GEORGIA TECH, DEPARTMENT OF ISYE

ISyE 6740 - Take Home Exam 1

February 11, 2015

Contents

1 About the problem and data: Information Loss

Originally, the data of handwritings is represented by 16 times 16 matrices. One of the best ways of defining the distance between two data points is convolution. Notices that the same digits that are written and represented on matrices could shift a little bit each time. Therefore, in order to decide the degree that two data points are similiar, we should try different shifts in the convolution between two data points, and use the one that fits best.

If we choose to deal with our data as vectors with 256 elements in each vector, however, this shift would not be considered any more. For two digits that are exactly the same, but with one of them shift one pixel to the right, they would be represented as two very different vectors. We should say that by doing this, there is loss happens to the information stored in our data.

Then in this problem, when we use PCA to reduce the demension of our data, loss happens to the information in our data again. With the first 6 principle components in hand, we may still lose more than 50% of information.

However, we could choose to bear this loss, as this may greatly save our computational capacity. In the original case, we need to perform several convolutions between each two data points. After ignoring the location shift, the number of convolutions we need to do each time became one. When implementing PCA method, the amount of computation we need to do was greatly decreased again. Yet this information loss is noticable.

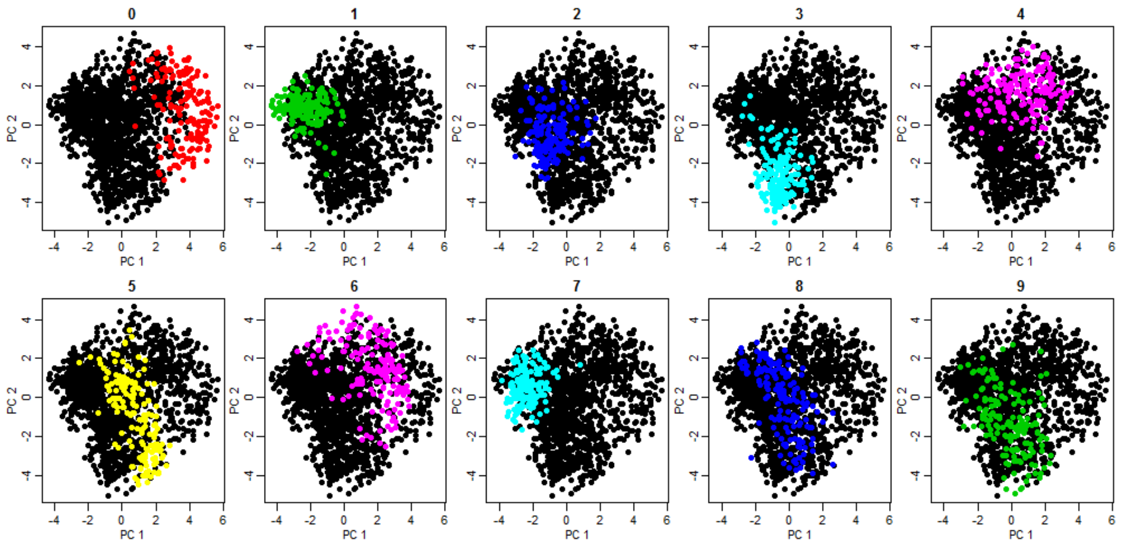

Figure 1.1: The group shape of each digits plotted on the first two PCAs. It can be seen that they are hard to be seperated by only two PCAs

2 Solving the Problem

All the code I used to generate the results is attached in the last section. Yet I will also discuss them seperately in this section along with tackling this problem. To begin with, here's the concole commend I used to clear memories, define global variables.

```r
# Clear memory, include libraries
rm(list=ls())
library(mvtnorm)

# Read handwritten digits data
myData=read.csv("semeion.csv",header=FALSE)

# Build data matrix with (thresholded) pixel and label data
myX=data.matrix(myData[,1:256])
myLabel=apply(myData[,257:266],1,function(xx){
  return(which(xx=="1")-1)
})

# Define global variables
ii <- dim(myX)[1]
dd <- dim(myX)[2]
kk <- 10
qq <- 2
niter <- 20
```

readdata.r

Here the libary *mvtnorm* is a R package that need to be installed. It provides some convenient functions for calculating probabilities and sample random values in multivariate normal distributions. The functions I used in this problem are *dmvnorm* and *rmvnorm*

semeion.csv is the handwriting digits data we used in this problem. *ii*, *dd* and *kk* are the number of data points, data deminsions and data groups respectively. *qq* here is the q, which represents the number of principle components we used in computing variance matrices. *niter* is the number of iterations each time we run the algoritm.

Then it came the first step.

2.1 Initialization

In this step, one of the most important work is to initialize a 1593 by 10 gamma matrix, which will store the probability that a certain data point belongs to a certain data group. As being asked, I used K-means method to complete this initialization by assigning the respective element on each row of gamma to be 1, with others stay 0.

We also defined *mu*, *pi* and *sigma*, which are parameters for data groups. They will be initialized in M step in our E-M iterations. *px* is also a 1593 by 10 matrix, with each elements in it to be the multiplication of the respective pi and normal probability.

the list *like* will store the likelihood in each iteration.

```r
# Cluster data by using k(10)-means methodS
myCluster <- kmeans(myX, 10, iter.max = 20, nstart = 10)

# Initialization: assignments of data
gamma <- matrix(0, nrow=ii, ncol=kk)
for(i in 1:ii) {gamma[i, myCluster$cluster[i]] <- 1}
mu <- matrix(0, nrow=kk, ncol=dd)
pi <- rep(0,10)
# Likelihood
like <- rep(0, niter)

# Initialization of covariance matrices
sigma <- array(dim=c(10,256,256))
px <- matrix(0,ii,kk)
```

initialization.r

Now we're ready for the E-M iterations. The code is shown in the next page. Please note that I changed the "%*%" into "% * %" (added two spaces) as the first one could not be recogized by *LaTeX*.

Similiar with the formular given in the problem sheet, I recompute (or say initialze, when it's the first iteration) the data group parameters *mu*, *pi* and *sigma* in M step, and recompute the gamma matrix in E step in each iteration. besides, I computed the log likelihood each time and stored it in the list *like*.

```r
# Iterations
for(it in 1:niter) {

  # ------------------------ M step ------------------------ #

  # Initialize / Recompute mu, which is the group average.
  for(k in 1:kk) {
    muk <- rep(0,256)
    for(i in 1:ii) {muk <- muk + gamma[i,k] * myX[i,] }
    mu[k,] <- muk/sum(gamma[,k])
  }
  # Initialize / Recompute pi, which is the group fraction of data points
  pi <- colSums(gamma)/ii

  # Initialize / Recompute pi, which is the group fraction of data points
  for(k in 1:kk) {
    covk <- matrix(0,256,256)
    # Calculate cov matrix for spectral decomposition
    for(i in 1:ii) {
      xiBar <- myX[i,]-mu[k,]
      covki <- (xiBar %*% t(xiBar)) * gamma[i,k]
      covk <- covk + covki
    }
    # PCA part
    covk <- covk / sum(gamma[,k])
    eigk <- eigen(covk,symmetric=TRUE)
    pcak <- eigk$vectors[,1:qq]
    # Compute estimation of variance matrix
    deltak <- sum(eigk$values[qq+1:dd], na.rm = T) / (dd-qq)
    diagk <- diag(qq)
    for(q in 1:qq) {diagk[q,q] <- sqrt(eigk$values[q]-deltak)}
    # Compute estimation of variance matrix - continue
    wq <- pcak %*% diagk
    sigma[k,,] <- wq %*% t(wq) + (deltak * diag(dd))
  }

  # ------------------------ E step ------------------------ #

  for(k in 1:kk) {px[,k] <- pi[k]*dmvnorm(myX, mu[k,], sigma[k,,], log = FALSE)}
  for(i in 1:ii) {for(k in 1:kk) { gamma[i,k] <- px[i,k] / sum(px[i,]) }}

  # ---------------------- Likelihood ---------------------- #

  like[it] <- sum(log(rowSums(px)))
  print(c('loop','no.',it,'log','likelihood','is',like[it]))

}
```

emalgorithm.r

When it came to q=0, the M step, especially the part of computing variance matrices list (3-dimension matrix) *sigma* in this algorithm would be slightly different, as the principle components vectors did not exist any more. The modified code is shown:

```r
# ------------------------- M step ------------------------- #

# Initialize / Recompute mu, which is the group average.
for(k in 1:kk) {
  muk <- rep(0,256)
  for(i in 1:ii) {muk <- muk + gamma[i,k] * myX[i,] }
  mu[k,] <- muk/sum(gamma[,k])
}
# Initialize / Recompute pi, which is the group fraction of data points
pi <- colSums(gamma)/ii

# Initialize / Recompute gamma, which is the group fraction of data points
for(k in 1:kk) {
  covk <- matrix(0,256,256)
  # Calculate cov matrix for spectral decomposition
  for(i in 1:ii) {
    xiBar <- myX[i,] - mu[k,]
    covki <- (xiBar %*% t(xiBar)) * gamma[i,k]
    covk <- covk + covki
  }
  # PCA part
  covk <- covk / sum(gamma[,k])
  eigk <- eigen(covk,symmetric=TRUE)
  # Compute estimation of variance matrix
  deltak <- sum(eigk$values[qq+1:dd], na.rm = T) / (dd-qq)
  sigma[k,,] <- (deltak * diag(dd))
}
```

emalgorithm_qq=0.r

Each iteration takes about 3 seconds to run, therefore the entire program takes more than 1 minute to finish. All the outcomes are stored nicely, and we will come to them now.

2.2 Convergence

So here is the outcome in console each time q changed. When the progress of the likelihoods between two iterations is less than 1, which is quite a small amount when the likelihoods are more than tens of thousands. Therefore, in q=0 and q=2, the number of iterations I did was 20, which could guarantee convergence at most of the times. In q=4 and q=6 I did 40 iterations. Yet the in the plot I only shows the first 20 iterations for each q (to make it looks nicer.)

```

# -------------------------- qq = 0 -------------------------- #

> like
[1] -210923.6 -210837.7 -210794.4 -210767.6 -210751.6 -210741.1 -210736.3
[8] -210733.5 -210731.0 -210729.8 -210729.6 -210729.5 -210729.3 -210729.2
[15] -210729.1 -210729.1 -210729.0 -210729.0 -210728.9 -210728.7
> AIC
[1] 421459.5

# -------------------------- qq = 2 -------------------------- #

> like
[1] -172986.3 -171832.4 -171514.7 -171262.1 -171079.1 -170905.6 -170777.3
[8] -170696.5 -170608.9 -170559.3 -170518.5 -170494.6 -170465.1 -170457.3
[15] -170455.6 -170455.1 -170454.5 -170454.3 -170453.7 -170451.5
> AIC
[1] 341924.9

# -------------------------- qq = 4 -------------------------- #

> like
[1] -150332.8 -148451.4 -147821.1 -147509.7 -147288.8 -147164.9 -147001.0
[8] -146878.4 -146794.9 -146746.8 -146721.5 -146694.6 -146667.5 -146643.8
[15] -146630.4 -146629.8 -146626.2 -146623.4 -146621.6 -146614.2 -146612.6
[22] -146611.7 -146602.3 -146594.2 -146588.7 -146577.1 -146566.7 -146557.2
[29] -146550.7 -146543.1 -146536.6 -146523.7 -146515.4 -146507.7 -146486.8
[36] -146477.0 -146469.6 -146467.7 -146466.6 -146464.4
> AIC
[1] 294954.8

# -------------------------- qq = 6 -------------------------- #

> like
[1] -130748.9 -128631.3 -127953.6 -127563.9 -127398.0 -127256.1 -127122.4
[8] -127030.6 -126948.2 -126854.6 -126848.2 -126843.6 -126837.6 -126823.3
[15] -126804.9 -126803.9 -126803.9 -126803.9 -126803.9 -126803.9 -126803.9
[22] -126803.9 -126803.9 -126803.9 -126803.9 -126803.9 -126803.9 -126803.9
[29] -126803.9 -126803.9 -126803.9 -126803.9 -126803.9 -126803.9 -126803.9
[36] -126803.9 -126803.9 -126803.9 -126803.9 -126803.9
> AIC
[1] 256621.8
```

summary.r

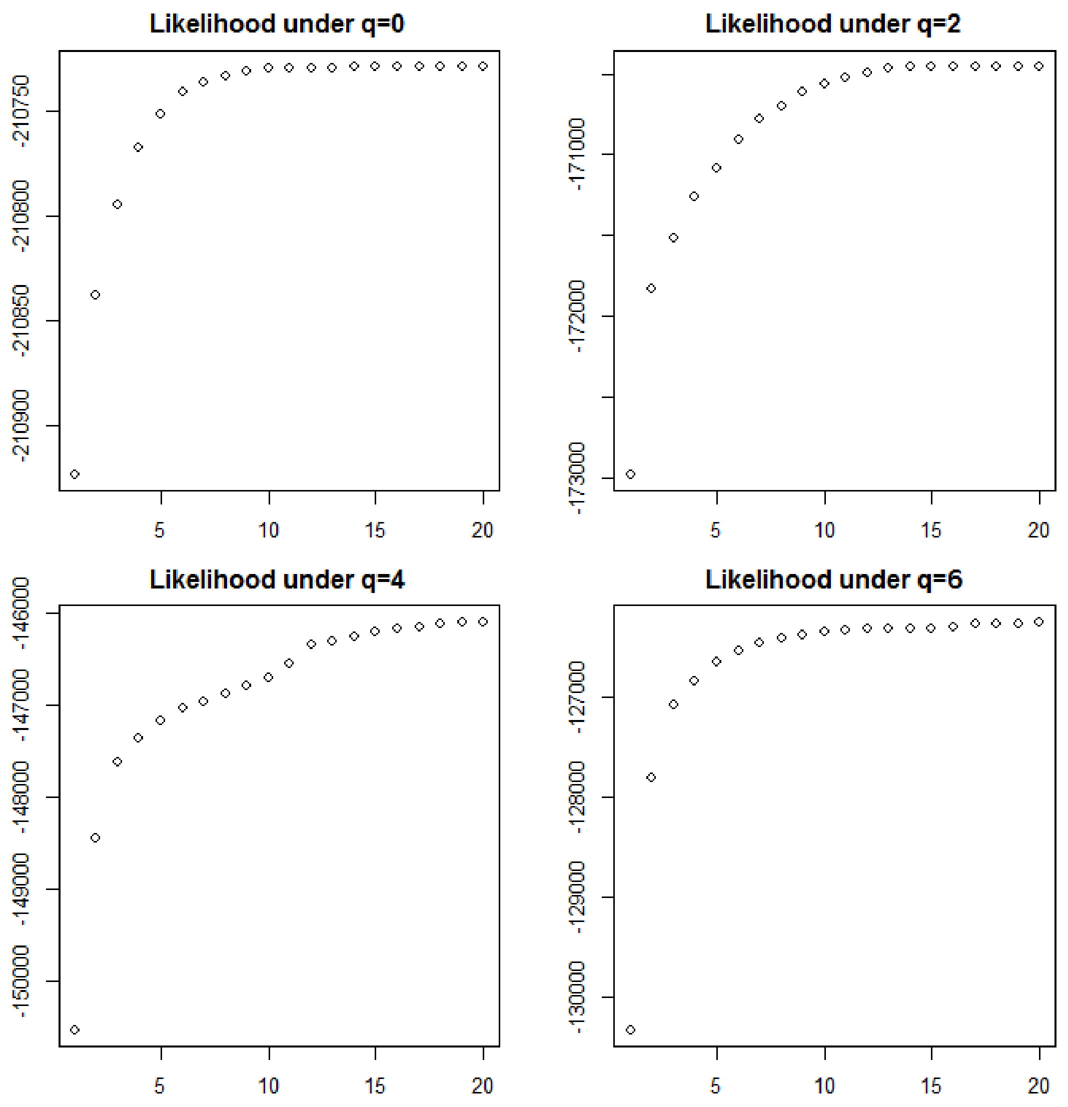

Figure 2.1: The likelihood of the first 20 iterations under each q

2.3 Choice of Number of Principle Components, q

The plot and the outcome data are got in different times I ran the program, so there might be a small difference. From these results we could see that the best q we should choose is 6, as it has the minimum AIC value. Now let's start to play with it.

2.4 Visualization of Clusters

Now I made a K = 10 by 6 panel plot for the case of q=6. In column 1, I plotted the cluster means. In each of the columns 2 through 6, I plotted 1 random draw from the cluster-speci c distribution (for a total of 5 random draws). The plotting code is shown below:

```r
# Print some pictures
dev.new(width=7,height=3.5)
par(mai=c(0.05,0.05,0.05,0.05),mfrow=c(10,6))

for(k in 1:kk){
  image(t(matrix(mu[k,], byrow=TRUE,16,16)[16:1,]),col=gray(0:1),axes=FALSE)
  box()
  for(j in 1:5){
    tempX <- rmvnorm(1, mean <- mu[k,], sigma[k,,])
    image(t(matrix(tempX, byrow=TRUE,16,16)[16:1,]),col=gray(0:1),axes=FALSE)
    box()
  }
}
```

visualization.r

Note that the elements in cluster means are not necessarily integer. Therefore, it might be better to plot it in a gray level image with more levels:

```r
# Print some pictures
dev.new(width=7,height=3.5)
par(mai=c(0.05,0.05,0.05,0.05),mfrow=c(10,6))

for(k in 1:kk){
  image(t(matrix(mu[k,], byrow=TRUE,16,16)[16:1,]),col=gray(seq(0,1,length=256)),axes=
    FALSE)
  box()
  for(j in 1:5){
    tempX <- rmvnorm(1, mean <- mu[k,], sigma[k,,])
    image(t(matrix(tempX, byrow=TRUE,16,16)[16:1,]),col=gray(seq(0,1,length=256)),axes=
    FALSE)
    box()
  }
}
```

visualization_gray.r

The result is shown in the next page:

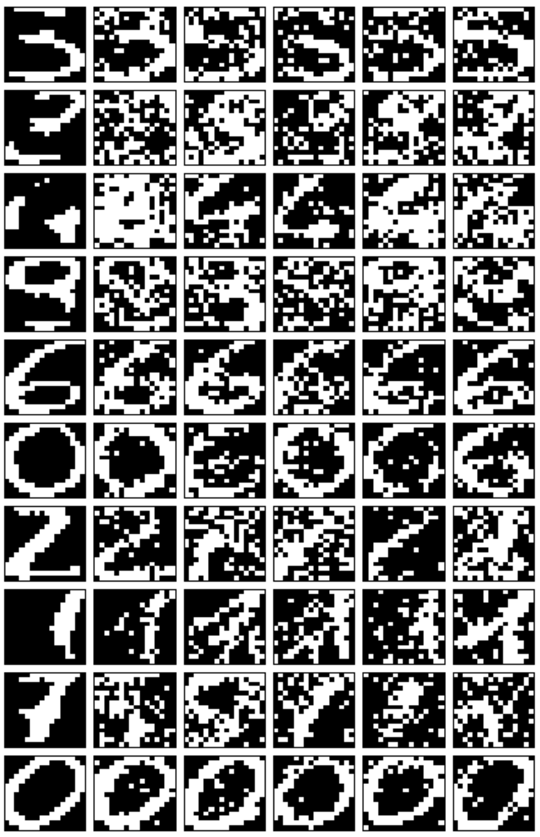

Figure 2.2: 10 by 6 panel plot in gray level with 2 level. The first column is the cluster means of each cluster; the columns 2 through 6 are random draws from each cluster respectively.

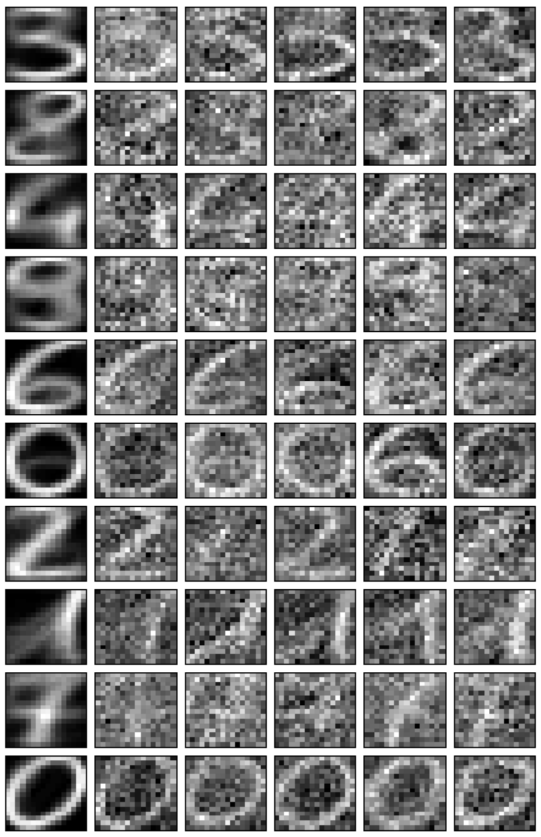

Figure 2.3: 10 by 6 panel plot in gray level with 256 level. The first column is the cluster means of each cluster; the columns 2 through 6 are random draws from each cluster respectively.

As we could see, the cluster means are pretty "like" a handwritten digit. In the meantime, though there is strong "white noise" in each random draw, we could still recognize the digit on it.

Yet we need to notice that the digits still mixes with each other. As we can see from this visualization, digits "0" and "6" were mixed together and formed a new cluster (sixth row); digits "3", "5" and "8" were not seperated clearly, and the plots of there cluster means looks pretty similiar. "1", "4" and "7" are looks very good though.

Let's look more on the details.

2.5 Accuracy Assessment

Firstly I assigned each data point to the group that has largest probability to contain it. Let's plit this cluster assignment and give a look on it:

```
> normLabel <- rep(0,ii)
> for(i in 1:ii) {normLabel[i] <- which.max(gamma[i,])}
>
> myGroup <- split(myLabel, normLabel)
> myGroup

$`1`
   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 5 5 6 8 8 9 9 9 9 3 3 3
  [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 9 3 3 3 3 3 3 3 3 3 3 3 3 3 3
  [75] 3 3 3 5 5 5 5 5 5 5 5 5 8 9 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5
 [112] 5 5 5 5 9 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [149] 3 3 3 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 8 8 9 9 3 3 3 3 3 3 3 3 3
 [186] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5
 [223] 5 8 9 9

$`2`
   [1] 0 2 2 2 2 2 2 2 3 3 5 5 5 5 5 7 8 8 8 8 8 8 8 8 8 8 8 9 2 2 2 2 2 2 2 5 5
  [38] 5 5 5 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 0 2 2 2 2 2 3 5 5 5 5 8 8 8 8 8
  [75] 8 8 8 8 8 9 9 9 2 2 2 2 2 2 3 5 5 8 8 8 8 8 8 8 8 8 9 9 9 2 2 2 2 2 2 2 3
 [112] 5 5 5 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 0 2 2 2
 [149] 2 2 2 2 2 2 2 2 2 3 3 3 3 5 5 5 5 5 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 [186] 8 8 8 8 9 9 9 9 9 9 9

$`3`
   [1] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 6 9 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
  [38] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 6 0 1 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1
  [75] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 9 4 4 4
 [112] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 6 9

$`4`
   [1] 1 4 4 5 5 5 7 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 3 4 5 5 5 5 5 6 7 8 8 8
  [38] 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 3 3 5 5 5 5 5 6 8 8 8 8 8 8 8 9 9 9 9 9 9 9
  [75] 9 9 9 9 9 9 9 3 4 5 5 5 5 6 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9
 [112] 9 9 3 3 3 4 5 5 5 5 5 5 5 5 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9
 [149] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 1 3 4 4 5 5 5 5 5 5 5 5 5 7 8 8 8 8 8
 [186] 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

$`5`
   [1] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 4 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 0 5 6
  [38] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 5 6 6 6
  [75] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 5 6 6 6 6 6 6 6 6 6 6
 [112] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6

$`6`
   [1] 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 6 6 6 0 0 0 0 0 6 0 0 0 0 0 0
  [38] 0 0 0 6 6 6 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 6 6 6 6 6 6 6 6 6 6
  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 6 6 6 6 6 6 6 6 6 6
```

```
$`7`
   [1] 1 1 2 2 2 2 2 2 2 2 2 2 2 2 7 7 7 7 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 6 7 7
  [38] 7 7 7 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 7 7 7 7 7 8 8 1 1 1 2 2 2 2 2 2 2
  [75] 2 2 2 2 2 2 2 4 4 7 7 7 7 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 6 7 7 7 7 7 7 7 7 7 7 1 1 2 2 2 2 2 2 2 2
 [149] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 6 7 7 7 8 9 9

$`8`
   [1] 1 1 1 1 1 1 1 1 1 1 1 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 7 1 1 1 1 1 1
  [38] 1 1 1 1 4 4 1 1 1 1 1 1 1 1 1 1 1 1 3 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4
  [75] 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 4 7 7

$`9`
   [1] 1 1 1 1 1 5 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 9 9 1 1 3 5 5 7 7 7 7 7 7 7 7
  [38] 7 7 7 7 7 9 9 1 1 1 1 1 1 1 1 4 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 9 1 1 1 1
  [75] 4 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 9 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 4 4
 [112] 4 4 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
 [149] 7 8 9 9 1 1 1 1 1 1 1 1 1 1 1 1 3 4 4 4 5 5 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7
 [186] 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 9

$`10`
  [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
 [39] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [77] 0 0 0 0 0 0 0 0 0 0
```

split.r

Now I could confirm my guess. Digit "1", "3", "4" were nearly all well clustered. "0" and "6" were well clustered with one cluster each, but also provided one more cluster as a mixture of the two. "2", "5", "8" and "9" mixed with each other badly and could not be seperated very well.

As the clusters could not match all the ten digits one by one, we have to take the most common element in each cluster as its label. Therefore, there could be two clusters that have the same label.

Finally, we need to compute the mis-categorized rate for each cluster and for the entire data set. I assigned each data point to the respective cluster, splited the data set, and compute the rate for each cluster. All of commands I used and all of the results in the R console is shown in the next page. The overall mis-categorized rate is 33.21%, which is quite good for such a simple clustering program.

```r
# ---------------------- Accuracy ---------------------- #

# Find most common digits for each cluster
normLabel <- rep(0,ii)
for(i in 1:ii) {normLabel[i] <- which.max(gamma[i,])}

# Split the clusters
myGroup <- split(myLabel, normLabel)
perc <- lapply(myGroup, function(group){
  return(sort(table(group), decreasing=TRUE)[1])
})

# The result matrix
accuracy <- matrix(0,4,10)

for(k in 1:kk) {
  # Assign the first row as the most common digits
  accuracy[1,k] <- as.integer(names(perc[[k]]))
  # Assign the second row as theirs population in the respective cluster
  accuracy[2,k] <- as.integer(perc[[k]][[1]])
  # Assign the third row as the whole population of the respective cluster
  accuracy[3,k] <- as.integer(length(myGroup[[k]]))
  # Assign the forth row as the mis-categorized rate
  accuracy[4,k] <- 1 - (accuracy[2,k] / accuracy[3,k])
}

mixrate <- 1 - sum(accuracy[2,]) / sum(accuracy[3,])

# ---------------------------------------------- #

> accuracy
            [,1]        [,2]         [,3]        [,4]     [,5]        [,6]
[1,]    3.0000000   8.0000000   4.00000000   9.0000000   6.0000   0.0000000
[2,]  138.0000000  89.0000000 129.00000000 108.0000000 120.0000  70.0000000
[3,]  226.0000000 196.0000000 140.00000000 221.0000000 128.0000 101.0000000
[4,]    0.3893805   0.5459184   0.07857143   0.5113122   0.0625   0.3069307
            [,7]    [,8]        [,9] [,10]
[1,]    2.0000000   1.000   7.0000000     0
[2,]  114.0000000  91.000 119.0000000    86
[3,]  179.0000000 104.000 212.0000000    86
[4,]    0.3631285   0.125   0.4386792     0

> mixrate
[1] 0.3320778
```

accuracy.r

That's all for solving this problem.

3 R CODE

```r
# ----------------------------------------------------- #
#
# ISyE 6740 - Take Home Exam #1
# Author: Steve Hongzhang Shao
# GT ID: 903071021
#
# ----------------------------------------------------- #

# Clear memory, include libraries
rm(list=ls())
library(mvtnorm)

# Read handwritten digits data
myData=read.csv("semeion.csv",header=FALSE)

# Build data matrix with (thresholded) pixel and label data
myX=data.matrix(myData[,1:256])
myLabel=apply(myData[,257:266],1,function(xx){
  return(which(xx=="1")-1)
})

# Define global variables
ii <- dim(myX)[1]
dd <- dim(myX)[2]
kk <- 10
qq <- 4
niter <- 40

# ------------------------ Initialize ------------------------ #

# Cluster data by using k(10)-means methodS
myCluster <- kmeans(myX, 10, iter.max = 20, nstart = 10)

# Initialization: assignments of data
gamma <- matrix(0, nrow=ii, ncol=kk)
for(i in 1:ii) {gamma[i, myCluster$cluster[i]] <- 1}
mu <- matrix(0, nrow=kk, ncol=dd)
pi <- rep(0,10)
# Likelihood
like <- rep(0, niter)

# Initialization of covariance matrices
sigma <- array(dim=c(10,256,256))
px <- matrix(0,ii,kk)

# ----------------------------------------------------- #

# Iterations
for(it in 1:niter) {

  # -------------------------- M step -------------------------- #
  # Initialize / Recompute mu, which is the group average.
  for(k in 1:kk) {
    muk <- rep(0,256)
    for(i in 1:ii) {muk <- muk + gamma[i,k] * myX[i,] }
    mu[k,] <- muk/sum(gamma[,k])
  }
  # Initialize / Recompute pi, which is the group fraction of data points
  pi <- colSums(gamma)/ii

  # Initialize / Recompute pi, which is the group fraction of data points
  for(k in 1:kk) {
    covk <- matrix(0,256,256)
    # Calculate cov matrix for spectral decomposition
    for(i in 1:ii) {
      xiBar <- myX[i,]-mu[k,]
      covki <- (xiBar %*% t(xiBar)) * gamma[i,k]
      covk <- covk + covki
    }
    # PCA part
    covk <- covk / sum(gamma[,k])
    eigk <- eigen(covk,symmetric=TRUE)
    pcak <- eigk$vectors[,1:qq]
    # Compute estimation of variance matrix
    deltak <- sum(eigk$values[qq+1:dd], na.rm = T) / (dd-qq)
    diagk <- diag(qq)
    for(q in 1:qq) {diagk[q,q] <- sqrt(eigk$values[q]-deltak)}
    # Compute estimation of variance matrix - continue
    wq <- pcak %*% diagk
    sigma[k,,] <- wq %*% t(wq) + (deltak * diag(dd))
  }

#   # Initialize / Recompute gamma when qq=0
#   for(k in 1:kk) {
#     covk <- matrix(0,256,256)
#     # Calculate cov matrix for spectral decomposition
#     for(i in 1:ii) {
#       xiBar <- myX[i,] - mu[k,]
#       covki <- (xiBar %*% t(xiBar)) * gamma[i,k]
#       covk <- covk + covki
#     }
#     # PCA part
#     covk <- covk / sum(gamma[,k])
#     eigk <- eigen(covk,symmetric=TRUE)
#     # Compute estimation of variance matrix
#     deltak <- sum(eigk$values[qq+1:dd], na.rm = T) / (dd-qq)
#     sigma[k,,] <- (deltak * diag(dd))
#   }

  # ------------------------- E step ------------------------- #

  for(k in 1:kk) {px[,k] <- pi[k]*dmvnorm(myX, mu[k,], sigma[k,,], log = FALSE)}
  for(i in 1:ii) {for(k in 1:kk) { gamma[i,k] <- px[i,k] / sum(px[i,]) }}
  # ---------------------- Likelihood ---------------------- #

  like[it] <- sum(log(rowSums(px)))
  print(c('loop','no.',it,'log','likelihood','is',like[it]))

}

# ---------------------- Convergence ---------------------- #

# Compute AIC
AIC <- -2*like[niter] + 2*(dd*qq + 1 - qq*(qq-1))
# Plot convergence
par(mai=c(0.5,0.5,0.5,0.5),mfrow=c(2,2))
dev.new(width=7,height=3.5)
plot(seq(1:niter),like)
title("Likelihood under q=4")

# --------------------- Visualization --------------------- #

# Print some pictures
dev.new(width=7,height=3.5)
par(mai=c(0.05,0.05,0.05,0.05),mfrow=c(10,6))
for(k in 1:kk){
  image(t(matrix(mu[k,], byrow=TRUE,16,16)[16:1,]),col=gray(0:1),axes=FALSE)
  box()
  for(j in 1:5){
    tempX <- rmvnorm(1, mean <- mu[k,], sigma[k,,])
    image(t(matrix(tempX, byrow=TRUE,16,16)[16:1,]),col=gray(0:1),axes=FALSE)
    box()
}}

# ----------------------- Accuracy ----------------------- #

# Find most common digits for each cluster
normLabel <- rep(0,ii)
for(i in 1:ii) {normLabel[i] <- which.max(gamma[i,])}
myGroup <- split(myLabel, normLabel)
perc <- lapply(myGroup, function(group){
  return(sort(table(group), decreasing=TRUE)[1])
})
accuracy <- matrix(0,4,10)

# Assign the most common digits, their populations, cluster populations and rates
for(k in 1:kk) {
  accuracy[1,k] <- as.integer(names(perc[[k]]))
  accuracy[2,k] <- as.integer(perc[[k]][[1]])
  accuracy[3,k] <- as.integer(length(myGroup[[k]]))
  accuracy[4,k] <- 1 - (accuracy[2,k] / accuracy[3,k])
}
mixrate <- 1 - sum(accuracy[2,]) / sum(accuracy[3,])
```

main.r