ISyE6740 Take Home Exam 1 – Spring 2016

ISyE6740 – Computational Data Analytics

Presented by: Damien Thioulouse

Presented to: Ben Haaland

February 5, 2016

Introduction

In this homework, we will cluster handwritten digits using R's k-means algorithm and the Expectation Maximization algorithm with a principal component selection within each maximization step.

For more clarity and efficiency, I did my best to avoid the loops and used matrices and vectors as much as I could.
Since the algorithm is not deterministic, I ran it twice for better results.
The total codes ran in approximately 30 minutes for a total of 180 iterations.

Initialization

First, we import the data and perform a preliminary clustering using R's kmeans algorithm for 10 clusters with 10 random starts:

```
##### Data Import
myData=read.csv("semeion.csv",header=FALSE) # Read handwritten digits
data
Y=data.matrix(myData[,1:256]) # Build data matrix with (thresholded)
pixel
myLabel=apply(myData[,257:266],1,function(xx){return(which(xx=="1")-
1)}) #Get Label for accuracy check
N=dim(Y)[1]
Col=dim(Y)[2]
K=10

#### kmeans with several random starts for preliminary clustering
fit <- kmeans(Y, K,nstart = 10)
```

For each q the function γ_{ik} is initialized using this k means clustering: $\gamma_{ik} = 1$ if observation i is assigned to cluster k and $\gamma_{ik} = 0$ otherwise.

```
  gam=matrix(0,N,K)
  for (i in 1:N){ gam[i,fit$cluster[i]]=1 }
```

From this function γ_{ik}, follows:

The parameters μ_k and π_k are initialized as $\pi_k = \frac{N_k}{n}$ where $N_k = \sum_{i=1}^{n} \gamma_{ik}$ and $\mu_k = \frac{1}{N_k}\sum_{i=1}^{n} \gamma_{ik} \boldsymbol{X_i}$ which is the same as the mean of the k clusters given by the R function k-means:

```
  pik=colSums(gam)/N
  mu=fit$centers
```

`mu` is a $10 * 256$ matrix gathering all the μ_k.
`pik` as a list gathering all the π_k.

Σ_k is calculated from these values of μ_k and π_k as follows:

$$\Sigma_k = W_q W_q' + \sigma^2 I_d$$

Where:

$$W_q = V_q diag\left\{\sqrt{\lambda_1 - \sigma^2}, \dots, \sqrt{\lambda_p - \sigma^2}\right\}, \sigma^2 = \frac{1}{d-q}\sum_{i=q+1}^{d} \lambda_i$$

And V_q are the first q eigenvectors of the spectral decomposition $\boldsymbol{V} diag\{\lambda_1, \dots, \lambda_d\}\boldsymbol{V}' = \frac{1}{N_k}\sum_{i=1}^{n} \gamma_{ik} (\boldsymbol{X_i} - \mu_k)(\boldsymbol{X_i} - \mu_k)', \lambda_1 \geq \cdots \geq \lambda_d$.

For more clarity, I coded a function `ComputeSigma` to calculate Σ_k:

```
sigmaList=ComputeSigma(fit,gam,Y,mu,q,K)
```

`sigmaList` is a list gathering the k $256 * 256$ matrices for all the clusters.

The function is defined as:

```
#### SIgma Calculation
ComputeSigma<-function(fit,gam,Y,mu,q,K) {
  sigmaList=vector("list",K)

  for (k in 1:K) {
    covmatrix=matrix(0,Col,Col)
    for (i in 1:N){
      covmatrix=covmatrix + gam[i,k]*(Y[i,]-mu[k,])%*%t(Y[i,]-mu[k,])
    }
    covmatrix=covmatrix/sum(gam[,k])
    eigVal=eigen(covmatrix,symmetric=TRUE)
    sigma2Small=1/(Col-q)*sum(eigVal$values[(q+1):Col])
    if (q>0){
      W=eigVal$vectors[,1:q]%*%diag((eigVal$values[(1:q)]-
sigma2Small)^(1/2))
      SigmaBig=W%*%t(W)+sigma2Small*diag(Col)
    }
    else {
      SigmaBig=sigma2Small*diag(Col)
    }
    sigmaList[[k]]=SigmaBig
  }
  sigmaList
}
```

I also initialize the objects that will store these parameters for each q:

```
PCtested=c(0,2,4,6) #list of numbers of PC that we want to try
listMuQ=vector("list")
listAICQ=matrix(0,length(PCtested),1)
listSigmaQ=vector("list")
listloglikeQ=vector("list")
```

Convergence

I first calculate the probability P using the initial values of μ_k, $\pi_k$and Σ_k. I use it to calculate the log-likelihood of after the initialization step.

At each iteration, a new gamma is calculated and μ_k, $\pi_k$and Σ_k are updated as explained in the initialization part. The log-likelihood is also computed at the end of each step and the loops stops when the difference of log-likelihood between 2 steps is inferior to 0.0005.

```
  Prob=ComputeProb(Y,K,N,mu,sigmaList)
  listloglike=c(ComputeLoglike(pik,Prob)) #init
  conv=1
  iter = 0

  #Iteration for a given q
while (iter <1 | conv>0.0005) { #iter<1 to be consistent with the first
steps
    iter = iter + 1

    gam=ComputeGamma(Y,pik,sigmaList,mu,Prob)#Compute Gamma

    pik=ComputePik(fit,gam,N) #Update Pi

    mu=ComputeMu(fit,Y,gam,N,Col) #Update Mu

    sigmaList=ComputeSigma(fit,gam,Y,mu,q,K) #Update Sigma

    Prob=ComputeProb(Y,K,N,mu,sigmaList) #Calculate probability p

    loglike_new=ComputeLoglike(pik,Prob) #Calculate loglike for this
step

    conv= loglike_new-listloglike[length(listloglike)] #Difference with
previous loglike

    listloglike[iter]=loglike_new #Add loglike to list
  }
```

All the functions are defined as follows:

```
#### Compute Gaussian Probability
ComputeProb<-function(Y,K,N,mu,sigmaList){
  prob1=matrix(0,K,N)
  for (k in 1:K){ prob1[k,]=dmvnorm(Y, mu[k,], sigmaList[[k]])}
  prob1
}
```

```
#### Gamma Calculation
ComputeGamma<-function(Y,pik,sigmaList,mu,prob1){
  gam=matrix(0,N,K)
  for (i in 1:N){
    for (k in 1:K){
      gam[i,k]=prob1[k,i]*pik[k]/sum(diag(pik)%*%prob1[,i]) #Compute
Gamma
```

```
    }
  }
  gam
}
```

```
#### Pi Calculation
ComputePik<-function(fit,gam,N){
  colSums(gam)/N
}
```

```
#### Mu Calculation
ComputeMu<-function(fit,Y,gam,N,Col){
  mu=matrix(0,K,Col)
  for (k in 1:K){
    muk=0
    for (i in 1:N){
      muk = muk + gam[i,k]*Y[i,]
    }
    muk = muk/sum(gam[,k])
    mu[k,]=muk

  }
  mu
}
```

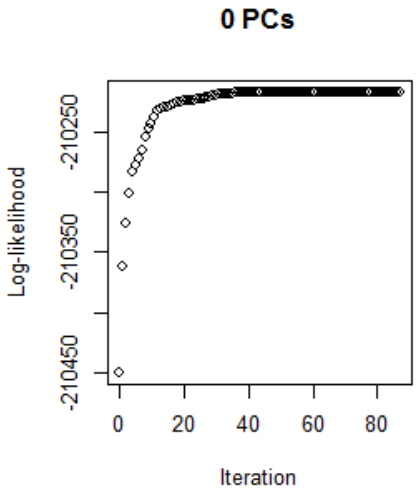
```
####Log-likelihood Calculation
ComputeLoglike<-function(pik,prob1){
  loglike=sum(log(colSums(diag(pik)%*%prob1))) #Compute loglikelihood
}
```

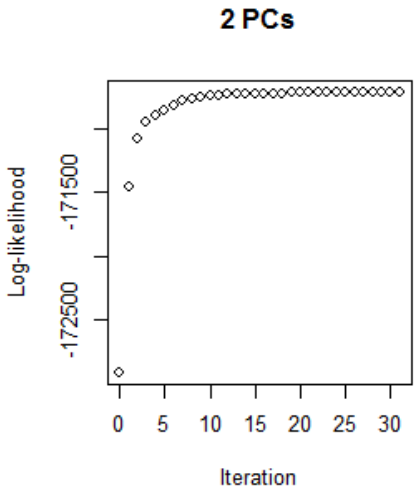
The initialization and the loop are ran for each q. The results are gathered in lists and plotted as follows:

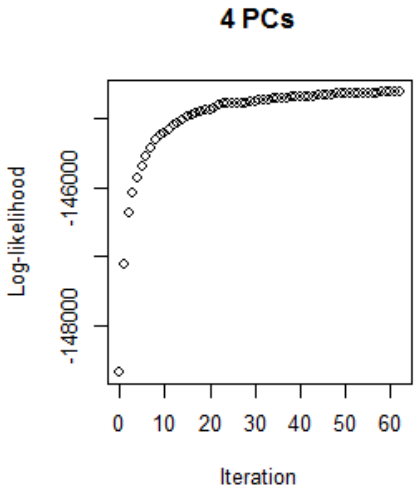
```
  listloglikeQ[[qindx]]=listloglike
  listMuQ[[qindx]]=mu
  listSigmaQ[[qindx]]=sigmaList

plot(c(1:length(listloglikeQ[[qindx]])),listloglikeQ[[qindx]],main=past
e(toString(q),"PCs",sep=" "),
            ylab="Log-likelihood", xlab="Iteration")

    qindx=qindx+1
```

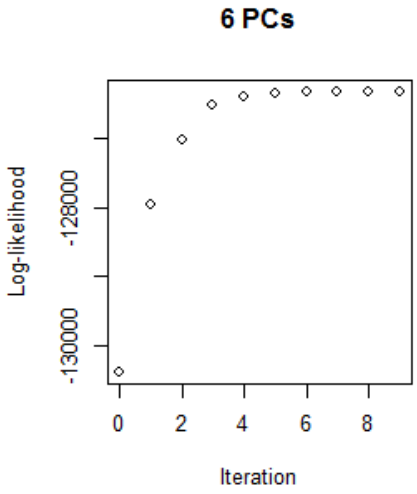
The code has been run twoce: the first time, it ran in `1881.61` seconds (31min) and the second one ran in `1560` seconds (25 min). The outputs are shown below.

First run:

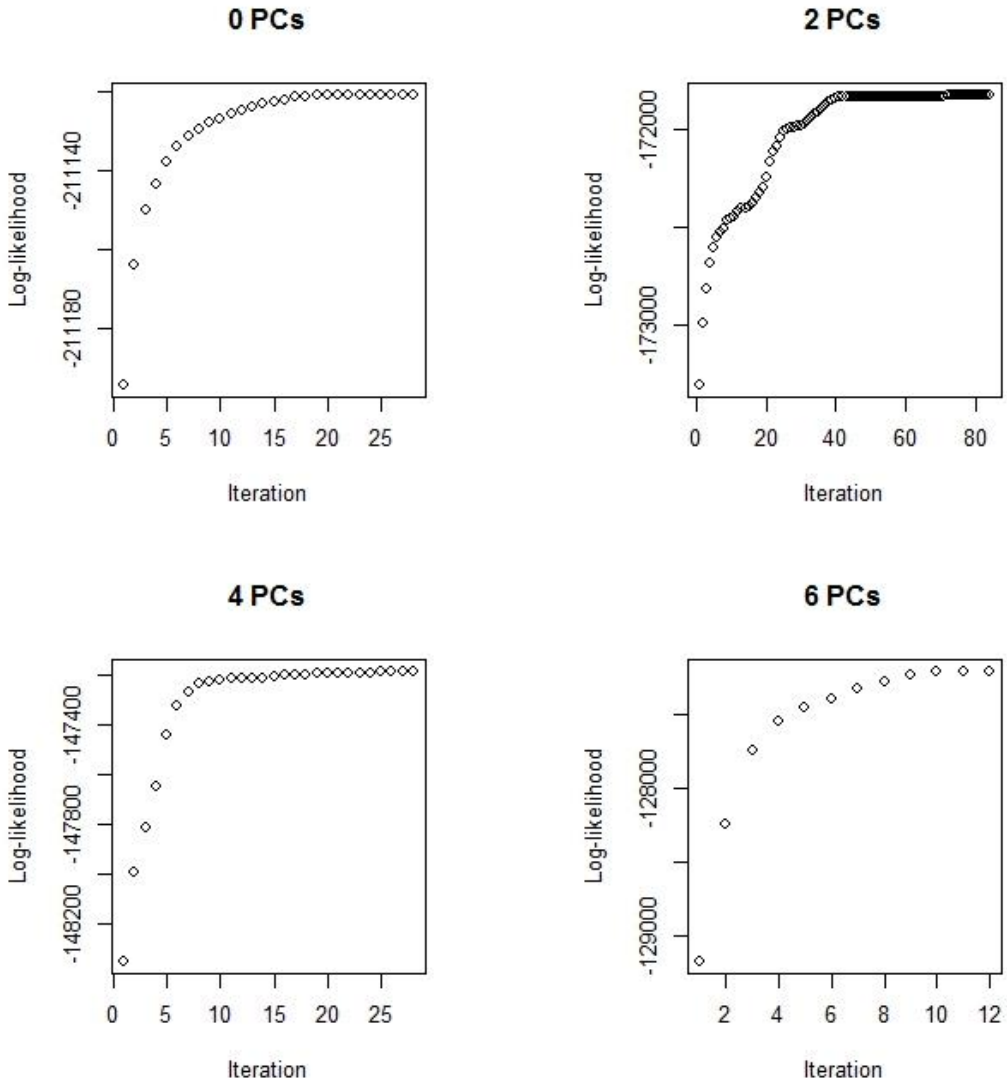

0 PCs
Log-likelihood
-210450
-210350
-210250
0
20
40
60
80
Iteration


2 PCs
Log-likelihood
-172500
-171500
0
5
10
15
20
25
30
Iteration


4 PCs
Log-likelihood
-148000
-146000
0
10
20
30
40
50
60
Iteration


6 PCs
Log-likelihood
-130000
-128000
0
2
4
6
8
Iteration

Second run:



These plots are not sufficient to show a trend in the convergence speed. Indeed, this algorithm not being deterministic, the number of iterations for each q is highly depending on the random seed.

However, from these two plots, we can say that the log-likelihood increase with the number of principle components. This log-likelihood being approximately the same between each run.

Choice of Number of Principle Components

In order to choose the number of principal components. We calculates the AIC at convergence:

```
  AIC=AICfun(listloglike[length(listloglike)],q,Col)
  listAICQ[qindx]=AIC
```

using the function defined as:

```
AICfun<-function(loglike,q,Col){-2*loglike+2*(Col*q+1-q*(q-1)/2)}
```

Which gives us the following results:

```
         [,1]
[1,] 420434.9
[2,] 342459.5
[3,] 291258.3
[4,] 255716.9
```

```
qBestIndx=which.min(listAICQ) ## Get Best q index
```

Since we want to minimize the AIC, we will choose $q = 6$ as the number of principal components.

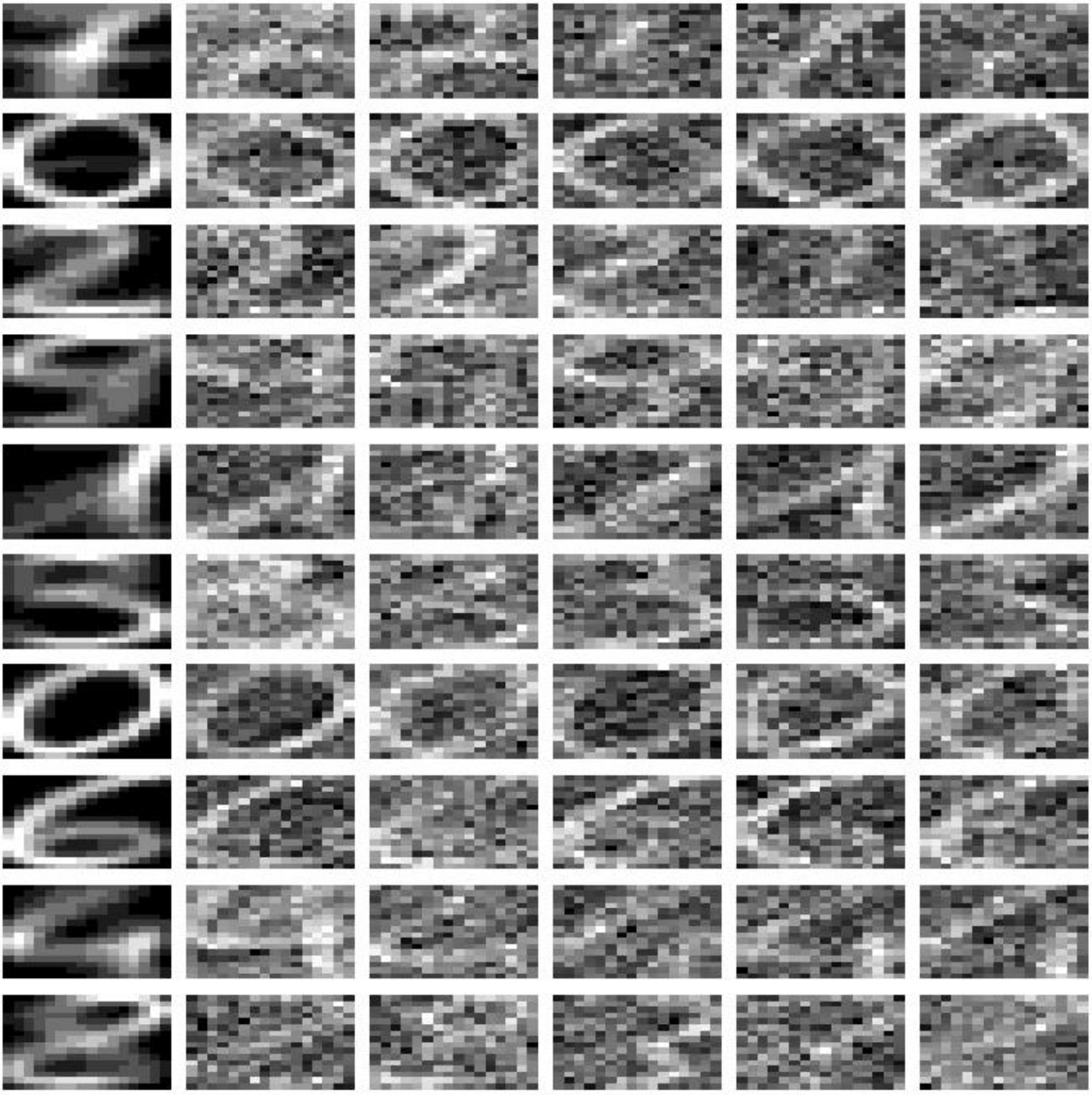
Visualization of the Clusters

For each k, we plot the cluster means corresponding to $q = 6$ and we generate 5 new images:

```
###### Vizualization of Clusters
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 1:10){
  image(t(matrix(listMuQ[[qBestIndx]][k,],byrow=TRUE,16,16)[16:1,]),col
=gray(seq(0,1,length=10)),axes=FALSE)
  for (r in 1:5){

random=mvrnorm(1,matrix(listMuQ[[qBestIndx]][k,]),listSigmaQ[[qBestIndx
]][[k]])
    image(t(matrix(random,byrow=TRUE,16,16)[16:1,]),col
=gray(seq(0,1,length=1000)),axes=FALSE)
  }
}
```

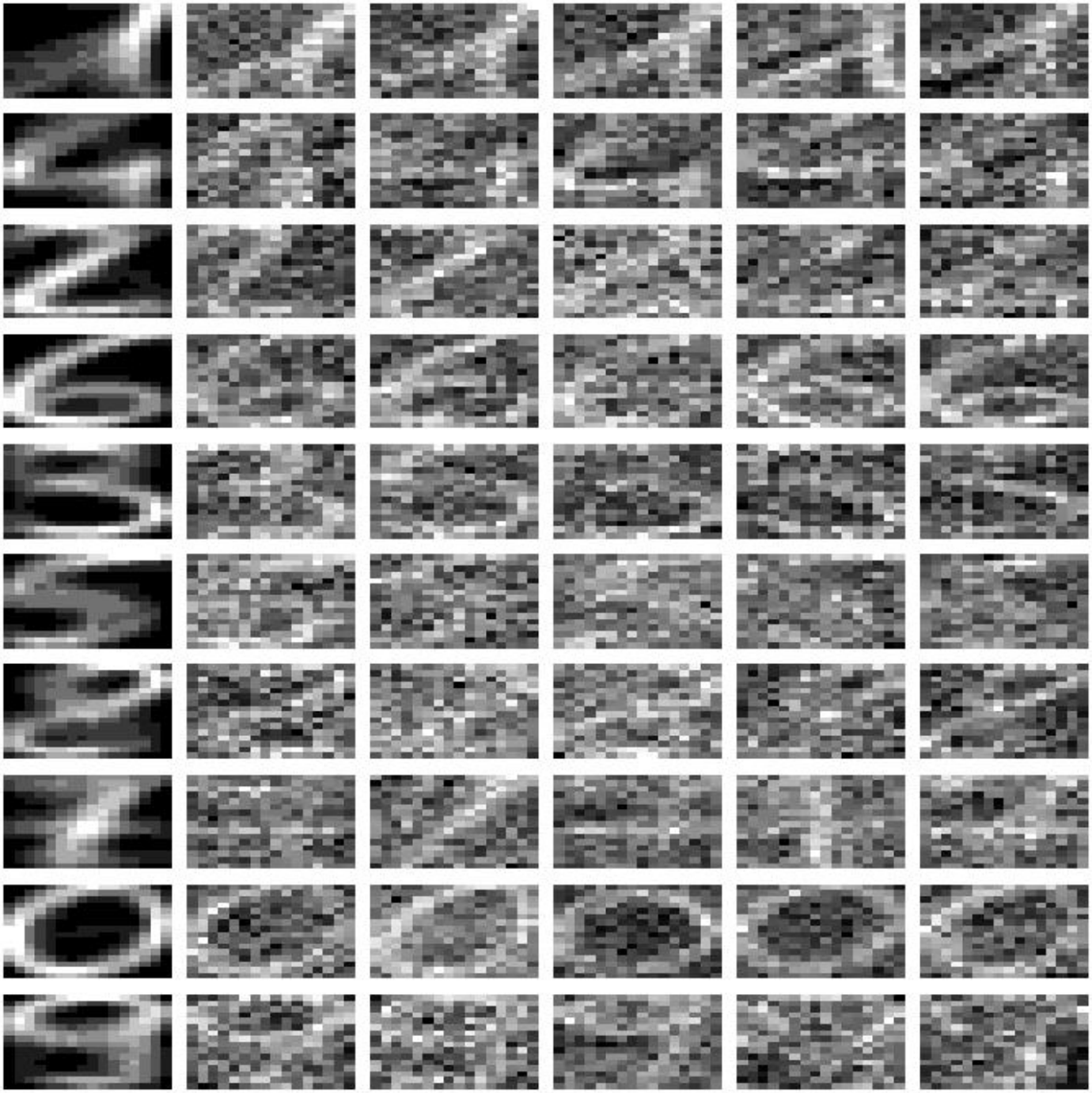
The results are shown in the next page.

First run:



The cluster means on the left definitely look like digits. However, two of them are similar to “0” and it is difficult to identify the “5” and “9”.

Second run:



In this case, we can distinct easily the 10 digits.

The random draws are difficult to identify but are similar to the cluster means.

Accuracy Assessment

In order to assess the accuracy, I assigned each observation to the most probable cluster. For each class label, I counted the mis-categorized observation (i.e that were not the most common observation in the cluster).

```
#### Accuracy Assessment
Acc=ComputeAccuracy(gam,N,myLabel)
Acc[[1]] #mis-categorization rate for each label class
Acc[[2]] #overall mis-categorization rate
```

With the function `ComputeAccuracy` defined as:

```
ComputeAccuracy<-function(gam,N,myLabel){
  Cat=vector()
  for (a in 1:N){Cat[a]=which.max(gam[a,])} #get cluster with maximum
probability for each element
  missClassMat=matrix(0,10,3)
  OverallMissCount=0
  i=1
  for (label in unique(myLabel)){
    LabelCat=table(Cat[myLabel==label]) #Count occurence of clusters
for each digit
    MissCat=sum(LabelCat)-max(LabelCat) #COunt the observation not in
the most common category
    MissCatRate=MissCat/sum(myLabel==label) #divide by the number of
occurences of each digit
    missClassMat[i,]= c(label,MissCat,MissCatRate)
    OverallMissCount=OverallMissCount+MissCat #count overall mis
categorization
    i=i+1
  }
  Overallrate=OverallMissCount/length(myLabel)
    list(missClassMat,Overallrate)
}
```

For each class label, the mis-categorization rate is in column 3 below. (The column 2 represents the number of mis-categorized observations.)

First run:

```
      [,1] [,2]       [,3]
 [1,]    0   75 0.46583851
 [2,]    1   72 0.44444444
 [3,]    2   50 0.31446541
 [4,]    3   15 0.09433962
 [5,]    4   26 0.16149068
 [6,]    5   82 0.51572327
 [7,]    6   42 0.26086957
 [8,]    7   25 0.15822785
 [9,]    8   38 0.24516129
[10,]    9   72 0.45569620
```

The mis-categorization rate is high for the digits we couldn't identify ("5" and "9") and that appeared twice ("0")

The overall mis-categorization rate is:
```
[1] 0.31199
```

Second run:

```
      [,1] [,2]      [,3]
 [1,]    0    5 0.0310559
 [2,]    1   72 0.4444444
 [3,]    2   73 0.4591195
 [4,]    3   17 0.1069182
 [5,]    4   27 0.1677019
 [6,]    5   64 0.4025157
 [7,]    6   37 0.2298137
 [8,]    7   35 0.2215190
 [9,]    8   56 0.3612903
[10,]    9   54 0.3417722
```

The overall mis-categorization rate is:
```
[1] 0.2762084
```

The overall mis-categorization rate is lower than for the previous run which is coherent with the fact that we could identify all the digits without difficulties.

Full Code

```
ptm <- proc.time()
setwd("~/GT/Spring 2016/ISYE6740 Computational Data Analysis/R code")
library("mvtnorm")
library("MASS")

#set.seed(231)

################################## Function Definitions

#### Compute Gaussian Probability
ComputeProb<-function(Y,K,N,mu,sigmaList){
  prob1=matrix(0,K,N)
  for (k in 1:K){ prob1[k,]=dmvnorm(Y, mu[k,], sigmaList[[k]])}
  prob1
}


#### Gamma Calculation
ComputeGamma<-function(Y,pik,sigmaList,mu,prob1){
  gam=matrix(0,N,K)
  for (i in 1:N){
    for (k in 1:K){
      gam[i,k]=prob1[k,i]*pik[k]/sum(diag(pik)%*%prob1[,i]) #Comoute
Gamma
    }
  }
  gam
}

#### SIgma Calculation
ComputeSigma<-function(fit,gam,Y,mu,q,K) {
  sigmaList=vector("list",K)

  for (k in 1:K) {
    covmatrix=matrix(0,Col,Col)
    for (i in 1:N){
      covmatrix=covmatrix + gam[i,k]*(Y[i,]-mu[k,])%*%t(Y[i,]-mu[k,])
    }
    covmatrix=covmatrix/sum(gam[,k])
    eigVal=eigen(covmatrix,symmetric=TRUE)
    sigma2Small=1/(Col-q)*sum(eigVal$values[(q+1):Col])
    if (q>0){
      W=eigVal$vectors[,1:q]%*%diag((eigVal$values[(1:q)]-
sigma2Small)^(1/2))
      SigmaBig=W%*%t(W)+sigma2Small*diag(Col)
    }
    else {
      SigmaBig=sigma2Small*diag(Col)
    }
    sigmaList[[k]]=SigmaBig
  }
  sigmaList
}
```

```
#### Pi Calculation
ComputePik<-function(fit,gam,N){
  colSums(gam)/N
}

#### Mu Calculation
ComputeMu<-function(fit,Y,gam,N,Col){
  mu=matrix(0,K,Col)
  for (k in 1:K){
    muk=0
    for (i in 1:N){
      muk = muk + gam[i,k]*Y[i,]
    }
    muk = muk/sum(gam[,k])
    mu[k,]=muk

  }
  mu
}

####Log-likelihood Calculation
ComputeLoglike<-function(pik,prob1){
  loglike=sum(log(colSums(diag(pik)%*%prob1))) #Compute loglikelihood
}

#### AIC Calculation
AICfun<-function(loglike,q,Col){-2*loglike+2*(Col*q+1-q*(q-1)/2)}

#### Accuracy Assessment
ComputeAccuracy<-function(gam,N,myLabel){
  Cat=vector()
  for (a in 1:N){Cat[a]=which.max(gam[a,])} #get cluster with maximum
probability for each element
  missClassMat=matrix(0,10,3)
  OverallMissCount=0
  i=1
  for (label in unique(myLabel)){
    LabelCat=table(Cat[myLabel==label]) #Count occurence of clusters
for each digit
    MissCat=sum(LabelCat)-max(LabelCat) #COunt the observation not in
the most common category
    MissCatRate=MissCat/sum(myLabel==label) #divide by the number of
occurences of each digit
    missClassMat[i,]= c(label,MissCat,MissCatRate)
    OverallMissCount=OverallMissCount+MissCat #count overall mis
categorization
    i=i+1
  }
  Overallrate=OverallMissCount/length(myLabel)
    list(missClassMat,Overallrate)
}
######################## End of the function Definition#########

############## Main

##### Data Importation
```

```
myData=read.csv("semeion.csv",header=FALSE) # Read handwritten digits
data
Y=data.matrix(myData[,1:256]) # Build data matrix with (thresholded)
pixel
myLabel=apply(myData[,257:266],1,function(xx){return(which(xx=="1")-
1)}) #Get Label for accuracy check
N=dim(Y)[1]
Col=dim(Y)[2]
K=10

##### Initialization
PCtested=c(0,2,4,6) #list of numbers of PC that we want to try
listMuQ=vector("list")
listAICQ=matrix(0,length(PCtested),1)
listSigmaQ=vector("list")
listloglikeQ=vector("list")

#### kmeans with several random starts for preliminary clustering
fit <- kmeans(Y, K,nstart = 10)

dev.new(width=4,height=4)
par(mai=c(0.8,0.8,0.8,0.8),mfrow=c(2,2))

#########################################################################
#Loop for each Q
#########################################################################
qindx=1
for (q in PCtested){
  #Initialize Gamma, Pi and Sigma
  gam=matrix(0,N,K)
  for (i in 1:N){ gam[i,fit$cluster[i]]=1 }
  pik=colSums(gam)/N
  mu=fit$centers
  sigmaList=ComputeSigma(fit,gam,Y,mu,q,K)
  Prob=ComputeProb(Y,K,N,mu,sigmaList)
  listloglike=c(ComputeLoglike(pik,Prob)) #init
  conv=1
  iter = 0

#########################################################################
  #Iteration for a given q
  while ( iter <1 | conv>0.0005) { #iter<1 to be consistent with the
first steps
    iter = iter + 1

    gam=ComputeGamma(Y,pik,sigmaList,mu,Prob)#Compute Gamma

    pik=ComputePik(fit,gam,N) #Update Pi

    mu=ComputeMu(fit,Y,gam,N,Col) #Update Mu

    sigmaList=ComputeSigma(fit,gam,Y,mu,q,K) #Update Sigma

    Prob=ComputeProb(Y,K,N,mu,sigmaList) #Calculate probability p

    loglike_new=ComputeLoglike(pik,Prob) #Calculate loglike for this
step
```

```
    conv= loglike_new-listloglike[length(listloglike)] #Difference with
previous loglike

    listloglike[iter]=loglike_new #Add loglike to list
  }
  AIC=AICfun(listloglike[length(listloglike)],q,Col)
  listAICQ[qindx]=AIC

  listloglikeQ[[qindx]]=listloglike
  listMuQ[[qindx]]=mu
  listSigmaQ[[qindx]]=sigmaList


plot(c(1:length(listloglikeQ[[qindx]])),listloglikeQ[[qindx]],main=past
e(toString(q),"PCs",sep=" "),
            ylab="Log-likelihood", xlab="Iteration")

    qindx=qindx+1
}
qBestIndx=which.min(listAICQ) ## Get Best q index

###### Vizualization of Clusters
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 1:10){
  image(t(matrix(listMuQ[[qBestIndx]][k,],byrow=TRUE,16,16)[16:1,]),col
=gray(seq(0,1,length=10)),axes=FALSE)
  for (r in 1:5){

random=mvrnorm(1,matrix(listMuQ[[qBestIndx]][k,]),listSigmaQ[[qBestIndx
]][[k]])
    image(t(matrix(random,byrow=TRUE,16,16)[16:1,]),col
=gray(seq(0,1,length=1000)),axes=FALSE)
  }
}

#### Accuracy Assessment
Acc=ComputeAccuracy(gam,N,myLabel)
Acc[[1]] #mis-categorization rate for each label class
Acc[[2]] #overall mis-categorization rate
proc.time() - ptm
```