Computational Data Analytics ISYE 6740

Homework 1

Arjun Mishra - 903230877

OVERVIEW

In this assignment, we are given a dataset comprising of 1593 handwritten digits which have been encoded in the form of Boolean variables based on the presence of a pixel in a 16x16 grayscale image. We are given the task of performing soft clustering on these images using the EM algorithm.

APPROACH AND ALGORITHM:

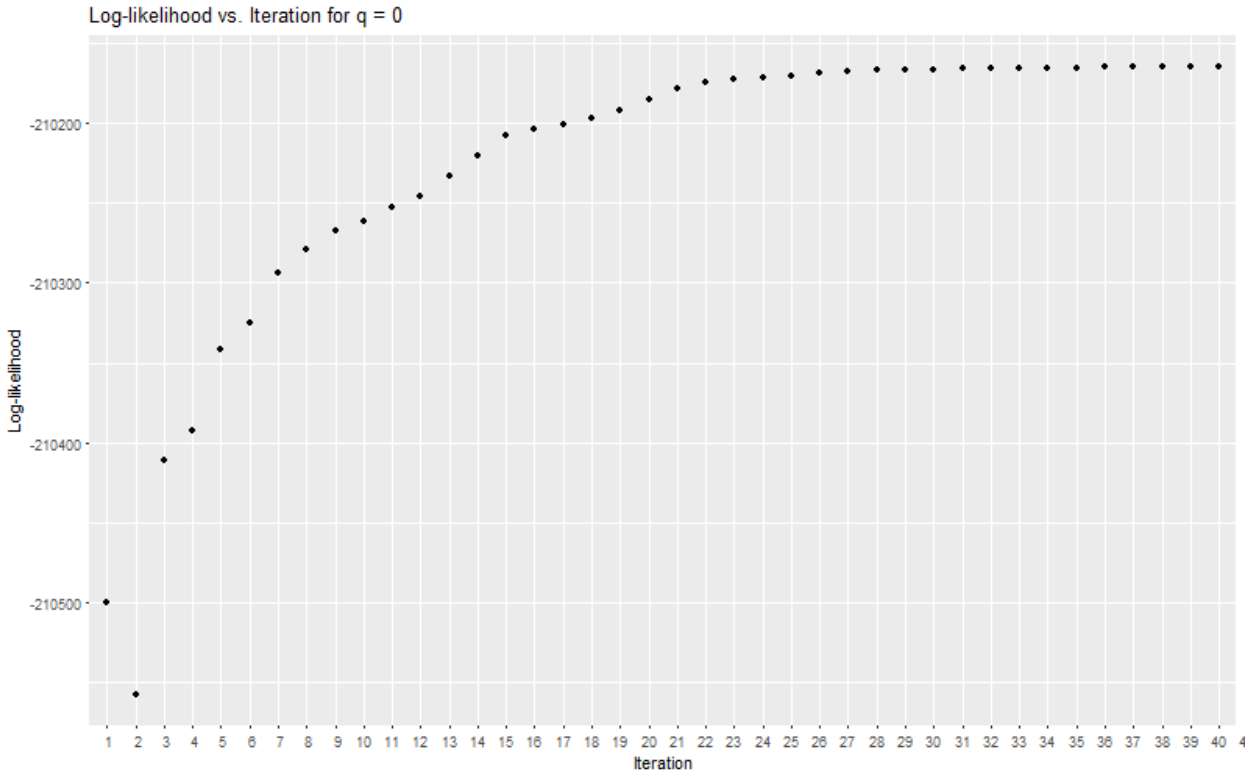
Some of the key steps taken for the algorithm are described below:

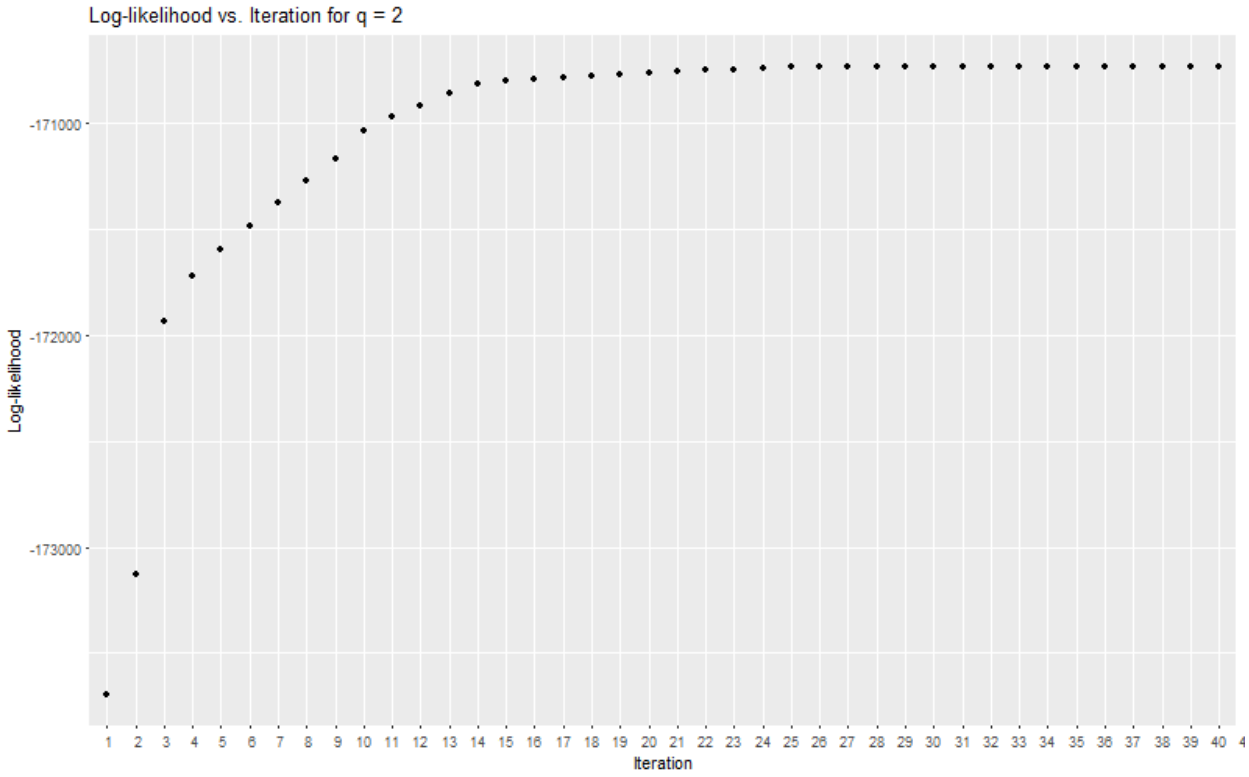
Initialization:

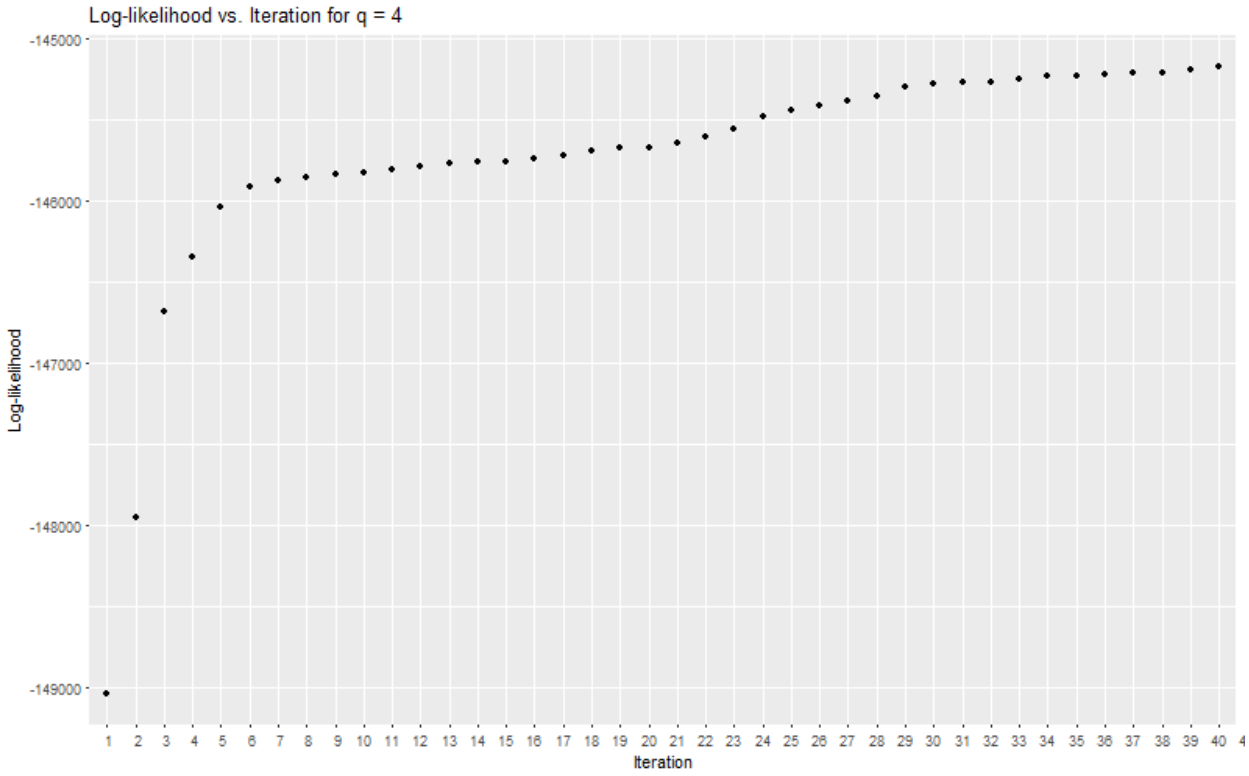
- The first steps consist of declaring the variables, matrices and lists to be used in the next steps of the algorithm. We encode some of the numbers which have to be used throughout in to variables. We also divide our given dataset into data and the actual cluster classification.
- The initialization step uses the kmeans function to perform an initial HARD clustering of the data. We obtain starting cluster centers and initial class membership (GAMMA) from this step.
- After this, we move on to the M-step of the EM algorithm. Here we use the class membership distribution to get the parameter estimates of the Gaussian distributions. These parameters consist of the mean vector and the point composition of each cluster. We also used spectral decomposition to obtain the principal components (which varied from 0,2,4 & 6). These principal components were finally used to compute a rank-q plus noise estimate for the covariance matrix **Σ** of the distributions. Due to the nature of 0 principal components, the covariance matrix was directly computed from the variance in that case.
- The estimates obtained in the M-step were used to calculate the prior and the posterior probabilities. The posteriors were used in the prior calculation which gave us the class membership distributions in terms of probabilities.
- Finally, we used all the calculations of the E and M steps to compute the log-likelihood. We used 40 iterations to arrive at the convergence of the algorithm.

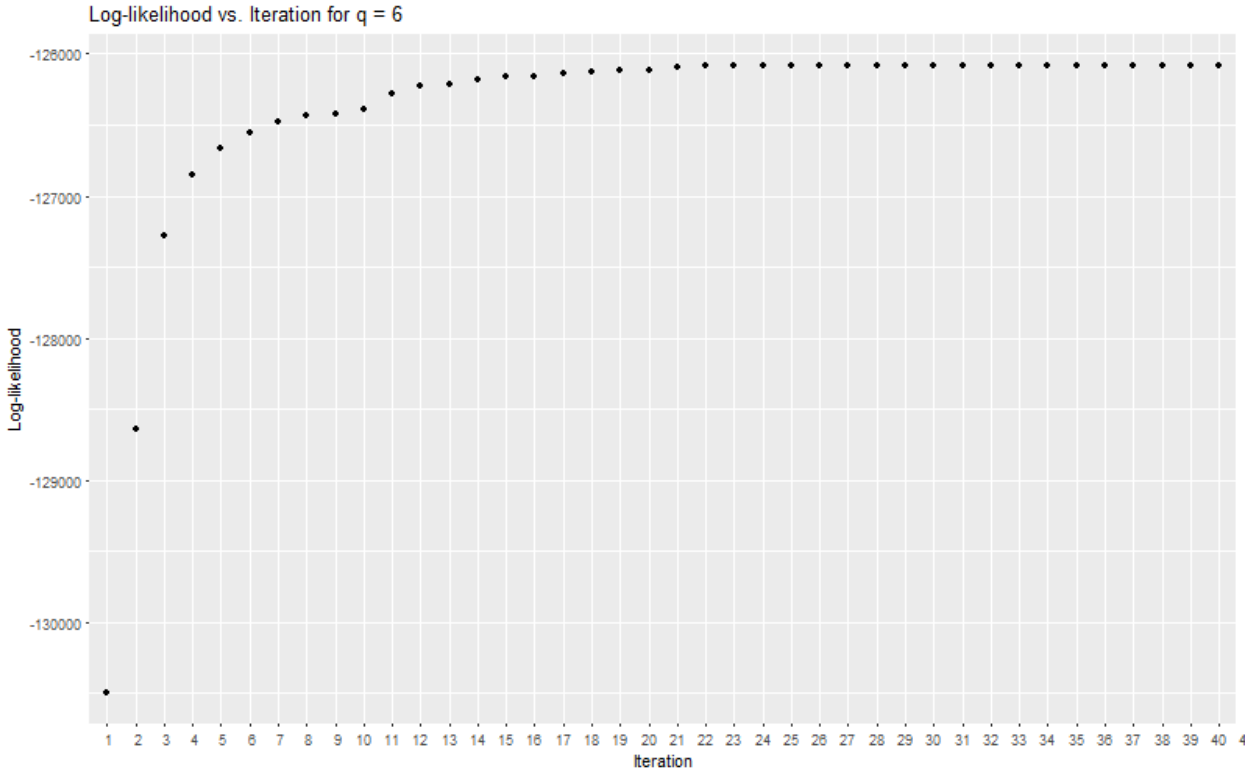
Convergence:

The convergence plots for the various values of q are given below:









From the plots, it is evident that convergence usually occurred at around 25+ iterations. We also observe that the value of the log likelihood keeps on increasing with an increase in the

value of number of principal components. Thus, the highest value and the best performance comes from choosing the value of q as 6.

Choice of Number of Principal Components:

The AIC value at convergence was used to select the best value of the number of principal components to be used. The AIC values for q = 0, 2, 4, 6 respectively are:

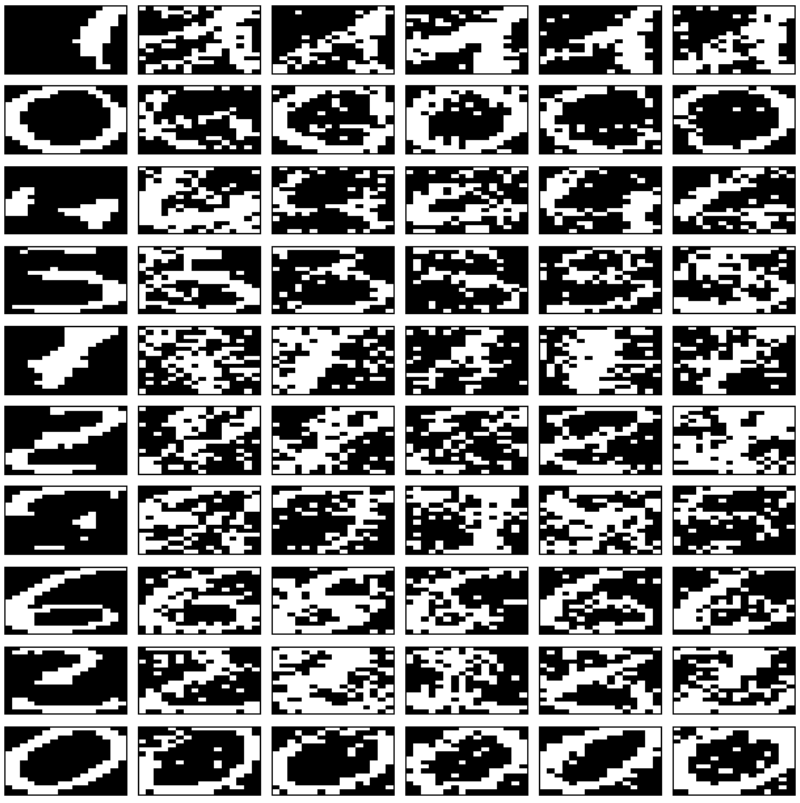
```
420331.1 342483.6 292380.9 255202.2
```

1. Q = 0 : 420331.1
2. Q = 2 : 342483.6
3. Q = 4 : 292380.9
4. Q = 6 : 255202.2

Thus, we see a steady decline in the AIC value with increasing q value and the lowest one occurs at q = 6.

Visualization of the Clusters:

In this section, we visualize 5 random entries from each cluster formed by using q value as 6 along with the cluster centers. This will allow to observe how well our clustering is performing and how well the cluster center is defined.



From the image, we can see that the centers are clear and less pixelated than the random images. The digit 0 is best represented out of all these – the center most resembles the data points. However, there is some overlap between the 6, 0 and 9 clusters. The 1,2 and 3 digit clusters also have moderate distinguishability. While their centers are not very well defined, the clusters contain correct classifications. According to me, the clustering is not very well done. This is because the individual components are not very clear and highly pixelated. It would have been better to have more number of principal components that we could have analyzed.

Accuracy Assessment

To evaluate the accuracy of our clustering, we make a table of the most occurring number in each of our clusters and then evaluate how many were classified correctly out of all the occurrences of the number.

For the table below, the first row represents the most occurring digit in the cluster, the second row gives us the correct classification, the third gives us the total number in that cluster and finally the fourth is a ratio of rows 2 and rows 3.

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1.0000000	0.0000000	4.00000000	3.000000	7.0000000	8.0000000	9.0000000	6.000	2.0000000	0.00000000
94.0000000	71.0000000	139.00000000	106.000000	130.0000000	118.0000000	77.0000000	119.000	106.0000000	86.00000000
117.0000000	108.0000000	147.00000000	243.000000	215.0000000	198.0000000	179.0000000	125.000	174.0000000	87.00000000
0.1965812	0.3425926	0.05442177	0.563786	0.3953488	0.4040404	0.5698324	0.048	0.3908046	0.01149425

We can see in this table that our clustering was not able to identify the cluster for the digit 5 and classified it as 0. The lowest mis-classification was observed for the digit 0 at 1.1%. Digits 4 and 7 were the most mis-classified out of all the clusters with the rate over 55%.

The overall mis-classification rate for the exercise came out as 34.3%.

R-CODE:

```
############################### HW1 Compuational Data Analysis ===================

############## Reading in dataset and installing some preliminary modules
#Set working directory
setwd("G:/Georgia Tech/Computational Data Analytics/Assignments")
#Loading packages
require(ggplot2)
require(data.table)
require(dplyr)
require(mvtnorm)
require(scales)


#Reading dataset
sem = fread("semeion.csv")
ncol(sem)
nrow(sem)
#1593 * 266 data table
#First 256 columns are the image data and the last 10 are the classification

images = subset(sem, select = c(1:256))

#Converting this to a matrix
#This will be 1593 x 256 matrix
X_data = data.matrix(data.frame(images))

#Clusters
clust_labels = subset(sem, select = c(257:266))

#Defining the dimensions and parameters in terms of variables
k = 10 #Number of clusters
n = 1593 #Number of data points
d = 256 #Number of Dimensions
q_list = c(0,2,4,6) #array of number of principal components
iterations = 40 #number of iterations


#Making the final lists to store the plotting variables
likelihood_list = vector(mode="list", length=length(q_list))
names(likelihood_list) = c("q0", "q2","q4","q6")

mu_list = vector(mode="list", length=length(q_list))
names(mu_list) = c("q0", "q2","q4","q6")

gamma_list = vector(mode="list", length=length(q_list))
names(gamma_list) = c("q0", "q2","q4","q6")

sigma_list = vector(mode="list", length=length(q_list))
names(sigma_list) = c("q0", "q2","q4","q6")
```

```
AIC_list = vector(length=length(q_list))


############################### Question 1 ================================
#Initializing the clusters
#K means clustering

#Will use 10 clusters as already known, 20 starting points for centers
# and 30 iterations for good convergence.
clust_model1 = kmeans(images, 10, iter.max = 30, nstart = 20)

#We have the initial cluster assignments
#We need to set Yik = 1 if it belongs to cluster k else 0
#Gamma matrix is going to hold the probabilities after soe iterations.
#It will have the probability that a data point (1593) will lie in a cluster.

#This is where we start the main loop of q's iterations

for (number in seq_along(q_list)){
q = q_list[number]
print(" ")
print(paste("Running loop wih q =", q))
#This is a n x k matrix - same as the labels matrix
gamma_matrix = matrix(0, nrow = nrow(clust_labels), ncol = ncol(clust_labels))

#Obtaining the cluster values
clusters = clust_model1$cluster

#Populating gamma matrix
for (i in 1:n){
  gamma_matrix[i, clusters[i]] = 1
}

#Next we need the matrix that will hold the centroid/mean of each cluster
#These means will also change as we iterate through the algorithm
#We will initialize it by the k-means centroids so that the estimate is good

#It is a k x d matrix (10 x 256)
mu_matrix = matrix(0, nrow = k, ncol = d)

#Mu matrix has to be calcuated by the matrix product of gamma with X_data and
#divided by Nk
#Initially it is equal to the centers given by the k-means
mu_matrix = clust_model1$centers

#We also need the pi vector which contains the proportion of points in a cluster
#We can get the proportion using the gamma matrix
pi = apply(gamma_matrix, 2, sum)/n

# Making the iterations list
likelihood <- rep(0, iterations)

# Initialization of covariance matrices
```

```
sigma <- array(dim=c(10,256,256))
px <- matrix(0, n, k)

################ Running the iteration loop with m and e step ==============

#M-step function
for (iter in 1:iterations){
 #We need a diagonal matrix with 1/Nk for each cluster.
 #This matrix gives us the inverse of the number of points in a cluster - 1/Nk
 point_composition_matrix = diag((pi*n)^-1)

 #Making a point composition vector
 point_composition_vector = (pi*n)^-1

 #Recalculating the mu_matrix
 mu_matrix = point_composition_matrix %*% t(gamma_matrix) %*% X_data

 #Recalculating pi
 pi = colSums(gamma_matrix)/n

 for(j in 1:k){
  #Initialize the covariance matrix
  cov_matrix = matrix(0,256,256)

  #Doing the spectral decomposition
  for(i in 1:n){
   cov_col = ((X_data[i,]-mu_matrix[j,]) %*%
                t((X_data[i,]-mu_matrix[j,]))) * gamma_matrix[i,j]
   cov_matrix = cov_matrix + cov_col
  }
  cov_matrix <- cov_matrix / sum(gamma_matrix[,j])
  #DOing the principal component analysis
  eigen_object = eigen(cov_matrix,symmetric=TRUE)

  #Computing the variance
  var_mat = sum(eigen_object$values[q+1:d], na.rm = T) / (d-q)

  #We need to put a condition on q = 0 as the sigma matrix fails to form
  if(q!=0) {
   prin_comp = eigen_object$vectors[,1:q]

   #Computing the diagonal eigen matrix
   diag_eigen = diag(q)
   for(i in 1:q) {
    diag_eigen[i,i] = sqrt(eigen_object$values[i]-var_mat)
   }

   #Computing rank-q plus noise estimate
   wq = prin_comp %*% diag_eigen
   #The sigma noise matrix is a 3 dimensional matrix
   sigma[j, , ] = wq %*% t(wq) + (var_mat * diag(d))

  }
```

```
   else {
     sigma[j, , ] = var_mat * diag(d)
   }
  }

  ################### Moving on to the E-step

  for(j in 1:k) {px[,j] = pi[j]*dmvnorm(X_data, mu_matrix[j,], sigma[j, , ], log = FALSE)}
  for(i in 1:n) {for(j in 1:k) { gamma_matrix[i,j] = px[i,j] / sum(px[i,]) }}

  ################### Calculating the likelihood

  likelihood[iter] <- sum(log(rowSums(px)))
  print(paste('The log-likelihood of loop no. ',iter,' is ',likelihood[iter]))

}

likelihood_list[[number]] = likelihood
gamma_list[[number]] = gamma_matrix
mu_list[[number]] = mu_matrix
sigma_list[[number]] = sigma

}


################ Plotting the various values of likelihood ================================

for (number in seq_along(q_list)) {

  final = as.data.frame(cbind(likelihood_list[[number]],c(1:iterations)))
  colnames(final) = c('ll','iteration')

  #Making the plots
  png(paste0('Log-likelihood for q = ',q_list[number],'.png'), width = 800, height = 500)

  #Using ggplot
  print(ggplot(final, aes(y=final$ll, x=final$iteration)) +
        geom_point() +
        labs(title = paste("Log-likelihood vs. Iteration for q =",q_list[number]),
             x = "Iteration", y = "Log-likelihood") +
        scale_x_discrete(limits=c(1:(iterations + 1))))
  dev.off()
}


############################ Calculating AIC to choose best q ===========================
for (number in seq_along(q_list)) {
  q = q_list[number]
  #calculate AIC
  AIC = -2*tail(likelihood_list[[number]],1) + 2*(d*q + 1 - (q*(q-1)/2))
  AIC_list[number] = AIC
  print(paste("The value of AIC for q = ", q, "is",round(AIC)))
}
```

```
print(paste('Best value of principal components should be chosen as',q_list[which.min(AIC_list)]))

########################### Visualizing the cluster ==================================
dev.new(width=7,height=3.5)
par(mai=c(0.05,0.05,0.05,0.05),mfrow=c(10,6))

for(i in 1:10){
  image(t(matrix(mu_list$q6[i,], byrow=TRUE,16,16)[16:1,]),col=gray(0:1),axes=FALSE)
  box()
  for(j in 1:5){
    tempX = rmvnorm(1, mean <- mu_list$q6[i,], sigma_list$q6[i,,])
    image(t(matrix(tempX, byrow=TRUE,16,16)[16:1,]),col=gray(0:1),axes=FALSE)
    box()
  }
}

########################### Accuracy assessment ======================================

# Alloting numbers to clusters
new_clusts = vector(length = n)

for(i in 1:n) {new_clusts[i] = which.max(gamma_list$q6[i,])}

#Getting the labels for the data points from the clust_labels
origLabel = apply(clust_labels,1,function(drow){return(which(drow=="1")-1)})

#Divinding the new labels according to the previous
clust_split = split(origLabel, new_clusts)
prop = lapply(clust_split, function(group){
  return(sort(table(group), decreasing=TRUE)[1])
})

# Initialize accuracy matrix
acc_matrix = matrix(0,4,10)

for(i in 1:10) {

  acc_matrix[1,i] = as.integer(names(prop[[i]]))
  acc_matrix[2,i] = as.integer(prop[[i]][[1]])
  acc_matrix[3,i] = as.integer(length(clust_split[[i]]))
  acc_matrix[4,i] = as.numeric(1 - (acc_matrix[2,i] / acc_matrix[3,i]))

}


acc_rate = 1 - sum(acc_matrix[2,]) / sum(acc_matrix[3,])

#Getting the mis classification rates
print("The mis-classification rates for each cluster are:")
print(percent(acc_matrix[4,]))
print(paste('The overall mis-classification rate is',percent(acc_rate)))
```