post-processing script to get the final accuracy of different threshold X:

```
with open(sname) as s:
    lines = [line.rstrip('\n') for line in open(sname)]
    for line in lines:
        line = line.strip()
        header, line = line.split(':')
        header, digit = header.split()
        pixels = line.split()

        close_distance = distance(pixels,0)
        close_index = 0
        for i in range(1,10):
            new_dist = distance(pixels,i)
            if(close_distance > new_dist):
                close_distance = new_dist
                close_index = i
        dist_store[close_index].append(tuple((close_distance,int(digit))))
        which_line+=1
        print(which_line)

with open(wname,'w') as w:
    for i in range(0,10):
        temp = []
        dist_store[i].sort(key=lambda tup: tup[0])
        for k in range(0,int(len(dist_store[i])*th)):
            temp.append(dist_store[i][k])

        index_count = [0] * 10
        for tup in temp:
            index_count[tup[1]] += 1
        final_index = index_count.index(max(index_count))
        pos_count = 0
        tot_count = 0
        for tup in dist_store[i]:
                if(tup[1] == final_index):
                    pos_count += 1
                tot_count += 1
        str_out = 'Cluster Number ' + str(i) + ':' + str(len(dist_store[i])) + ',' + str(int(len(dist_store[i]) * th))
        w.write(str_out)
```

Fig 10: part of the post-processing script

Here is the result after 10-20 iterations, when the result of K-Means mainly converges.

X = 0.05

Cluster Number	Images in the entire dataset	Images which are considered(m)	Major label of central images	Correctly clustered image	Classification Accuracy %
0	819	40	2	713	87
1	996	49	9	328	33
2	947	47	7	336	35
3	541	27	0	423	78
4	1066	53	8	576	54
5	1317	65	3	709	54
6	489	24	0	446	91
7	1273	63	7	492	39
8	1635	81	1	1108	68

9	917	45	6	794	87
Total				5925	59

X = 0.1

Cluster Number	Images in the entire dataset	Images which are considered(m)	Major label of central images	Correctly clustered image	Classification Accuracy %
0	819	81	2	713	87
1	996	99	9	328	33
2	947	94	7	336	35
3	541	54	0	423	78
4	1066	106	8	576	54
5	1317	131	3	709	54
6	489	48	0	446	91
7	1273	127	7	492	39
8	1635	163	1	1108	68
9	917	91	6	794	87
Total				5925	59.2

X = 0.5

Cluster Number	Images in the entire dataset	Images which are considered(m)	Major label of central images	Correctly clustered image	Classification Accuracy %
0	819	409	2	713	87
1	996	498	9	328	33
2	947	473	7	336	35
3	541	270	0	423	78
4	1066	533	8	576	54
5	1317	658	3	709	54
6	489	244	0	446	91
7	1273	636	9	445	35
8	1635	817	1	1108	68
9	917	458	6	794	87
Total				5878	58.7

X = 1

Cluster Number	Images in the entire dataset	Images which are considered(m)	Major label of central images	Correctly clustered image	Classification Accuracy %
0	819	819	2	713	87
1	996	996	4	422	42
2	947	947	7	336	35
3	541	541	0	423	78
4	1066	1066	8	576	54
5	1317	1317	3	709	54
6	489	489	0	446	91
7	1273	1273	7	492	39
8	1635	1635	1	1108	68
9	917	917	6	794	87
Total				6019	60.2

For the four possible values of X, it is observed that the difference is in cluster 1 and 7.

- When X=0.05 and 0.1, K-Means gives the same result upon convergence. The label of cluster 1 is not the best label while others are best labels (best->majority in all pts belonging to the cluster). The accuracy of the clustering is 59.2%.
- When X=0.5, K-Means performance suffers from some noise, the label of both clusters 1 and 7 is not perfectly determined. This leads to the accuracy decrease for 0.5% to 58.7%
- When X=1, the label of all clusters is ensured to be the majority in the whole cluster. Both cluster 1 and 7 are perfectly determined and the best performance is 60.2%

From the above observation, I think X = 1 would perform the best in most cases. The possible reason why the worst performance happens when X=0.5 is that there are probably lots of noise pts centralized in the middle belt of the cluster. To that end, I believe X=1 is the best value for X, but the difference of performance is relatively small.

(c) First, we merge the converted training and testing set into a total set containing 70000 images and their labels.

```
$cat mnist_train.txt > mnist_total.txt
$cat mnist_test.txt >> mnist_total.txt
```

Here we simply use bash script to split the lines into 5 and 10 equal parts, using:

```
$ split -l 14000 mnist_total.txt
$ split -l 7000 mnist_total.txt
```

Then we use the same MapReduce framework with X = 1:

5-Folder Validation

Testing Set	Classification Accuracy %
Part 1	61.9
Part 2	59.2
Part 3	58.9
Part 4	60.5
Part 5	60.7
Average	60.2

10-Folder Validation

Testing Set	Classification Accuracy %
Part 1	58.9
Part 2	61.7
Part 3	61.3
Part 4	61.5
Part 5	58.7
Part 6	57.2
Part 7	61.9
Part 8	59.3
Part 9	60.1
Part 10	61.4
Average	60.2

The result of the validation indicates that there is no overfitting in our K-Means algorithm (Indeed, unsupervised method is less prone to overfitting).

Q2. Bernoulli Mixture

(a) Mapper:

For each line (digit image):

E Step, compute score, based on previous result.

$$\gamma(z_{nk}) = \pi_k \, p(\mathbf{x_n}|\mathbf{q_k}) / \left(\sum_{j=1}^{K} \pi_j \, p(\mathbf{x_n}|\mathbf{q_j}) \right),$$

Emit(k,y(Znk))

Reducer:

For each line (possed by the reducer):

M Step, re-compute the pi(k) and q(k):

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

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

(b)

Sequential Implementation:

Initialize step:

```
# Init Model
pi = np.random.rand(K)
#mu = [[.5, .5, .5, .5], [.9, .1, .1, .9]]
mu = np.random.rand(K, D)
r = np.zeros([N, K]) # soft assignment
weight = np.zeros(K)
```

E step and M step:

```
# Main loop
while iteration <= iterations:
    # E step
    print('Iter:',iteration)
    which_e = 0
    for ii in range(0, N):
        observation = observations[ii]
        for kk in range(0, K):
            weight[kk] = pi[kk]
            for jj in range(0, D):
                weight[kk] *= mu[kk][jj]**int(observation[jj]) * (1-mu[kk][jj])**(1-int(observation[jj]))
                which_e +=1
                if(which_e % 78600000 == 0):
                    print('E Step:'+ str(int(which_e /78600000)) +'/6')
        if(sum(weight) != 0):
            r[ii, :] = weight / sum(weight)

    # M step
    nk = [sum(r[:, ii]) for ii in range(K)]

    new_mu = np.zeros([K, D])
    which_m = 0
    for kk in range(0, K):
        mean = np.zeros(D)
        for ii in range(0, N):
            mean += r[ii, kk]*int(observations[ii])
            which_m +=1
            if(which_m % 78600000 == 0):
                    print('E Step:'+ str(int(which_m /78600000)) +'/6')
        new_mu[kk] = mean / nk[kk]
    pi = nk / sum(nk)
```

MapReduce Implementation:

For the MapReduce implementation, I let the mapper to do the E Step, and the reducer to do the M Step.

Mapper:

```
with open(fname) as f:
    lines = [line.rstrip('\n') for line in open(fname)]
    for line in lines:
        line = line.strip()
        str_1, str_2 = line.split(',')
        pis = str_1.split()
        mus = str_2.split()
        for k in range(0,K):
            pi[k] = float(pis[k])
            for d in range(0,D):
                mu[k][d] = float(mus[k*784+d])

for line in sys.stdin:
    line = line.strip()
    header, line = line.split(':')
    header, digit = header.split()
    observation = line.split()

    for kk in range(0, K):
        weight[kk] = pi[kk]
        for jj in range(0, D):
            weight[kk] *= mu[kk][jj]**int(observation[jj]) * (1-mu[kk][jj])**(1-int(observation[jj]))

    for k in range(0,K):
        if(sum(weight) != 0):
            r[k] = weight[k] / sum(weight)

    for k in range(0,K):
        str_pass = ""
        for pixel in observation:
            str_pass = str_pass + str(pixel) + " "
        str_pass = str_pass[:-1]
        str_pass += ',' + str(r[k])
        print '%s\t%s' % (str(k), str_pass)
```

Reducer:

```
K = 10
D = 784
N = 60000

r = np.zeros(K)
pi = np.zeros(K)

current_index = None
count_score = 0

fname = 'params.txt'

for line in sys.stdin:
    line = line.strip()
    index, str_pass = line.split('\t')
    observation, score = str_pass.split(',')
    observation = observation.split()

    if (index == current_index):
        count_score += float(score)
        sum_pre = [pre+score*pixel for pre,pixel in zip(sum_pre, observation)]
    else:
        if current_index:
            pi[index] = count_score/N
            sum_pre = [x/count_score for x in sum_pre]
            print str(sum_x).lstrip('[').rstrip(']')
            current_index = index
            count_score = score
            sum_pre = [pre*score for pre in observation]

if (current_index == index):
    pi[index] = count_score/N
    sum_pre = [x/count_score for x in sum_pre]
    print str(sum_x).lstrip('[').rstrip(']')

print str(pi).lstrip('[').rstrip(']')
```

The runtime is about 2 min per iteration:

Job Name:	streamjob6397629586226348784.jar
User Name:	ly116
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Mon Apr 09 01:49:42 HKT 2018
Started:	Mon Apr 09 01:49:47 HKT 2018
Finished:	Mon Apr 09 01:51:34 HKT 2018
Elapsed:	1mins, 47sec
Diagnostics:	
Average Map Time	56sec
Average Shuffle Time	46sec
Average Merge Time	0sec
Average Reduce Time	5sec

X = 1

Cluster Number	Images in the entire dataset	Images which are considered(m)	Major label of central images	Correctly clustered image	Classification Accuracy %
0	2329	2329	0	894	38
1	724	724	1	578	80
2	1248	1248	2	636	51
3	978	978	3	590	60
4	802	802	4	576	72
5	432	432	5	252	32
6	1190	1190	6	839	70
7	1327	1327	7	692	52
8	276	276	8	86	31
9	694	694	9	340	49
Total				5483	55

The overall accuracy is about 55%(both sequential and MapReduce), which is worse than K-Means.

Q3. Dimensionality Reduction

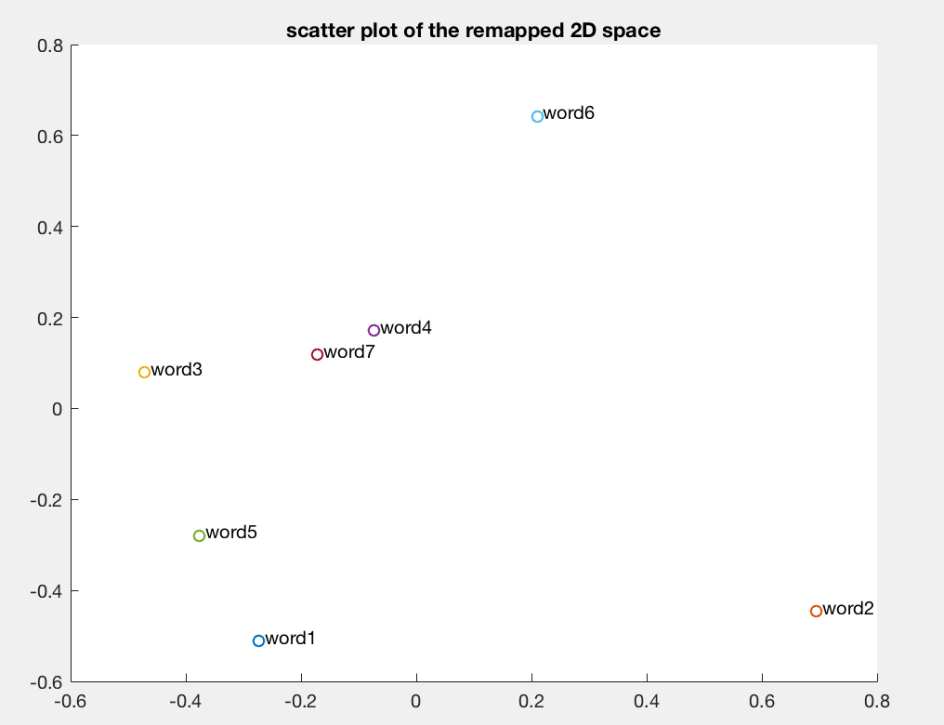
(a) Use MATLAB to achieve the SVD decomposition, we get:

```
U =

   -0.3058   -0.2808   -0.1790   -0.5767   -0.6431   -0.2224
   -0.2714    0.2169    0.7668    0.1711   -0.4291    0.2790
   -0.2315    0.4701   -0.0148    0.2724   -0.0570   -0.8049
   -0.3574   -0.4700   -0.3029    0.7123   -0.2103    0.0899
   -0.7026   -0.2264    0.2180   -0.2271    0.5943   -0.0531
   -0.3973    0.6173   -0.4904   -0.0710   -0.0405    0.4626


S =

   24.6102         0         0         0         0         0         0
         0   11.4451         0         0         0         0         0
         0         0    9.2191         0         0         0         0
         0         0         0    7.6694         0         0         0
         0         0         0         0    4.8178         0         0
         0         0         0         0         0    1.8224         0


V =

   -0.2729    0.6948   -0.4719   -0.0726   -0.3760    0.2101   -0.1713
   -0.5117   -0.4455    0.0792    0.1713   -0.2796    0.6418    0.1182
   -0.3660    0.4191    0.7229    0.1730   -0.1459   -0.2439    0.2391
   -0.2183   -0.3708   -0.1770   -0.1685   -0.6055   -0.6225   -0.0317
   -0.4344   -0.0330   -0.3697    0.6315    0.4248   -0.3070   -0.0001
   -0.3550   -0.0671    0.2396   -0.2856    0.2468   -0.0248   -0.8179
   -0.4092    0.0100   -0.1519   -0.6532    0.3888   -0.0451    0.4791
```

(b)

i)

```
A = [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
[U,S,V] = svd(A);

figure
for i = 1:7
    x = V(1,i);
    y = V(2,i);
    dx = 0.01; dy = 0.01;
    scatter(x,y)
    hold on
    name = strcat('word',int2str(i));
    text(x+dx, y+dy, name);
    hold on
end
title('scatter plot of the remapped 2D space');
```

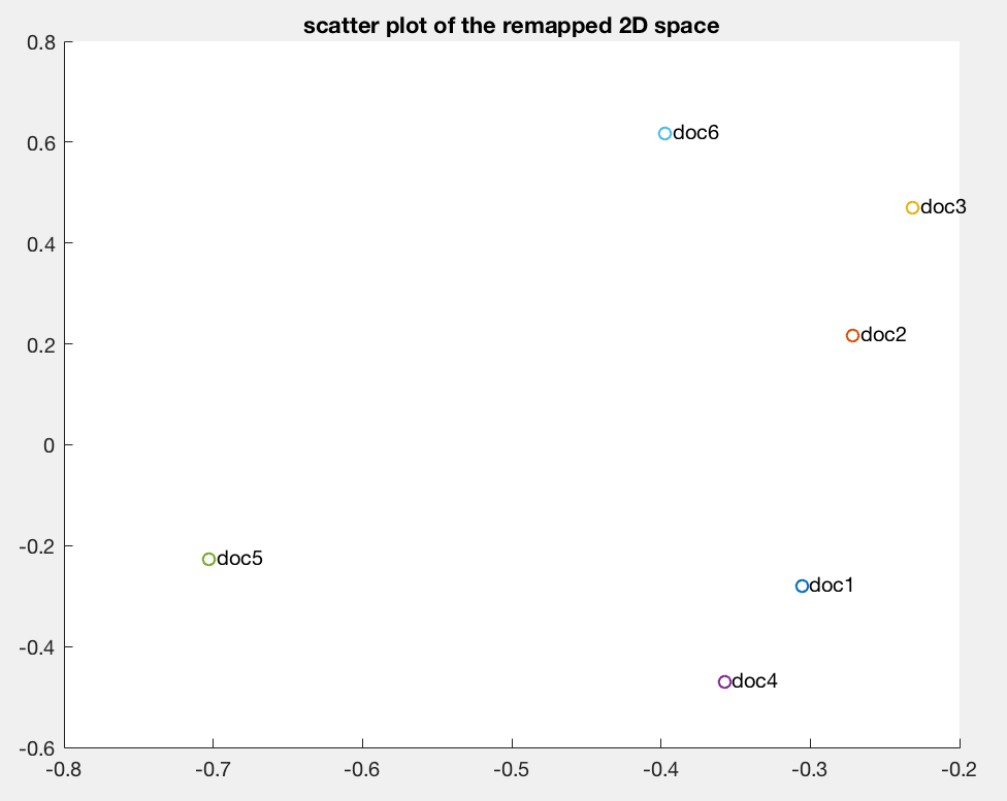


Group 1: word 4, word7
Group 2: word1, word3, word5
Group 3: word2
Group 4: word6

ii)

axis 1: [-0.2729 0.6948 -0.4719 -0.0726 -0.3760 0.2101 -0.1713]
axis 2: [-0.5117 -0.4455 0.0792 0.1713 -0.2796 0.6418 0.1182]

$$\begin{bmatrix} -0.27 & -0.51 & -0.37 & -0.22 & -0.43 & -0.35 & -0.41 \\ 0.69 & -0.45 & 0.42 & -0.37 & -0.03 & -0.07 & 0.01 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 5 \\ 0 \\ 4 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -5.68 \\ 5.37 \end{bmatrix}$$



Group 1: doc 2, doc 3, doc 6
Group 2: doc 1, doc 4
Group 3: doc 5