Assignment 7

Prepared by:

- Josh Levine (jl2108)
- Harsh Patel (hkp49)
- Jaini Patel (jp1891)
- Yifan Liao (yl1463)
- Aayush Shah (avs93)

Expectation-Maximization (EM) algorithm

The EM algorithm is used for obtaining maximum likelihood estimates of parameters when some of the data is missing. More generally, however, the EM algorithm can also be applied when there is unobserved data, which was never intended to be observed in the first place. In that case, we simply assume that the latent data is missing and proceed to apply the EM algorithm. The EM algorithm has many applications throughout statistics. It is often used for example, in machine learning and data mining applications, and in Bayesian statistics where it is often used to obtain the mode of the posterior marginal distributions of parameters.

EM is an iterative algorithm with two linked steps:

- E-step: fill-in hidden values using inference and
- M-step: apply standard MLE method to completed data.

EM algorithm always converges to a local optimum of the likelihood. It is mostly used for the multivariate distribution like Gaussian Mixture Models(GMM). A simple example of EM algorithm is K-means Clustering.

EM for Gaussian Mixture Model

Given a Gaussian mixture model, the goal is to maximize the likelihood function with respect to the parameters - mean and covariances of the components and the mixing coefficients.

The following are the steps to implement Expectation-Maximization (EM) algorithm on Gaussian Mixture Model(GMM):

1.) Initialize the μ_k's, σ_k's and π_k's and evaluate the log-likelihood with these parameters.

2.) E-step: Evaluate the posterior probabilities $\gamma_{Z_i}(k)$ using the current values of the μ_k's and σ_k's with equation

$$\gamma_{Z_i}(k) = \frac{\pi_k N(\mu_k, \sigma_k^2)}{\sum_{k=1}^{K} \pi_k N(\mu_k, \sigma_k)}$$

3.) M-step: Estimate new parameters $\hat{\mu_k}$, $\hat{\sigma_k^2}$ and $\hat{\pi_k}$ with the current values of $\gamma_{Z_i}(k)$ using equations,

$$\hat{\mu_k} = \frac{\sum_{i=1}^{n} \gamma_{z_i}(k) x_i}{\sum_{i=1}^{n} \gamma_{z_i}(k)} = \frac{1}{N_k} \sum_{i=1}^{n} \gamma_{z_i}(k) x_i$$

$$\hat{\sigma_k^2} = \frac{1}{N_k} \sum_{i=1}^{n} \gamma_{z_i}(k)(x_i - \mu_k)^2$$

$$\hat{\pi_k} = \frac{N_k}{n}$$

4.) Evaluate the log-likelihood with the new parameter estimates. If the log-likelihood has changed by less than some small ϵ, stop. Otherwise, go back to step 2.

Below are the code chunks and corresponding outputs of the functions used to perform k means clustering with EM algorithm.

"parameter_initialization" function initializes parameters (mean and covariance matrix).

```
parameter_initialization <- function(x, k, features){

  #mean matrix initialization
  mean_init = matrix(nrow = k, ncol = features)
  for(i in c(1:k)){
    for (j in c(1:features)){
      mean_init[i,j] = runif(n = 1, max = max(x[,j]), min = min(x[,j]))
    }
  }

  #weight initialization
  wt_init = c(0.5,0.1,0.4)

  #covariance matrix initialization
  cov_init = array(dim = c(features, features, k))
  for (i in (1:k)){
    cov_init[,,i] = genPositiveDefMat("eigen", dim = features)$Sigma
  }
  list(mean = mean_init, cov = cov_init, wt = wt_init)
}
```

"e" function does the E step of the algorithm.

```
e <- function(x,k,N,p_list){

  res_mat <- matrix(nrow = N, ncol = k)
  for (i in c(1:N)){
    for(j in c(1:k)){
      res_mat[i,j] = (p_list$wt[j]) * ( dmvn( as.numeric(x[i,]) ,
                                              t(p_list$mean[j,]), p_list$cov[,,j]))
    }
  }
  res_mat = res_mat/rowSums(res_mat)
  return(res_mat)
}
```

"m" function does the M step of the algorithm.

```
m <- function(x,res_mat, k, N, features){

  # mean update
  mean_updated = matrix(nrow = k, ncol = features)
  for(i in c(1:k)){
    mean_updated[i, ] = colSums(x*res_mat[,i])/sum(res_mat[,i])
  }

  # wt update
  wt_updated = apply(res_mat,2,sum)/10

  # covariance update
  cov_updated = array(NA, dim =c(features, features,k))
  for(i in c(1:k)){
    diff = sweep(x, 2, c(mean_updated[i,1], mean_updated[i,2]), "-")
    cov_updated[,,i] = t(diff)  %*%
      as.matrix(res_mat[,i] * diff )  / sum(res_mat[,i])
  }

  list(mean = mean_updated, cov = cov_updated,  wt = wt_updated)

}
```

"log_likelihood" function evaluates the likelihood.

```
#log likelihood
log_likelihood <- function(x, k, N, p_list){
  sum = vector()

  for (i in c(1:N)){
    temp = 0
    for(j in c(1:k)){
      temp = ( (p_list$wt[j]) * ( dmvn(as.numeric(x[i,]), t(p_list$mean[j,])
                                       , p_list$cov[,,j]) ) ) + temp
    }
    sum = c(sum, temp)
  }
  return(sum(log(sum)))
}
```

"em" function implements the em algorithm by performing e step, m step and likelihood on the data.

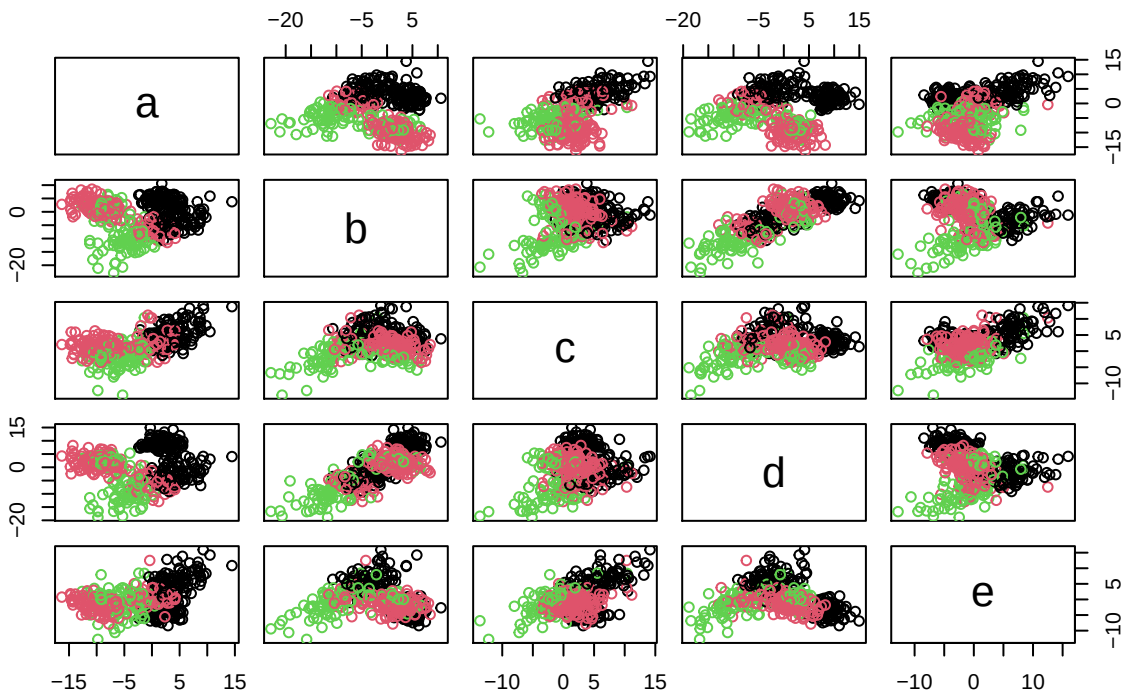
```
#Main function
em <- function(x, k=3){
```

```
  #no. of features
  features = ncol(x)
  #no. of data rows
  N = nrow(x)

  prev_log_likelihood = 0
  Delta = 5
  #Initialization of parameter list
  p_list = parameter_initialization(x, k, features)  # parameter list

  n_steps = 0
  delta_list = c(1)

  while(Delta>0.1)
  {

    print(paste(n_steps, "step"))
    n_steps = n_steps + 1

    #E-STEP - recalculates responsibility matrix based on new cluster assignments.
    res_mat = e(x, k, N, p_list)

    #M-step - p_list contains the updated mean and variance of the clusters
    p_list = m(x,res_mat, k, N, features)

    #log-likelihood
    new_log_likelihood = log_likelihood(x,k,N, p_list)

    #Delat is the difference between likelihood values.
    Delta = abs(new_log_likelihood - prev_log_likelihood)
    delta_list = c(delta_list, Delta)
    prev_log_likelihood = new_log_likelihood
    print(paste("Delta", Delta))

  }

  #final cluster
  final_cluster = res_mat
  final_cluster = cbind(final_cluster, apply(res_mat,1,which.max))
  final_cluster = cbind(x, final_cluster[,4]) %>% rename("cluster" = "final_cluster[, 4]")

  #visualization with pair-pair plot.
  with(x, pairs(x, col=final_cluster$cluster, main = "Pair-Pair Plots"))


  return(list(data_with_clusters = final_cluster, res_mat = res_mat,
              delta_list = delta_list, clustered_parameters = p_list))

}
```

```
# data
df <- read.csv("Downloads/kmeans.csv")
#no of clusters
```

```r
k = 3
# running em algo
output = em(df, k)
```

```
## [1] "0 step"
## [1] "Delta 4733.19836109562"
## [1] "1 step"
## [1] "Delta 29.3973455086925"
## [1] "2 step"
## [1] "Delta 5.49515558785606"
## [1] "3 step"
## [1] "Delta 3.34205306419699"
## [1] "4 step"
## [1] "Delta 3.68934041563261"
## [1] "5 step"
## [1] "Delta 4.44086688434982"
## [1] "6 step"
## [1] "Delta 4.99118007516336"
## [1] "7 step"
## [1] "Delta 5.19937760498578"
## [1] "8 step"
## [1] "Delta 5.1152203915517"
## [1] "9 step"
## [1] "Delta 4.80910832225618"
## [1] "10 step"
## [1] "Delta 4.38167577272998"
## [1] "11 step"
## [1] "Delta 3.96588445350517"
## [1] "12 step"
## [1] "Delta 3.32556724036476"
## [1] "13 step"
## [1] "Delta 2.20050659596109"
## [1] "14 step"
## [1] "Delta 1.20893346216963"
## [1] "15 step"
## [1] "Delta 0.747464091652546"
## [1] "16 step"
## [1] "Delta 0.55125743899589"
## [1] "17 step"
## [1] "Delta 0.424112038592284"
## [1] "18 step"
## [1] "Delta 0.312109071442137"
## [1] "19 step"
## [1] "Delta 0.207615620919569"
## [1] "20 step"
## [1] "Delta 0.110191249027594"
## [1] "21 step"
## [1] "Delta 0.0178151075888309"
```




```
# extracting output
res_mat = output$res_mat
parameters = output$clustered_parameters
print(parameters)
```

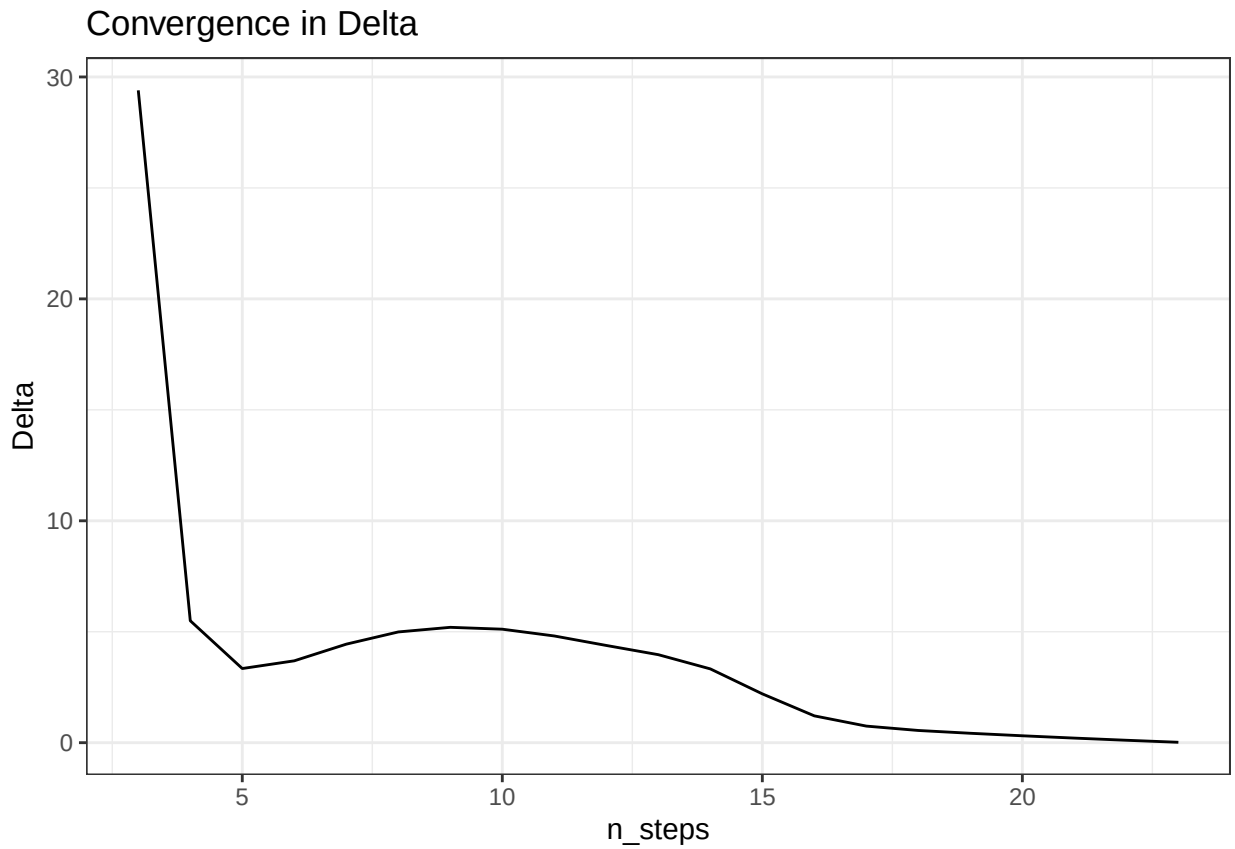
```
## $mean
##           [,1]       [,2]      [,3]       [,4]       [,5]
## [1,]  3.040082  0.8099551  4.458355  3.6742615  0.9273165
## [2,] -7.277069 -0.3920041  1.421441 -0.3529998 -0.9638594
## [3,] -5.819222 -6.7090722 -1.146800 -5.2837111 -1.0457731
##
## $cov
## , , 1
##
##           [,1]       [,2]      [,3]       [,4]       [,5]
## [1,] 10.904862  -6.932501  9.238915  -5.421862   9.478253
## [2,] -6.932501  24.053864 -9.170024  22.413220 -17.658362
## [3,]  9.238915  -9.170024 17.884081  -3.702339   9.457382
## [4,] -5.421862  22.413220 -3.702339  41.045632 -28.735298
## [5,]  9.478253 -17.658362  9.457382 -28.735298  33.703536
##
## , , 2
##
##            [,1]       [,2]      [,3]       [,4]      [,5]
## [1,]  46.553850 -41.115873 -2.197534 -40.372249  5.965131
## [2,] -41.115873  44.535913  1.156576  36.032451 -7.517609
```

```
## [3,]  -2.197534   1.156576 51.821320   1.144405 32.411584
## [4,] -40.372249  36.032451  1.144405  45.050026 -7.938302
## [5,]   5.965131  -7.517609 32.411584  -7.938302 29.825407
##
## , , 3
##
##            [,1]       [,2]      [,3]      [,4]      [,5]
## [1,]  11.911970 -10.209515  5.534434 -8.305063  4.046024
## [2,] -10.209515  50.875715  5.322472 40.318808  6.746388
## [3,]   5.534434   5.322472 38.575518 13.264600 35.161572
## [4,]  -8.305063  40.318808 13.264600 47.323537 15.894614
## [5,]   4.046024   6.746388 35.161572 15.894614 40.813552
##
##
## $wt
## [1] 19.00159 14.33881 11.25960
```

Below is the code to show the convergence in delta (difference between log-likelihood values)

We have used the difference in the log-likelihood function in consecutive iterations for convergence. We stop when the improvement in log-likelihood is less than some threshold value.

```
# plotting Delta to show convergence in delta
t <- output$delta_list
df_temp <- data.frame(col_name = unlist(t[3:length(t)]))
df_temp %>% ggplot(aes(c(3:length(t)),col_name)) + geom_line() +
  xlab("n_steps") + ylab("Delta")  +theme_bw() + ggtitle("Convergence in Delta")
```



Reference : https://stephens999.github.io/fiveMinuteStats/intro_to_em.html