Statistical learning project

Minh Nhat Do - Yasmin Mosbah

10/27/2021

Expectation-Maximization (EM) for multivariate GMMs on the Wine dataset

The goal is to apply EM for multivariate GMMs on the Wine dataset, available in the `pgmm` package.

Instructions

1) Implementing the EM

- Implement from scratch the EM algorithm for a GMM on the variables 2 and 4 of the wine data set.
- Cluster the data and compare your results with k-means.
- To assess the quality of the clustering, you may use the function classError and/or adjustedRandIndex from the Mclust package.

2) Model selection

- Try to find a relevant number of clusters using the three methods seen in class: AIC, BIC, and (cross-)validated likelihood.

3) Towards higher dimensional spaces

- Try to model more than just two variables of the same data set. Do you find the same clusters, the same number of clusters.

---

1 - Setup library and dataset summary

1.1. Setup library

In this project, we will use pgmm, ggplot2, expm, mclust libraries to implement.

1.2. Wine dataset

From the pgmm package are loaded, we can load the **wine** data in the environment. In the first 2 parts, we just work on the variables 2 and 4 of the wine data set (Alcohol and Fixed Acidity). We also load the Type of wine for evaluating the cluster result.

```
data(wine)
head(wine)
```

```
##   Type Alcohol Sugar-free Extract Fixed Acidity Tartaric Acid Malic Acid
## 1    1   14.23              24.82          73.1          1.21       1.71
## 2    1   13.20              26.30          72.8          1.84       1.78
## 3    1   13.16              26.30          68.5          1.94       2.36
## 4    1   14.37              25.85          74.9          1.59       1.95
## 5    1   13.24              26.05          83.5          1.30       2.59
```

```
## 6    1   14.20              28.40          79.9          2.14       1.76
##   Uronic Acids   pH  Ash Alcalinity of Ash Potassium Calcium Magnesium
## 1         0.72 3.38 2.43              15.6       950      62       127
## 2         0.71 3.30 2.14              11.2       765      75       100
## 3         0.84 3.48 2.67              18.6       936      70       101
## 4         0.72 3.43 2.50              16.8       985      47       113
## 5         1.10 3.42 2.87              21.0      1088      70       118
## 6         0.96 3.39 2.45              15.2       868      71       112
##   Phosphate Chloride Total Phenols Flavanoids Non-flavanoid Phenols
## 1       320       82          2.80       3.06                  0.28
## 2       395       90          2.65       2.76                  0.26
## 3       497       67          2.80       3.24                  0.30
## 4       580       49          3.85       3.49                  0.24
## 5       408       65          2.80       2.69                  0.39
## 6       418       58          3.27       3.39                  0.34
##   Proanthocyanins Color Intensity  Hue OD280/OD315 of Diluted Wines
## 1            2.29            5.64 1.04                         3.92
## 2            1.28            4.38 1.05                         3.40
## 3            2.81            5.68 1.03                         3.17
## 4            2.18            7.80 0.86                         3.45
## 5            1.82            4.32 1.04                         2.93
## 6            1.97            6.75 1.05                         2.85
##   OD280/OD315 of Flavanoids Glycerol 2-3-Butanediol Total Nitrogen Proline
## 1                      4.77     9.29            757            153    1065
## 2                      3.80     8.93            881            194    1050
## 3                      3.46    11.74            900            206    1185
## 4                      3.54    10.13           1119            292    1480
## 5                      3.22    10.27            799            215     735
## 6                      3.16    10.85            865            364    1450
##   Methanol
## 1      113
## 2       94
## 3      125
## 4       80
## 5       73
## 6       68
```

```
summary(wine[,c(1,2,4)])
```

```
##       Type          Alcohol      Fixed Acidity
##  Min.   :1.000   Min.   :11.03   Min.   : 50.50
##  1st Qu.:1.000   1st Qu.:12.36   1st Qu.: 70.95
##  Median :2.000   Median :13.05   Median : 80.00
##  Mean   :1.938   Mean   :13.00   Mean   : 85.64
##  3rd Qu.:3.000   3rd Qu.:13.68   3rd Qu.: 99.67
##  Max.   :3.000   Max.   :14.83   Max.   :137.80
```

```
X = as.matrix(wine[,c(2,4)])
y = wine[,1]
print("Feature dimensions: ")
```

```
## [1] "Feature dimensions: "
```

```
print(paste0("Number of rows: " , nrow(X)))
```

```
## [1] "Number of rows: 178"
```

```r
print(paste0("Number of columns: " , ncol(X)))
```

```
## [1] "Number of columns: 2"
```

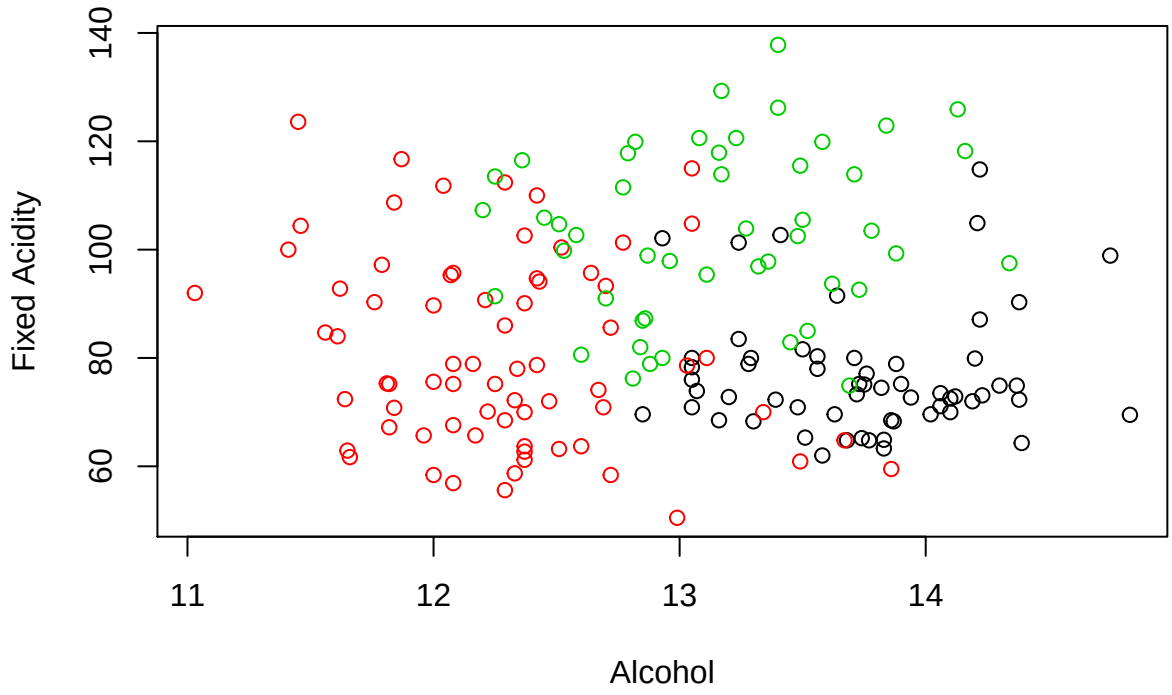
```r
print("Type valuesof wine: , ")
```

```
## [1] "Type valuesof wine: , "
```

```r
print(unique(y))
```

```
## [1] 1 2 3
```

```r
plot(X,col=y)
```



2. Implement Expectation-Maximization (EM) Algorithm

2.1. Create usefull functions for implementing

```r
# The logarithm of the sum of the exponentials of the arguments
logsumexp <- function (x) {
  y = max(x)
  y + log(sum(exp(x - y)))
}

#Normelize the arguments
normalise <- function (x) {
  logratio = log(x) - logsumexp(log(x))
  exp(logratio)
```

```
}

# Generator of randoms centroids depended on our data range (Min and Max of each colums in data) for mu
creator_centroides <- function(n_centroides = 3,datos = datos, seed = 99){#100
  set.seed(seed)

  x = matrix(ncol = ncol(datos), nrow = n_centroides)
  for (i in 1:ncol(datos)) {
    x[,i] = runif(n_centroides,min(datos[,i]), max(datos[,i]))
  }
  x = data.frame(x, stringsAsFactors = FALSE)
  return(as.matrix(x))
}
```

2.2. Create function to get computations on Log-likelihood

```
metrics_computation <- function(X, K, prob, mu, sigma) {
  n = nrow(X)
  d = ncol(X)
  log_gamma_numerator = matrix(unlist(lapply(1:K, FUN = function(k) {log(prob[k]) + dmvnorm(X, mu[k,], s

  log_likelihood = sum(apply(log_gamma_numerator , 1, logsumexp))
  return (list(logLik=log_likelihood))
}
```

2.3. EM Algorithm implement

The EM algorithm is an iterative approach that cycles between two modes. We start by initializing random cluster centers and then iteratively refine the clusters based the expectation step and the maximization step.

The goal is now to estimate our parameters

$$\theta = (\pi_1, ..., \pi_K, \mu_1, ...\mu_K, \Sigma_1, ...\Sigma_K)$$

The density of this mixture distribution can be expressed as:

$$p_\theta(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

The complete data log likelihood can be expressed as:

$$\ell(\theta) = \sum_{n=1}^{N} log(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k))$$

The EM algorithm consists of 3 major steps:

1. Initialization

We start by initializing random cluster centers

2. Expectation (E-step)

The Expectation step is to estimate the distribution of hidden variable given the data with the current value of the parameters. The responsablilities or posterior probabilities which will be denoted by $\gamma(z_{nk})$

$$\gamma(z_{nk}) = p(k|x_n) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

for all n, k, compute:

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

3. Maximization (M-step)

The Maximization step is to maximize and optimize the parameters. The process is repeated until a good set of parameters and a maximum likelihood is achieved that fits the data.

$$\pi_k^{'} = \frac{\sum_{i=1}^{N} Pr(u_i = k \,|x_i \,; \mu, \sigma, \pi)}{N} = \frac{N_k}{N}$$

$$\mu_k^{'} = \frac{1}{N_k} \sum_{j=1}^{n} \gamma(z_{nk}) x_n$$

$$\Sigma_k^{'} = \frac{1}{N_k} \sum_{j=1}^{n} \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T$$

After each iteration of the EM algorithm (one E step followed by one M step), we can monitor convergence by looking at the evolution of the value of the log-likelihood.

EMOrigin function is to implement EM algorithm manually.

The Input parameters:

- X: Data
- K: Number of clusters
- tol: value minimum to early stop
- max_it: The maximum number of iteration

The Out parameters:

- prob: A vector of probabilities of clusters (π_k)
- mu: The list of final mu values after training (μ_k)
- sigma: The list of final sigma values after training (Σ_k)
- gamma: The list of final gamma values after training ($\gamma(z_{nk})$)

```
EMOrigin <- function(X, K, max_it=50, tol=1e-8, showits=T){
  # number rows of X
  n = nrow(X)
  # number rows of X
  p = ncol(X)

  # initialization of \theta^ - mu, sigma, gamma
  prob = rep(1/K, K)
  # Use creator_centroides to create random value depended on our data range for mu
  mu = creator_centroides(K, X)
  sigma = lapply(1:K, function(i) cov(X))
  gamma = matrix(NA, n, K)
```

```
  log_likelihood = 0
  for  (i in 1:max_it) {
    #Store old value of log_lik and prob to compare after each iters
    log_likelihood_old = log_likelihood
    prob_old = prob
    # E step
    log_gamma_numerator = matrix(nrow=n, ncol = K)
    for (k in 1:K){
      log_gamma_numerator[,k] = log(prob[k]) + dmvnorm(X, mu[k,], sigma[[k]], log = TRUE)
    }
    ## normalize each line of gamma
    log_gamma = log_gamma_numerator - apply(log_gamma_numerator , 1, logsumexp)
    gamma = exp(log_gamma)
    # M step
    for (k in 1:K){
      nk = sum(gamma[,k])
      prob[k] = nk / n
      mu[k,] = colSums(gamma[,k]*X) / nk
      sigma[[k]] = ( t(sweep(X, 2, mu[k,])) %*% diag(gamma[,k]) %*% (sweep(X, 2, mu[k,])) ) / nk
    }

    # Evaluate the log-likelihood
    log_likelihood = sum(apply(log_gamma_numerator , 1, logsumexp))
    ### compare old to current for convergence
    parmlistold =  c(log_likelihood_old, prob_old)           # c(log_likelihood_old, prob_old)
    parmlistcurrent = c(log_likelihood, prob)              # c(log_likelihood, prob)
    if (showits & (i == 1 | i%%5 == 0))          # if showits true, & it =1 or modulo of 5 print message
      cat(paste("Iterations ", format(i),  " of EM: ", " ------ loglik : ", log_likelihood,"\n", sep =
    if(min(abs(parmlistold - parmlistcurrent)) <= tol){
      break
    }
  }
  # compute the sum log-likelihood
  metrics = metrics_computation(X, K, prob, mu, sigma)
  logLik <- metrics$logLik

  return (list(prob=prob, mu=mu, sigma=sigma, gamma=gamma, logLik=logLik))
}
```

After that, we will create a function to get the cluster result from parameters which we got from learning data. In this function:

- The input: gamma is gamma parameter from learning result.
- The output: List of cluster result corresponds to each data appearing in X (format: $[c_1, c_2, c_3, ..., c_N]$ with $c_i \in [0, K]$, K is number of cluster, N: number rows of data X).

```
# Input param is gamma from EM model result
EMOrigin.findCluster <- function(gamma){
  clust = rep(NA, nrow(gamma))
  # For each rows, we find order of value gamma where have the max gamma value
  for (i in 1:nrow(gamma)){
    clust[i] = which.max(gamma[i,])
  }
  return (clust)
```

```
}
```

2.4. K-mean Algorithm implement

The usual way to implement K-means Algorithm is the following:

- Step 1: Initialize randomly the centers by selecting k-observations
- Step 2: While some convergence criterion is not met
  - Step 2.1: Assign each observation to its closest center
  - Step 2.2: Update each center. The new centers are the mean of the observation of each group.
  - Step 2.3: Update the convergence criterion.

The input of function: - data: Input data - K: Number of clusters - stop_crit: Stop point to stop loop

```
kmeans=function(data,K=4,stop_crit=10e-5)
{
  #Initialization step (Initialization of clusters)
  n = nrow(data)
  p = ncol(data)
  centroids=data[sample.int(n,K),]
  current_stop_crit=1000
  cluster=rep(0,n)
  converged=F
  it=1
  while(current_stop_crit>=stop_crit & converged==F)
  {
    it=it+1
    if (current_stop_crit<=stop_crit)
    {
      converged=T
    }
    old_centroids=centroids
    # Assigning each point to a centroid
    for (i in 1:n)
    {
      min_dist=10e10
      for (centroid in 1:nrow(centroids))
      {
        distance_to_centroid=sum((centroids[centroid,]-data[i,])^2)
        if (distance_to_centroid<=min_dist)
        {
          cluster[i]=centroid
          min_dist=distance_to_centroid
        }
      }
    }
    ##Assigning each point to a centroid
    for (i in 1:nrow(centroids))
    {
      centroids[i,]=apply(data[cluster==i,],2,mean)
    }
    current_stop_crit=mean((old_centroids-centroids)^2)
  }
  return(list(size=tabulate(cluster),cluster=cluster,centers=centroids))
}
```

3. Comparing EM and K-Means results

We compares two different methods of the unsupervised classification. EM Algorithm have similarity with Kmeans, because they use the same optimization strategy on the M Step algorithm.

K-Means will first generate random cluster centers, then the data in the dataset will be distributed in each of these clusters according to which centroid they are closest. And at each iteration, the cluster center is recalculated and redefined according to the average of the data assigned in the cluster.

In EM case (with mixture of Gaussians) , each Gaussian has an associated mean and covariance matrix. The parameters are initialized by randomly selecting means (like k means), then the algorithm converges on a locally optimal solution by iteratively updating values for means and variance.

K-means finds the parameters of the centroid to minimize X minus the mean squared. The EM model finds the centroid to minimize X minus mean squared over the standard deviation squared. Mathematically the difference is the denominator σ^2, which means EM Algorithm takes variance into consideration when it calculates the measurement.

That's why K-Means will assign each data points to exactly one cluster but EM Algorithm will assign data points to cluster with some probability.

3.1 EM algorithm result

```
# Learning data with 3 cluster and 1000 iterations.
params = EMOrigin(X, 3, 1000, showits = T)
```

```
## Iterations 1 of EM:  ------ loglik : -1357.95370719882
## Iterations 5 of EM:  ------ loglik : -983.451108326672
## Iterations 10 of EM:  ------ loglik : -982.104834829053
## Iterations 15 of EM:  ------ loglik : -980.539170160433
## Iterations 20 of EM:  ------ loglik : -973.406554793745
## Iterations 25 of EM:  ------ loglik : -967.88518743623
## Iterations 30 of EM:  ------ loglik : -963.578358707479
## Iterations 35 of EM:  ------ loglik : -959.528872628531
## Iterations 40 of EM:  ------ loglik : -953.970334356682
## Iterations 45 of EM:  ------ loglik : -953.394120851145
## Iterations 50 of EM:  ------ loglik : -953.326363758286
## Iterations 55 of EM:  ------ loglik : -953.279080983415
## Iterations 60 of EM:  ------ loglik : -953.234309649315
## Iterations 65 of EM:  ------ loglik : -953.190536615061
## Iterations 70 of EM:  ------ loglik : -953.14705493542
## Iterations 75 of EM:  ------ loglik : -953.102763283718
## Iterations 80 of EM:  ------ loglik : -953.0556334081
## Iterations 85 of EM:  ------ loglik : -953.002068876818
## Iterations 90 of EM:  ------ loglik : -952.935679327785
## Iterations 95 of EM:  ------ loglik : -952.844700489412
## Iterations 100 of EM:  ------ loglik : -952.706866194139
## Iterations 105 of EM:  ------ loglik : -952.480753539611
## Iterations 110 of EM:  ------ loglik : -952.096245468153
## Iterations 115 of EM:  ------ loglik : -951.457158852884
## Iterations 120 of EM:  ------ loglik : -950.54316241711
## Iterations 125 of EM:  ------ loglik : -949.921721593273
## Iterations 130 of EM:  ------ loglik : -949.80680668176
## Iterations 135 of EM:  ------ loglik : -949.795784952623
## Iterations 140 of EM:  ------ loglik : -949.79480896783
## Iterations 145 of EM:  ------ loglik : -949.794713172983
## Iterations 150 of EM:  ------ loglik : -949.794701148197
```

```
## Iterations 155 of EM:  ------ loglik : -949.79469908999
## Iterations 160 of EM:  ------ loglik : -949.794698649555
## Iterations 165 of EM:  ------ loglik : -949.794698544952
```

When the loglikelihood does not evolve anymore, the code decides to stop because we have reached the maximum loglikelihood. We see that at the beginning of the iterations the loglikelihood evolves enormously, then we notice that loglikelihood converges progressively to the value -949.79. The EM algorithm is used to obtain maximum loglikelihood estimates of the parameters and here the result is -949.79.

This is the final result after training data with GMM model with 3 clusters and variable 2 and variable 4 of wine dataset

```
params
```

```
## $prob
## [1] 0.4841907 0.1768994 0.3389099
##
## $mu
##            X1        X2
## [1,] 12.86082 101.14812
## [2,] 12.19020  67.62040
## [3,] 13.62335  72.89922
##
## $sigma
## $sigma[[1]]
##                  Alcohol Fixed Acidity
## Alcohol       0.6184142      2.061868
## Fixed Acidity 2.0618680    198.619103
##
## $sigma[[2]]
##                   Alcohol Fixed Acidity
## Alcohol         0.1123079    -0.9141354
## Fixed Acidity -0.9141354    62.0045671
##
## $sigma[[3]]
##                   Alcohol Fixed Acidity
## Alcohol         0.2330766    -0.8467555
## Fixed Acidity -0.8467555    36.6723276
##
##
## $gamma
##                [,1]         [,2]         [,3]
##   [1,] 0.011942403 1.446830e-10 9.880576e-01
##   [2,] 0.050279549 1.761692e-03 9.479588e-01
##   [3,] 0.046212049 1.146682e-02 9.423211e-01
##   [4,] 0.020465795 5.573526e-12 9.795342e-01
##   [5,] 0.383695367 5.503370e-05 6.162496e-01
##   [6,] 0.096961960 3.142030e-11 9.030380e-01
##   [7,] 0.003279291 2.150637e-10 9.967207e-01
##   [8,] 0.013440114 2.768043e-09 9.865599e-01
##   [9,] 0.008105027 3.045352e-15 9.918950e-01
##  [10,] 0.007552900 4.746113e-07 9.924466e-01
##  [11,] 0.010648394 1.892897e-09 9.893516e-01
##  [12,] 0.011526450 1.156736e-09 9.884735e-01
##  [13,] 0.024211670 3.139487e-07 9.757880e-01
##  [14,] 0.999994569 1.158311e-22 5.431482e-06
```

```
##  [15,] 0.010251012 1.103559e-11 9.897490e-01
##  [16,] 0.012678768 1.201459e-05 9.873092e-01
##  [17,] 0.019608826 2.144709e-11 9.803912e-01
##  [18,] 0.007018072 3.153064e-06 9.929788e-01
##  [19,] 0.009359485 4.405929e-10 9.906405e-01
##  [20,] 0.964410419 2.772163e-09 3.558958e-02
##  [21,] 0.008345764 6.221613e-09 9.916542e-01
##  [22,] 0.999975521 4.188292e-08 2.443731e-05
##  [23,] 0.095355541 1.534671e-07 9.046443e-01
##  [24,] 0.115583155 2.171881e-01 6.672288e-01
##  [25,] 0.180695550 2.542905e-06 8.193019e-01
##  [26,] 0.167649299 2.478986e-03 8.298717e-01
##  [27,] 0.028629155 1.617053e-04 9.712091e-01
##  [28,] 0.029037883 2.056931e-03 9.689052e-01
##  [29,] 0.007315655 4.339569e-07 9.926839e-01
##  [30,] 0.006897535 2.138326e-08 9.931024e-01
##  [31,] 0.025316331 4.196562e-07 9.746832e-01
##  [32,] 0.016331874 4.918962e-04 9.831762e-01
##  [33,] 0.010023069 3.436912e-05 9.899426e-01
##  [34,] 0.039364187 1.504417e-07 9.606357e-01
##  [35,] 0.015854484 3.470397e-04 9.837985e-01
##  [36,] 0.019916614 7.084537e-05 9.800125e-01
##  [37,] 0.117843431 1.148321e-04 8.820417e-01
##  [38,] 0.070075309 1.969083e-02 9.102339e-01
##  [39,] 0.082503108 6.408299e-03 9.110886e-01
##  [40,] 1.000000000 6.820471e-24 1.968628e-12
##  [41,] 0.116661820 1.463560e-06 8.833367e-01
##  [42,] 0.999988694 2.362542e-11 1.130647e-05
##  [43,] 0.062959277 1.258177e-08 9.370407e-01
##  [44,] 0.999956805 1.075613e-09 4.319385e-05
##  [45,] 0.233269172 1.584526e-03 7.651463e-01
##  [46,] 0.999999832 2.973966e-19 1.676467e-07
##  [47,] 0.976850678 3.109541e-15 2.314932e-02
##  [48,] 0.021791830 2.581311e-08 9.782081e-01
##  [49,] 0.006817260 4.546559e-09 9.931827e-01
##  [50,] 0.012212638 2.877752e-08 9.877873e-01
##  [51,] 0.114215246 4.554786e-03 8.812300e-01
##  [52,] 0.007354734 6.168729e-06 9.926391e-01
##  [53,] 0.019724308 1.207740e-07 9.802756e-01
##  [54,] 0.008034836 8.528577e-06 9.919566e-01
##  [55,] 0.008562925 1.159724e-05 9.914255e-01
##  [56,] 0.061316025 2.679003e-06 9.386813e-01
##  [57,] 0.794995220 1.051617e-12 2.050048e-01
##  [58,] 0.151127831 7.548829e-05 8.487967e-01
##  [59,] 0.017462338 8.756854e-07 9.825368e-01
##  [60,] 0.976174422 1.087782e-02 1.294776e-02
##  [61,] 0.092324869 8.688433e-01 3.883184e-02
##  [62,] 0.998394668 7.030805e-05 1.535024e-03
##  [63,] 0.010287335 4.001498e-05 9.896726e-01
##  [64,] 0.063316168 9.061287e-01 3.055512e-02
##  [65,] 0.024477914 9.736973e-01 1.824812e-03
##  [66,] 0.018848152 9.782716e-01 2.880227e-03
##  [67,] 0.207585056 7.685330e-04 7.916464e-01
##  [68,] 0.021489970 9.743567e-01 4.153284e-03
```

```
##  [69,] 0.027052872 6.740866e-04 9.722730e-01
##  [70,] 0.975660159 1.867977e-02 5.660072e-03
##  [71,] 0.010882155 9.890350e-01 8.280127e-05
##  [72,] 0.009712592 2.113380e-05 9.902663e-01
##  [73,] 0.025495129 2.860007e-03 9.716449e-01
##  [74,] 0.031806331 9.643816e-01 3.812109e-03
##  [75,] 0.029082662 9.704551e-01 4.621963e-04
##  [76,] 0.062700104 9.372749e-01 2.500900e-05
##  [77,] 0.185526061 2.902395e-03 8.115715e-01
##  [78,] 0.999995460 4.514096e-06 2.611305e-08
##  [79,] 0.012424728 9.871432e-01 4.321217e-04
##  [80,] 0.993079943 1.694849e-04 6.750572e-03
##  [81,] 0.018044554 9.819148e-01 4.059791e-05
##  [82,] 0.024299544 9.645262e-01 1.117428e-02
##  [83,] 0.014789300 9.851755e-01 3.522068e-05
##  [84,] 0.999997457 1.077682e-09 2.541605e-06
##  [85,] 0.065118909 9.338848e-01 9.963244e-04
##  [86,] 0.215328771 2.935241e-01 4.911471e-01
##  [87,] 0.302259479 6.631520e-01 3.458852e-02
##  [88,] 0.065936087 9.340275e-01 3.643030e-05
##  [89,] 0.127918063 8.715731e-01 5.088546e-04
##  [90,] 0.032156522 9.660527e-01 1.790741e-03
##  [91,] 0.276808046 7.028529e-01 2.033909e-02
##  [92,] 0.132013350 8.604496e-01 7.537007e-03
##  [93,] 0.141694664 4.648511e-01 3.934542e-01
##  [94,] 0.867337022 9.380544e-02 3.885753e-02
##  [95,] 0.971848306 2.798782e-02 1.638711e-04
##  [96,] 0.119486890 7.715003e-01 1.090128e-01
##  [97,] 0.137576695 8.601843e-01 2.238993e-03
##  [98,] 0.041353523 9.479796e-01 1.066686e-02
##  [99,] 0.015840588 9.825341e-01 1.625329e-03
## [100,] 0.999999981 1.678484e-08 2.118362e-09
## [101,] 0.125766462 8.623155e-01 1.191807e-02
## [102,] 0.035611057 9.347549e-01 2.963406e-02
## [103,] 0.325609136 5.680645e-01 1.063264e-01
## [104,] 0.045015244 9.546528e-01 3.319220e-04
## [105,] 0.025904057 9.629312e-01 1.116473e-02
## [106,] 0.409273949 4.122713e-01 1.784548e-01
## [107,] 0.154096024 8.063149e-01 3.958912e-02
## [108,] 0.805702893 7.183729e-03 1.871134e-01
## [109,] 0.050944229 9.391669e-01 9.888910e-03
## [110,] 0.636862276 3.618491e-01 1.288581e-03
## [111,] 0.999858249 1.415843e-04 1.668890e-07
## [112,] 0.999929714 1.075090e-05 5.953554e-05
## [113,] 0.930448994 6.871463e-02 8.363742e-04
## [114,] 0.998776945 1.221163e-03 1.892377e-06
## [115,] 0.996753826 2.909609e-03 3.365651e-04
## [116,] 0.992058357 7.936048e-03 5.594888e-06
## [117,] 0.133622789 8.640489e-01 2.328280e-03
## [118,] 0.997617504 7.308344e-04 1.651661e-03
## [119,] 0.999957587 4.911822e-07 4.192141e-05
## [120,] 0.937223047 5.941187e-02 3.365081e-03
## [121,] 0.999999999 5.616997e-10 5.317006e-15
## [122,] 0.700292886 2.988066e-01 9.005477e-04
```

```
## [123,] 0.999999946 3.164727e-08 2.245617e-08
## [124,] 1.000000000 1.657472e-13 9.741823e-11
## [125,] 0.999999986 1.445252e-08 1.830412e-11
## [126,] 0.995882803 3.717601e-03 3.995958e-04
## [127,] 0.996710273 9.490912e-04 2.340636e-03
## [128,] 0.996480917 3.469690e-03 4.939256e-05
## [129,] 0.999981475 8.984987e-06 9.540402e-06
## [130,] 0.999999816 1.812423e-07 2.734252e-09
## [131,] 0.852444525 9.394183e-04 1.466161e-01
## [132,] 0.274331922 1.420289e-02 7.114652e-01
## [133,] 0.220670264 5.814876e-02 7.211810e-01
## [134,] 0.977543890 6.185594e-04 2.183755e-02
## [135,] 0.999997014 6.455395e-07 2.340712e-06
## [136,] 0.539875503 1.039314e-01 3.561931e-01
## [137,] 0.983966244 1.114335e-02 4.890407e-03
## [138,] 0.999893902 1.449901e-05 9.159881e-05
## [139,] 1.000000000 4.445843e-17 1.831647e-11
## [140,] 0.482485021 8.829756e-03 5.086852e-01
## [141,] 0.296747988 6.194108e-03 6.970579e-01
## [142,] 0.999396581 2.530261e-09 6.034169e-04
## [143,] 0.459601310 6.568251e-07 5.403980e-01
## [144,] 0.991023601 7.999655e-10 8.976398e-03
## [145,] 0.999999989 1.020747e-08 7.024335e-10
## [146,] 1.000000000 1.707256e-15 2.627703e-12
## [147,] 0.999912747 7.345172e-14 8.725331e-05
## [148,] 0.999739667 8.416288e-07 2.594918e-04
## [149,] 0.998820634 9.077713e-09 1.179357e-03
## [150,] 1.000000000 4.527346e-16 1.145880e-13
## [151,] 0.999999227 4.582389e-13 7.729193e-07
## [152,] 1.000000000 5.883514e-13 6.026086e-12
## [153,] 0.996788169 4.898823e-07 3.211341e-03
## [154,] 1.000000000 3.274302e-17 7.517423e-14
## [155,] 0.999986567 1.329581e-06 1.210303e-05
## [156,] 1.000000000 8.088749e-21 5.406180e-19
## [157,] 1.000000000 7.581311e-24 2.774738e-16
## [158,] 0.999998693 4.826322e-07 8.244717e-07
## [159,] 0.999894121 1.544033e-17 1.058789e-04
## [160,] 0.999987827 8.509685e-12 1.217297e-05
## [161,] 1.000000000 3.257345e-10 3.371629e-11
## [162,] 0.024881605 8.631882e-07 9.751175e-01
## [163,] 0.833499949 1.247008e-03 1.652530e-01
## [164,] 0.999444402 5.792527e-07 5.550189e-04
## [165,] 0.999997309 1.353277e-14 2.690818e-06
## [166,] 0.983302706 2.678195e-10 1.669729e-02
## [167,] 0.273564722 3.667673e-06 7.264316e-01
## [168,] 1.000000000 5.357271e-14 4.768523e-13
## [169,] 1.000000000 7.283615e-20 5.434349e-14
## [170,] 1.000000000 2.327647e-27 2.080344e-25
## [171,] 0.999998416 1.403262e-06 1.810529e-07
## [172,] 0.999999994 1.941343e-10 5.524736e-09
## [173,] 1.000000000 6.619842e-25 3.253925e-14
## [174,] 1.000000000 2.602026e-18 5.752134e-11
## [175,] 1.000000000 2.863131e-21 1.974376e-17
## [176,] 0.999995302 8.494963e-11 4.697696e-06
```

```
## [177,] 1.000000000 6.676323e-14 2.646774e-10
## [178,] 1.000000000 1.380106e-28 6.296067e-19
##
## $logLik
## [1] -949.7947
```

```
params$prob
```

```
## [1] 0.4841907 0.1768994 0.3389099
```

First of all we see that the dataset is not perfectly distributed. We have a cluster (cluster number 3) holding almost half of the data (~48%), while another one holds only ~18% (cluster number 2), and a 3rd cluster (cluster number 1) with 1/3 of the data.

```
params$mu
```

```
##            X1        X2
## [1,] 12.86082 101.14812
## [2,] 12.19020  67.62040
## [3,] 13.62335  72.89922
```

`mu` attribute of `params` shows us the position of the centroids of our 3 clusters. It is the average of the characteristics of each type of wine.

We notice that the cluster holding 48% (cluster number 3) of the wines is on average more acidic than the other two types of wines.

The other two types of wines differ in their alcohol concentration.

Cluster number 2 representing only 18% of the dataset has less alcohol than cluster number 1(34% of the dataset).

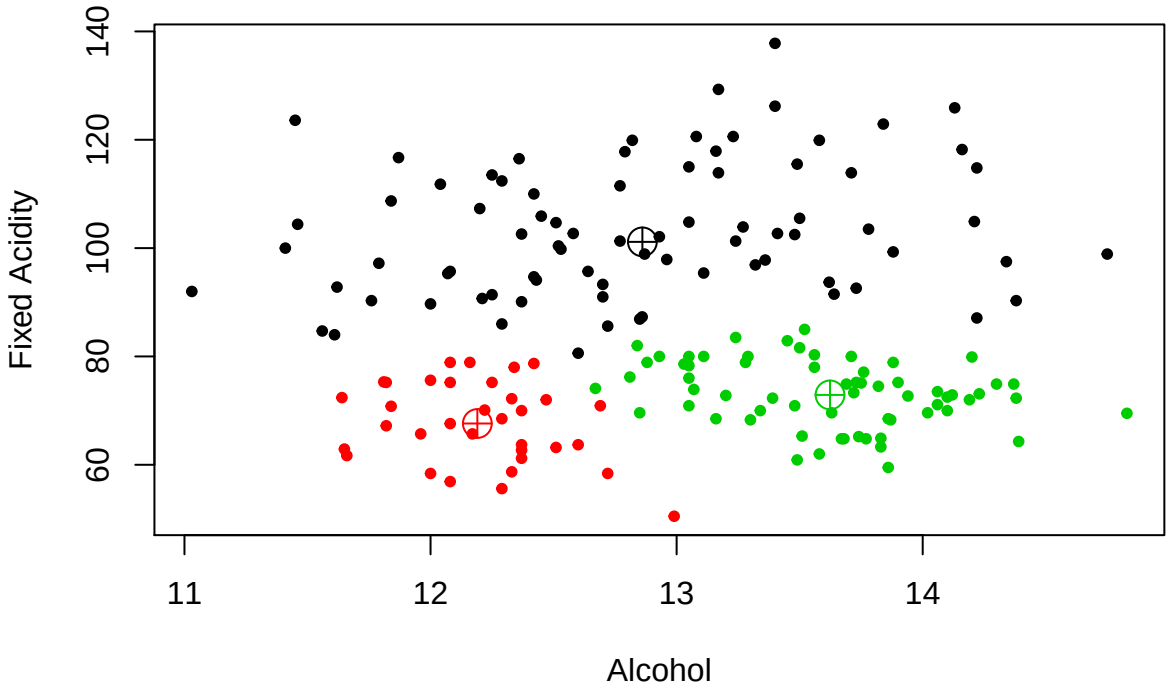
3.2. Visualization of the EM algorithm result

```
cluster_result = EMOrigin.findCluster(params$gamma)
cluster_result
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 1 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [38] 3 3 1 3 1 3 1 3 1 1 3 3 3 3 3 3 3 3 1 3 3 1 2 1 3 2 2 2 3 2 3 1 2 3 3 2
##  [75] 2 2 3 1 2 1 2 2 2 1 2 3 2 2 2 2 2 2 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 1 2 1 1
## [112] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 3 3 1 3 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1
```

Now, we plot the cluster results on X data.

```
plot(X, col=cluster_result, pch=20)
points(params$mu, col = 1:3, pch = 10, cex=2)
```



This graph confirms our observations. We can see that one cluster is different from the other two in terms of acidity. In the graph representing the 3 brands of wine seen earlier, we can already identify that each brand is different from the others according to these two characteristics (alcohol, acidity). EM Algorithm has more or less succeeded in identifying these 3 types of wine.

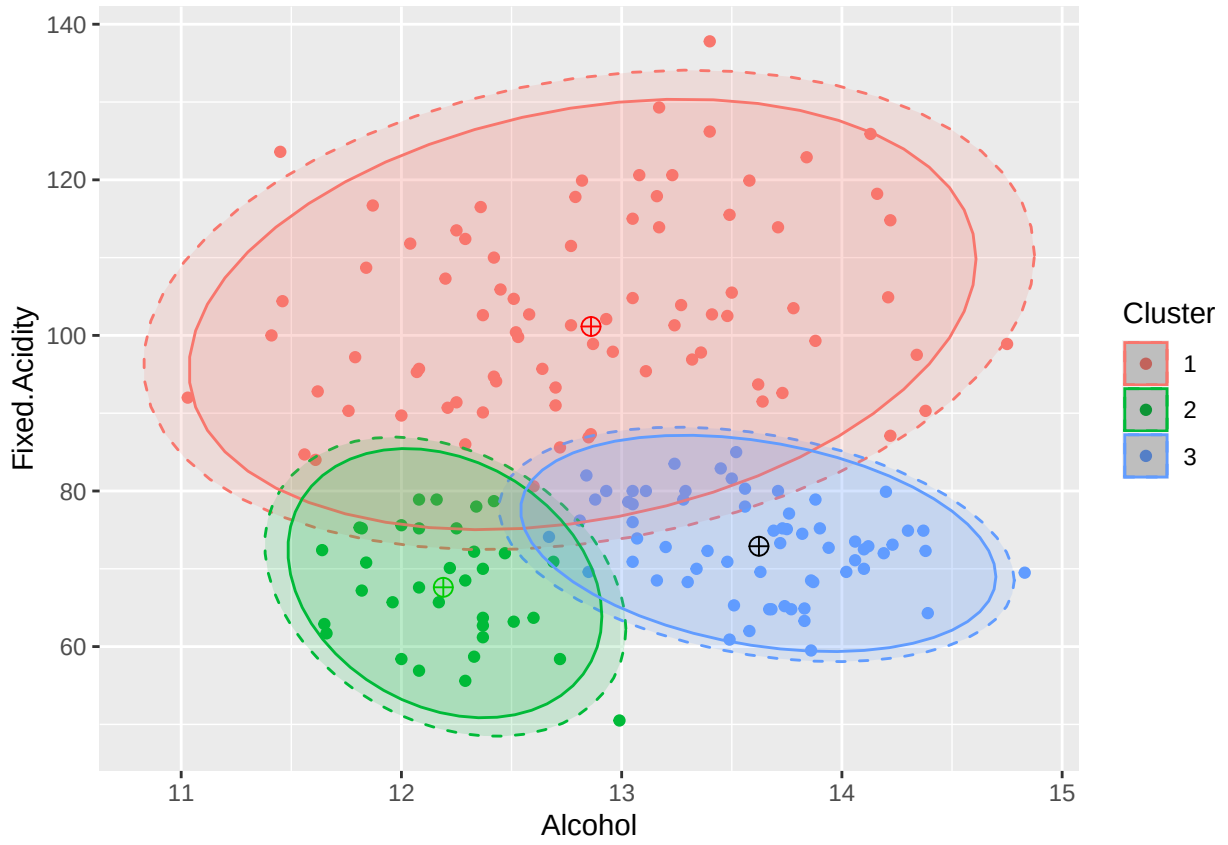
Plot cluster areas with multivariate t-distribution and a multivariate normal distribution

We use ggplot geom_point to create a scatter plot by group and use `stat_ellipse` to create the ellipses for each group. The `stat_ellipse` uses geom_path by default to create the ellipse, but if you set `geom = "polygon"` a polygon will be created. Note that you can change the level of transparency with `alpha`.

Based on the cluster results, we will plot 2 eclipses for each cluster corresponding to a multivariate t-distribution (solid line) and a multivariate normal distribution (dotted line).

```
plot_cluster_area <- function(X, centroids, cluster_result){
  x_df = data.frame(X)
  mu_clusters = data.frame(centroids)
  names(mu_clusters) = names(x_df)
  colors = as.factor(cluster_result)
  ggplot(x_df, aes_string(names(x_df)[1], names(x_df)[2], color=colors)) +
      geom_point() +
      stat_ellipse(geom="polygon", aes(fill=colors), alpha=0.15, type = "norm", linetype = 2) +
    labs(color="Cluster") +
    stat_ellipse(geom="polygon", aes(fill=colors), type = "t", alpha=0.15) +guides(fill = "none") + geo
}
```

```
plot_cluster_area(X, params$mu, cluster_result)
```



To compare the result with K-means, we will K-Means algorithm function which we developed above, to get the result.

3.3. K-Means algorithm result

Let's go to K-means algorithm in variable 2 and 4 of wine data set with 3 clusters:

```
cl <- kmeans(X, 3,stop_crit=1e-8)
cl
```

```
## $size
## [1] 54 27 97
##
## $cluster
##   [1] 3 3 3 3 1 3 3 3 3 3 3 3 3 1 3 3 3 3 3 1 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [38] 3 3 2 3 1 3 1 3 1 1 3 3 3 3 3 3 3 3 3 1 3 3 1 3 1 3 3 3 3 3 3 3 1 3 3 3 3
##  [75] 3 3 3 2 3 1 3 3 3 1 3 3 3 3 3 3 3 3 3 1 1 3 3 3 3 2 3 3 3 3 3 3 3 1 3 1 1
## [112] 1 1 1 1 1 3 1 1 1 2 1 2 2 2 1 1 1 1 2 1 3 3 1 1 3 1 1 2 3 3 1 1 1 2 2 1 1
## [149] 1 2 1 2 1 2 1 2 2 1 1 1 2 3 1 1 1 1 3 2 2 2 2 2 2 2 2 1 2 2
##
## $centers
##      Alcohol Fixed Acidity
## 106 12.82167      95.70000
## 128 12.95889     117.85556
## 102 13.11186      71.07835
```

```
## [1] "Distribution of data in the 3 clusters :"
```

```
## [1] "Kmeans :"
## [1] "0.303370786516854" "0.151685393258427" "0.544943820224719"
## [1] "GMM :"
## [1] "0.484190740443889" "0.176899372829121" "0.33890988672699"
```

We notice that the number of data per cluster have more or less the same distribution.

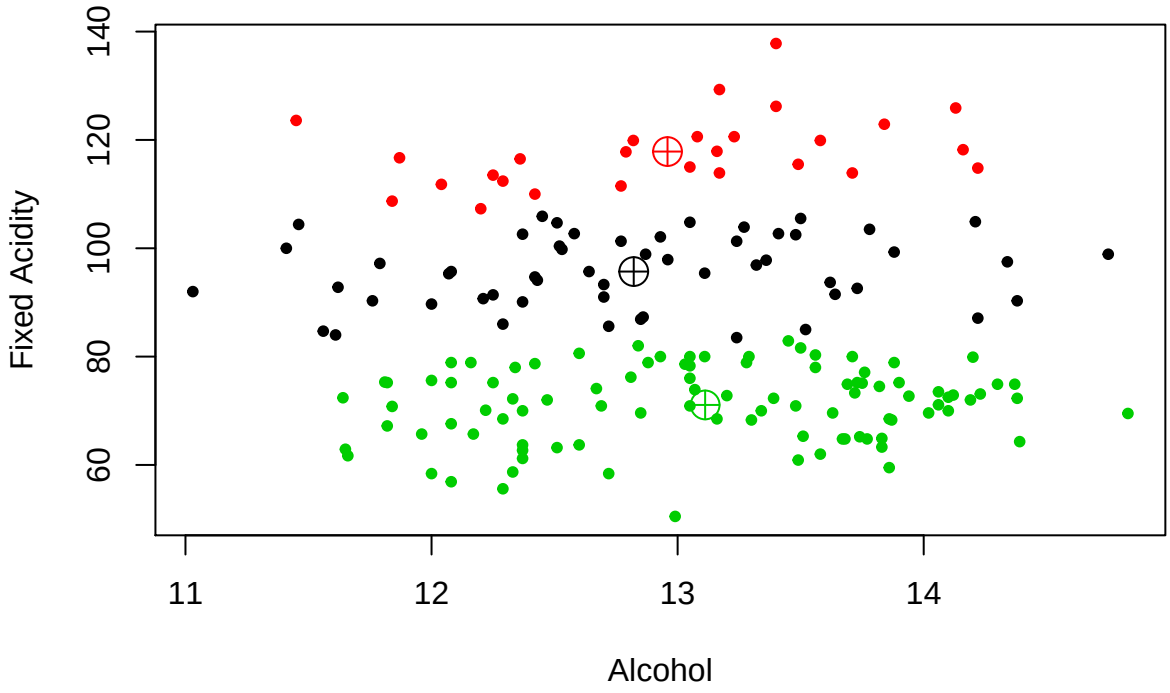
```
cl$centers
```

```
##      Alcohol Fixed Acidity
## 106 12.82167      95.70000
## 128 12.95889     117.85556
## 102 13.11186      71.07835
```

Using Kmeans, we notice that the 3 clusters are separated according to the acidity of the wine. All of centroids have more or less the same concentration of alcohol.

3.4. Visualization of K-Means algorithm result

```
plot(X, col = cl$cluster,  pch = 20)
points(cl$centers, col = 1:3, pch = 10, cex=2)
```

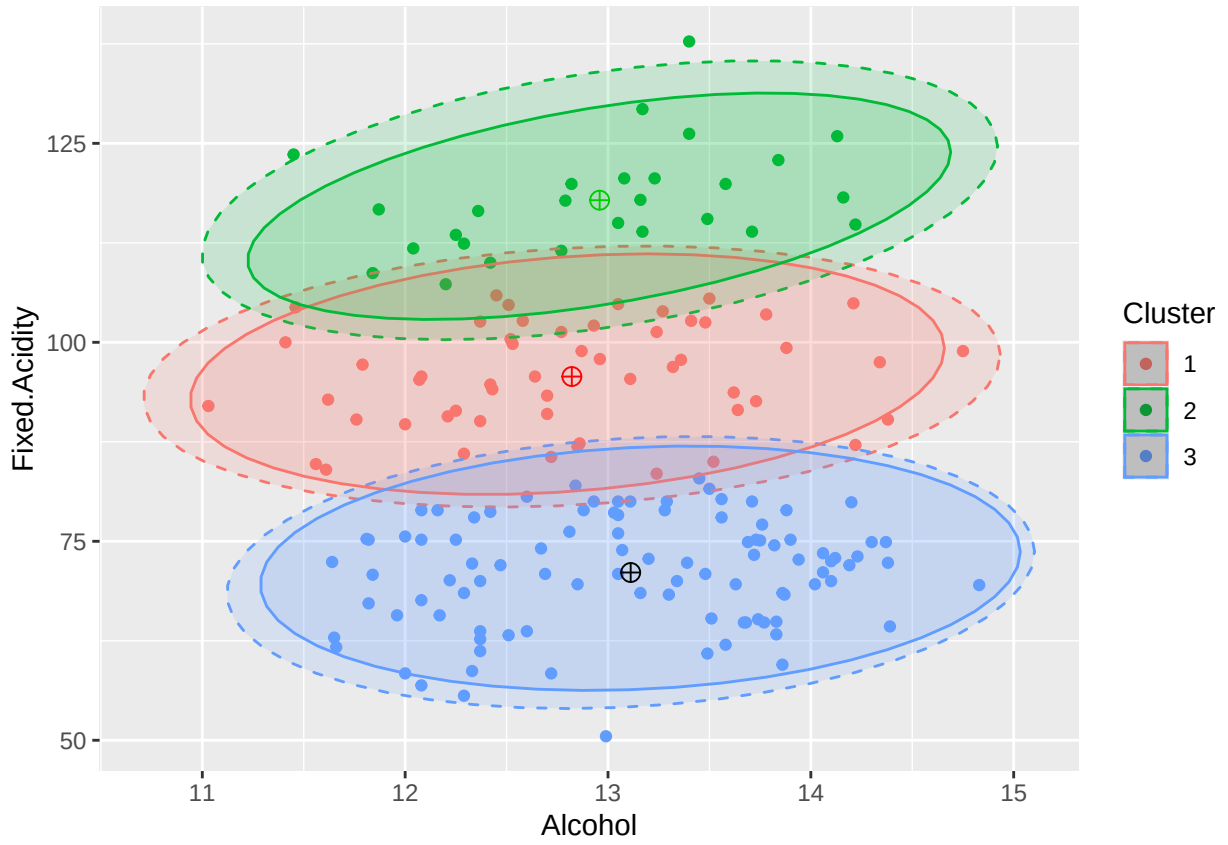


At first sight, EM algorithm seems to have much better results than K-means. We know the existence of the 3 types of wines and we can see that K-means does not detect the existence of these 3 categories.

K-means has made clusters that are visually linear. It separated each category according to the acidity of the wine and did not classify them according to the alcohol content.

we will plot 2 eclipses for each cluster corresponding to a multivariate t-distribution (solid line) and a multivariate normal distribution (dotted line).

```
plot_cluster_area(X, cl$centers, cl$cluster)
```



3.5. The quality of the clustering

First, We will split data X to training dataset and testing dataset

```
## 75% of the sample size
smp_ratio = 0.75
smp_size <- floor(smp_ratio * nrow(X))

## set the seed to make your partition reproducible
set.seed(123)
train_idx <- sample(seq_len(nrow(X)), size = smp_size)

x_train <- X[train_idx, ]
x_test <- X[-train_idx, ]
y_train <- y[train_idx]
y_test <- y[-train_idx]
```

Now, we can check the splitted result data and we check if the sub-dataset contains heterogeneously distributed data before training the models.

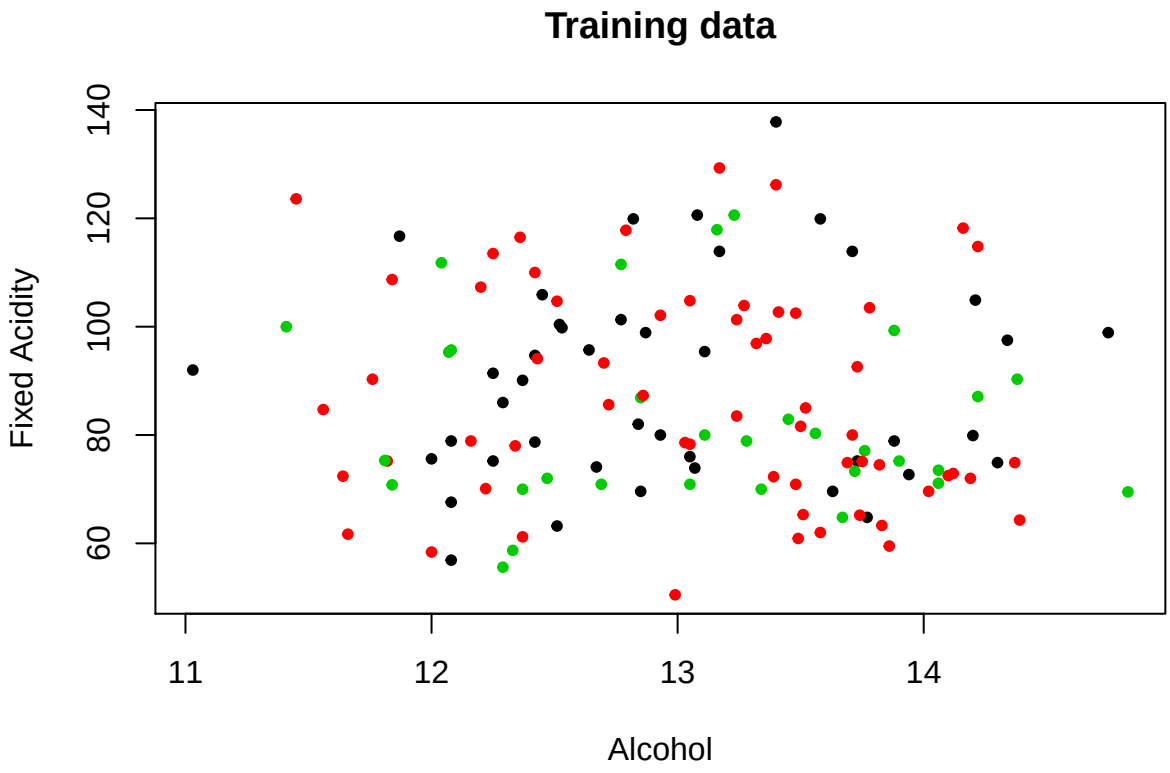
```
print("Feature dimensions: ")
```

```
## [1] "Feature dimensions: "
```

```r
print(paste0("Number rows of training dataset: " , nrow(x_train)))
```

```
## [1] "Number rows of training dataset: 133"
```

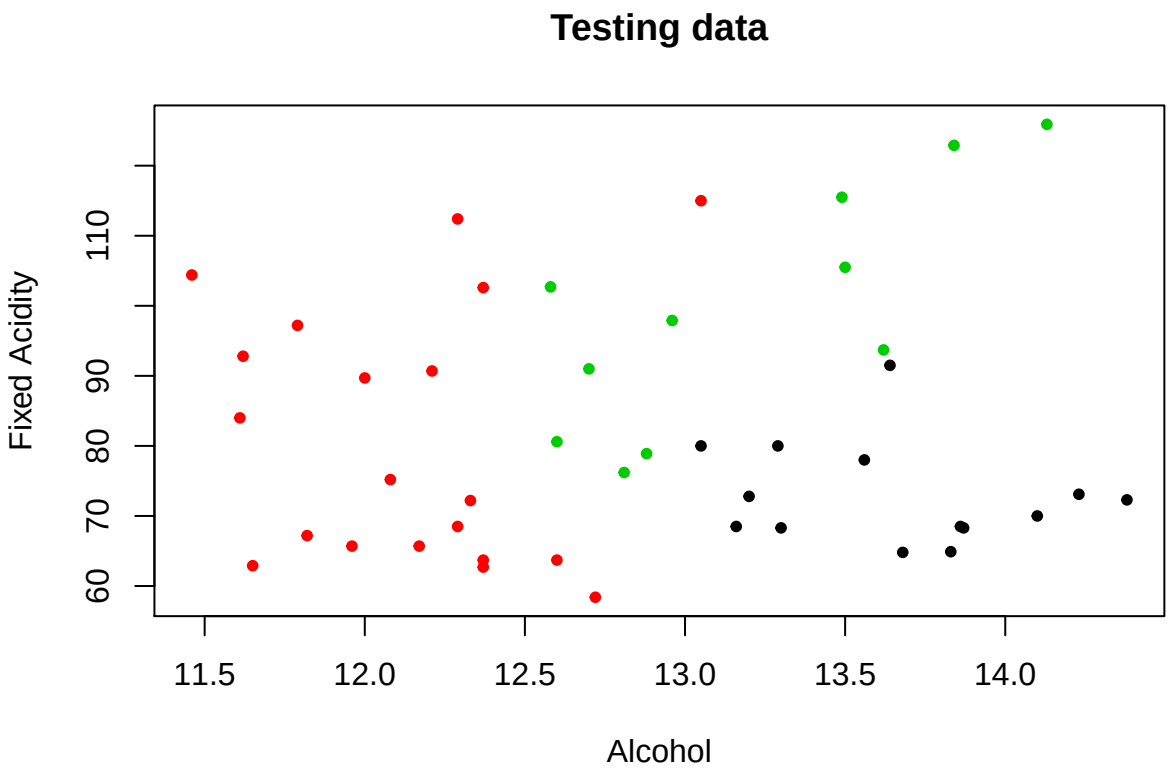
```r
plot(x_train,col=y_test, pch=20, main="Training data")
```




```r
print(paste0("Number rows of testing dataset: " , nrow(x_test)))
```

```
## [1] "Number rows of testing dataset: 45"
```

```r
plot(x_test,col=y_test, pch=20, main="Testing data")
```



The testing data has less input data, so it will be a challenge for the clustering model. Now we will use this datasets to train and test in EM model and Kmeans

- Training dataset:

```
number_of_cluster = 3
resultTrainEM = EMOrigin(x_train, number_of_cluster, 1000, showits = F)
# Find cluster on resultTrainEM
resultTrainEM$cluster = EMOrigin.findCluster(resultTrainEM$gamma)

resultTrainKmeans = kmeans(x_train, 3, stop_crit=1e-8)
```

- Testing dataset:

```
resultTestEM = EMOrigin(x_test, number_of_cluster, 1000, showits = F)
# Find cluster on resultTrainEM
resultTestEM$cluster = EMOrigin.findCluster(resultTestEM$gamma)

resultTestKmeans = kmeans(x_test, 3, stop_crit=1e-8)
```

3.5.1. Classification Error analysis

- Expectation-Maximization on training dataset:

```
classError(resultTrainEM$cluster, y_train)
```

```
## $misclassified
##  [1]   2   5  16  17  20  26  32  40  42  43  45  47  49  50  53  55  56  58  62
## [20]  67  69  72  75  78  89  91  93  95  96  98 102 105 107 108 109 110 113 118
```

```
## [39] 119 121 122 125 126 127 129 131 132 133
##
## $errorRate
## [1] 0.3609023
```

- K-Means on training dataset:

```
classError(resultTrainKmeans$cluster, y_train)
```

```
## $misclassified
##  [1]   1   2   6   9  10  13  14  15  16  17  20  21  24  25  26  28  31  32  37
## [20]  38  40  41  42  45  51  52  53  54  55  58  66  67  69  70  72  75  80  82
## [39]  83  84  87  88  89  93  95  96  99 101 105 107 109 110 111 113 118 122 123
## [58] 124 127 129 131 132
##
## $errorRate
## [1] 0.4661654
```

The error rate achieved by EM algorithm is 36% against the 47% of K-means on train dataset.

- Expectation-Maximization on testing dataset:

```
classError(resultTestEM$cluster, y_test)
```

```
## $misclassified
##  [1]  1  4  5  6  9 10 12 19 23 25 29 30 31 32 33 34 35 36 38 40
##
## $errorRate
## [1] 0.4444444
```

- K-Means on testing dataset:

```
classError(resultTestKmeans$cluster, y_test)
```

```
## $misclassified
##  [1]  7 15 16 17 18 20 21 22 24 25 26 27 28 32 35 36 37 38 40 41 42 44
##
## $errorRate
## [1] 0.4888889
```

Look at the results, we can see that Em Algorithm is still better than K means and this despite a smaller dataset.

3.5.2. Adjusted Rand Index analysis The Adjusted Rand score is to determine whether two cluster results are similar to each other. When it is equal to 0 the points are randomly assigned to the clusters and when it is equal to 1 the results of both clusters are identical.

- Expectation-Maximization on training datatset:

```
adjustedRandIndex(resultTrainEM$cluster, y_train)
```

```
## [1] 0.2193034
```

- K-Means on training datatset:

```
adjustedRandIndex(resultTrainKmeans$cluster, y_train)
```

```
## [1] 0.1469045
```

We know that the closer we get to a score of 0, the more likely the clustering is to be random, on the contrary if we get closer to a score of 1, then the clustering is almost perfect. Here we see that during the training,

EM Algorithm is better than K-Means

- Expectation-Maximization on testing datatset:

```
adjustedRandIndex(resultTestEM$cluster, y_test)
```

```
## [1] 0.1481718
```

- K-Means on testing datatset:

```
adjustedRandIndex(resultTestKmeans$cluster, y_test)
```

```
## [1] 0.09226625
```

We see that during the test, both models were worse than during the training, but EM Algorithm remains much better than KMean.

3.6. Observation and comments

The difference between the results of K-means and EM algorithm are truly visible. We don't have the same clusters at all. This result is different because K-means does not take into account the standard deviation of the data.

In our case we see that Em Algorithm produces much better results than K-Means. K-means has difficulty in clustering data when the clusters are of varying size and density. It is a clustering method that requires data with rather spherical clusters, which is not the case in the dataset we study. This may explain the significant difference in results between K-Means and EM Algorithm.

4. Model Selection

We have many way to select the best number of cluster such as BIC, AIC,.... So now, we will use 3 ways to select the K parameter:

- The **Akaike information criterion (AIC)** : An estimator of prediction error and thereby relative quality of statistical models for a given set of data
- the **Bayesian information criterion (BIC)** : A criterion for model selection among a finite set of models
- **Cross-validate K folds** on likelihood: A resampling method that uses different portions of the data to test and train a model on different iterations. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.

4.1. Model selection based on AIC score

To compute penalization in AIC criteria, we need to compute number of model's parameters.

In practice, number of model's parameters is easy to compute:

$$\eta(GMM \text{ with K}) = \text{number of } \pi_k + \text{number of } \mu_k + \text{number of } \Sigma_k = (K-1) + K \times d + K \times \frac{d(d+1)}{2}$$

AIC formula:

$$BIC = log(\mathcal{L}(x;\phi)) - \eta(GMM \text{ with K})$$

```r
AIC <- function(log_likelihood, K, data) {
  # K is number of clusters
  # D is number of columns of Data
  n = nrow(data)
  d = ncol(data)
  ######
  #### Implementation of the the Akaike Information Criterion such that:
  #### AIC = LogLikelihood - eta(M) with eta is number of parameters of model
  nb.param <- (K - 1) + K * d + K*d*(d+1)/2
  aic_score = log_likelihood - nb.param
  return(aic_score)
}

modelSelectionAIC <- function(data, listNumberCluster=2:10, print_steps=TRUE){
  ####
  #### Computes the AIC of an EM algorithm implementation given a dataset
  #### Input includes:
  #### data: Input data
  #### listNumberCluster: range of clusters for checking
  #### print_steps: Print process steps
  #### Output: result - result AIC scores, bestK - the best of number of cluster

  # Variable declaration
  aic_results = c()

  # Use for loop to compute the corresponding AIC on each cluster parameter and push to a list
  for (j in listNumberCluster){
    res = EMOrigin(data, j, showits=F)
    aic_value = AIC(res$logLik, j, data)
    aic_results = append(aic_results, aic_value)
    if (print_steps) {
      print(paste("Cluster ", j, "--- Log-likelihood = ",
                  round(aic_value, 3)))
    }
  }

  # Prints the result
  print(paste("The best K value is ", listNumberCluster[which.max(aic_results)] , " clusters (Based on
  return (list(result=aic_results,bestK= which.max(aic_results)))
}
```

4.2. Model selection based on BIC score

It's same steps with computation of AIC, we need to find the number of model's parameters to compute BIC score:

$$\eta(GMM \text{ with K}) = \text{number of } \pi_k + \text{number of } \mu_k + \text{number of } \Sigma_k = (K-1) + K \times d + K \times \frac{d(d+1)}{2}$$

BIC formula:

$$BIC = log(\mathcal{L}(x;\phi)) - \frac{1}{2} \times log(n) \times \eta(GMM \text{ with K})$$

```
BIC <- function(log_likelihood, K, data) {
  # K is number of clusters
  # D is number of columns of Data
  n = nrow(data)
  d = ncol(data)
  ######
  #### Implementation of the Bayesian Information Criterion such that:
  #### BIC = LogLikelihood - (eta(M)*log(n))/2
  nb.param <- (K - 1) + K * d + K*d*(d+1)/2
  bic_score = log_likelihood - nb.param *log(n) / 2
  return (bic_score)
}

modelSelectionBIC <- function(data, listNumberCluster=2:10, print_steps=TRUE){
  ######
  #### Computes the BIC of an EM algorithm implementation given a dataset
  #### Input includes:
  #### data: Input data
  #### listNumberCluster: range of clusters for checking
  #### print_steps: Print process steps
  #### Output: result - result BIC scores, bestK - the best of number of cluster

  # Variable declaration
  bic_results = c()

  # Use for loop to compute the corresponding AIC on each cluster parameter and push to a list
  for (j in listNumberCluster){
    res = EMOrigin(data, j, showits=F)
    bic_value = BIC(res$logLik, j, data)
    bic_results = append(bic_results, bic_value)
    if (print_steps) {
      print(paste("Cluster ", j, "--- Log-likelihood = ",
                  round(bic_value, 3)))
    }
  }

  # Prints the result
  print(paste("The best K value is ", listNumberCluster[which.max(bic_results)] , " clusters (Based on
  return (list(result=bic_results,bestK= which.max(bic_results)))
}
```

4.3. Model selection based on Cross-validate on Log-likelihood

The general procedure of cross-validation k-fold is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    - Take the group as a hold out or test data set
    - Take the remaining groups as a training data set
    - Fit a model on the training set and evaluate it on the test set

- Retain the evaluation score and discard the model

4. Return the averaged of evaluation score

We will use a for loop from minimum number of clusters to maximum number of cluster to compute the averaged of evaluation score by cross-validation k-folds to find the best number of cluster (K).

```
crossValidation <- function(data,clusters=2, folds=10, seeds=143) {
  # Shuffle data
  set.seed(seeds)
  rows <- sample(nrow(data))
  shuffle_data = data[rows, ]
  # Variable declaration
  n_train = dim(shuffle_data)[1]
  fold_indexes = split(c(1:n_train),
                       ceiling(seq_along(c(1:n_train))/(n_train/folds)))
  listLogLiks = c()

  for (kFold in fold_indexes){
    # create train and validation dataset from created kFold indexes
    x_train = shuffle_data[-kFold,]
    x_val = shuffle_data[kFold,]
    # Get result cluster from EMOrigin with x_train
    resultsEM = EMOrigin(x_train, clusters,tol=1e-9,showits = F)
    valLogLik = metrics_computation(x_val, clusters, resultsEM$prob, resultsEM$mu, resultsEM$sigma)
    listLogLiks = append(listLogLiks, valLogLik$logLik)
  }
  #Return mean of log-likelihoods archived from validation data set during 10 Folds
  return(mean(listLogLiks))

}

modelSelectionCrossValidate <- function(data, listNumberCluster=2:10, print_steps=TRUE){
  ######
  #### Computes the BIC of an EM algorithm implementation given a dataset
  #### Input includes:
  #### data: Input data
  #### listNumberCluster: range of clusters for checking
  #### print_steps: Print process steps
  #### Output: result - result Cross-validation scores, bestK - the best of number of cluster

  # Variable declaration
  crossval_results = c()

  # Use for loop to compute the corresponding AIC on each cluster parameter and push to a list
  for (j in listNumberCluster){
    validationLogLik = crossValidation(data, cluster=j, folds=10)
    if (print_steps) {
      print(paste("Cluster ", j, "--- Log-likelihood = ",
                  round(validationLogLik, 3)))
    }
    # Append val log-likelihood to list result
    crossval_results = append(crossval_results, validationLogLik)
  }
```

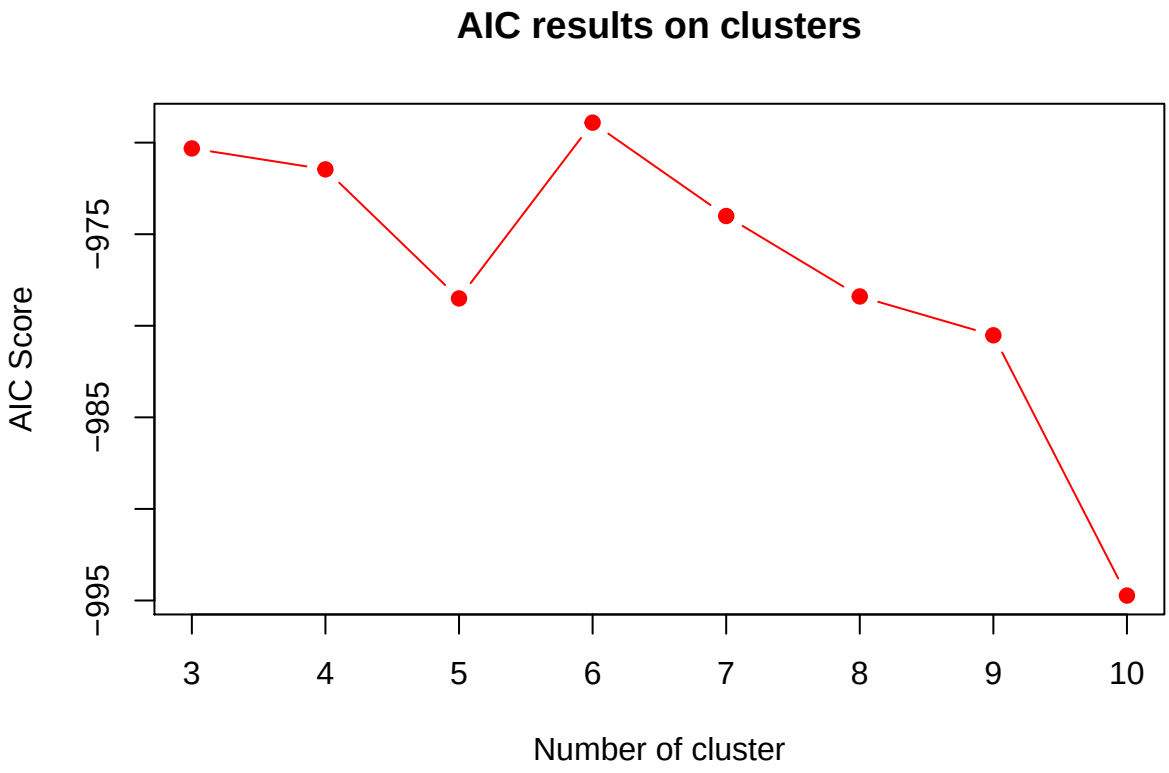
```
  # Prints the result
  print(paste("The best K value is ", listNumberCluster[which.max(crossval_results)] , " clusters (Base
  return (list(result=crossval_results,bestK= which.max(crossval_results)))
}
```

Let's apply AIC, BIC and cross-validation to find the best number of clusters in range from 3 to 10 clusters.

```
nbClusterRange = 3:10
aic_results = modelSelectionAIC(X, nbClusterRange)
```

```
## [1] "Cluster  3 --- Log-likelihood =  -970.316"
## [1] "Cluster  4 --- Log-likelihood =  -971.46"
## [1] "Cluster  5 --- Log-likelihood =  -978.507"
## [1] "Cluster  6 --- Log-likelihood =  -968.911"
## [1] "Cluster  7 --- Log-likelihood =  -974.008"
## [1] "Cluster  8 --- Log-likelihood =  -978.399"
## [1] "Cluster  9 --- Log-likelihood =  -980.521"
## [1] "Cluster  10 --- Log-likelihood =  -994.731"
## [1] "The best K value is  6  clusters (Based on AIC score)."
```

```
plot(nbClusterRange, aic_results$result, type = "b", pch = 19,
     col = "red", xlab = "Number of cluster", ylab = "AIC Score", main="AIC results on clusters")
```

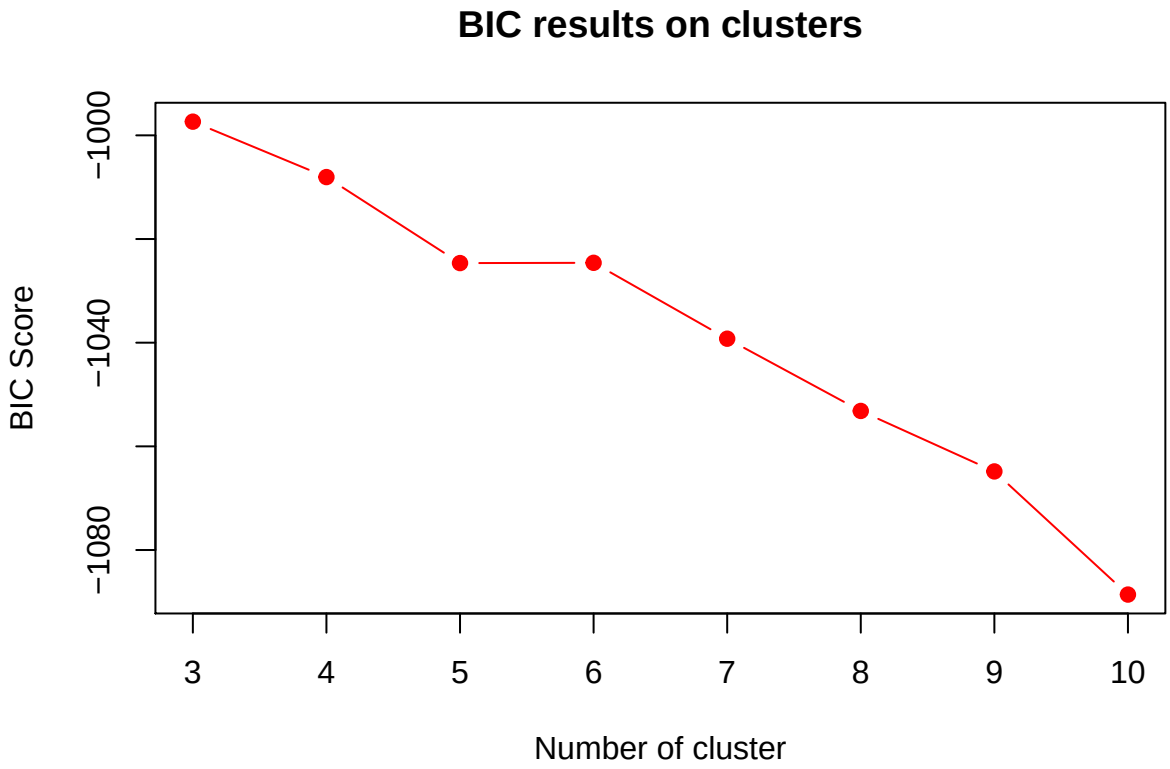


At first sight, AIC offers us a rather unstable result. According to the graph, the ideal number of clusters proposed by AIC is 6.

```
bic_results = modelSelectionBIC(X, nbClusterRange)
```

```
## [1] "Cluster  3 --- Log-likelihood =  -997.362"
## [1] "Cluster  4 --- Log-likelihood =  -1008.051"
## [1] "Cluster  5 --- Log-likelihood =  -1024.643"
## [1] "Cluster  6 --- Log-likelihood =  -1024.592"
## [1] "Cluster  7 --- Log-likelihood =  -1039.234"
## [1] "Cluster  8 --- Log-likelihood =  -1053.171"
## [1] "Cluster  9 --- Log-likelihood =  -1064.838"
## [1] "Cluster  10 --- Log-likelihood =  -1088.593"
## [1] "The best K value is  3  clusters (Based on BIC score)."
```

```
plot(nbClusterRange, bic_results$result, type = "b", pch = 19,
     col = "red", xlab = "Number of cluster", ylab = "BIC Score", main="BIC results on clusters")
```

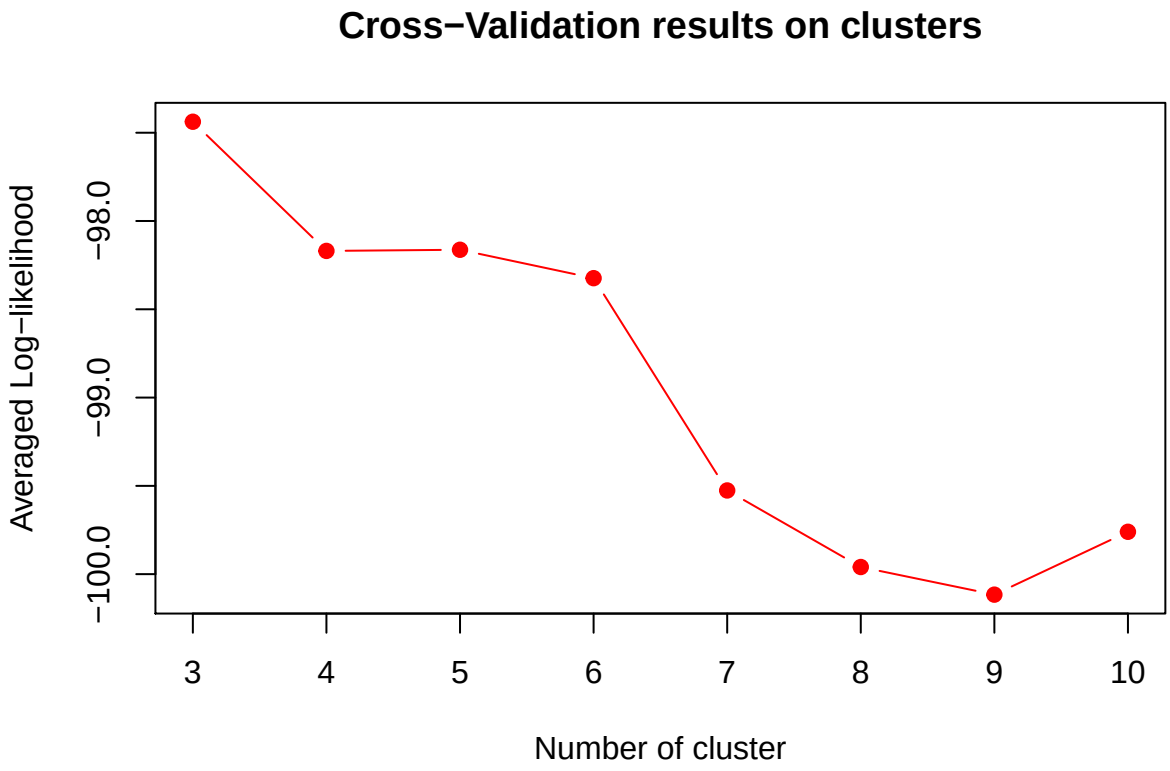


BIC gives us a much more stable result than AIC. According to the graph, the ideal number of clusters proposed by BIC is 3.

```
nbClusterRange = 3:10
crossval_results = modelSelectionCrossValidate(X, listNumberCluster = nbClusterRange)
```

```
## [1] "Cluster  3 --- Log-likelihood =  -97.438"
## [1] "Cluster  4 --- Log-likelihood =  -98.169"
## [1] "Cluster  5 --- Log-likelihood =  -98.163"
## [1] "Cluster  6 --- Log-likelihood =  -98.324"
## [1] "Cluster  7 --- Log-likelihood =  -99.526"
## [1] "Cluster  8 --- Log-likelihood =  -99.96"
## [1] "Cluster  9 --- Log-likelihood =  -100.116"
## [1] "Cluster  10 --- Log-likelihood =  -99.76"
## [1] "The best K value is  3  clusters (Based on Cross-validation on Log-likelihood)."
```

```
plot(nbClusterRange, crossval_results$result, type = "b", pch = 19,
     col = "red", xlab = "Number of cluster", ylab = "Averaged Log-likelihood", main="Cross-Validation r
```



Like BIC, Cross Validation gives us a much more stable result than AIC.

According to the graph, the ideal number of clusters proposed by BIC is 3.

4.4. Observation and comments

We see that cross validation and BIC both propose the same number of ideal clusters. They also have Averaged Log-likelihood results that drop progressively with the number of clusters. AIC is affected quite a lot when the number of clusters grows too high. This leads to a decrease in the reliability of the result due to its instability. Meanwhile, BIC increases the penalty when the number of clusters increases, making the result still stable

Often users will want the smallest possible number of clusters so that their data is consistent. Thereby we can see that the results on BIc and cross-validation are better than AIC.

5. Analysis on the higher dimensions of wine data set

5.1. Selecting features by hand

In this analysis, we will choose 3 features of wine data set to increase dimensions of data training. We will look at the impact of them on the below results. Because it is random I will take 3 even numbers 2, 4, 6 for the 3 features positions that I will take as input data.

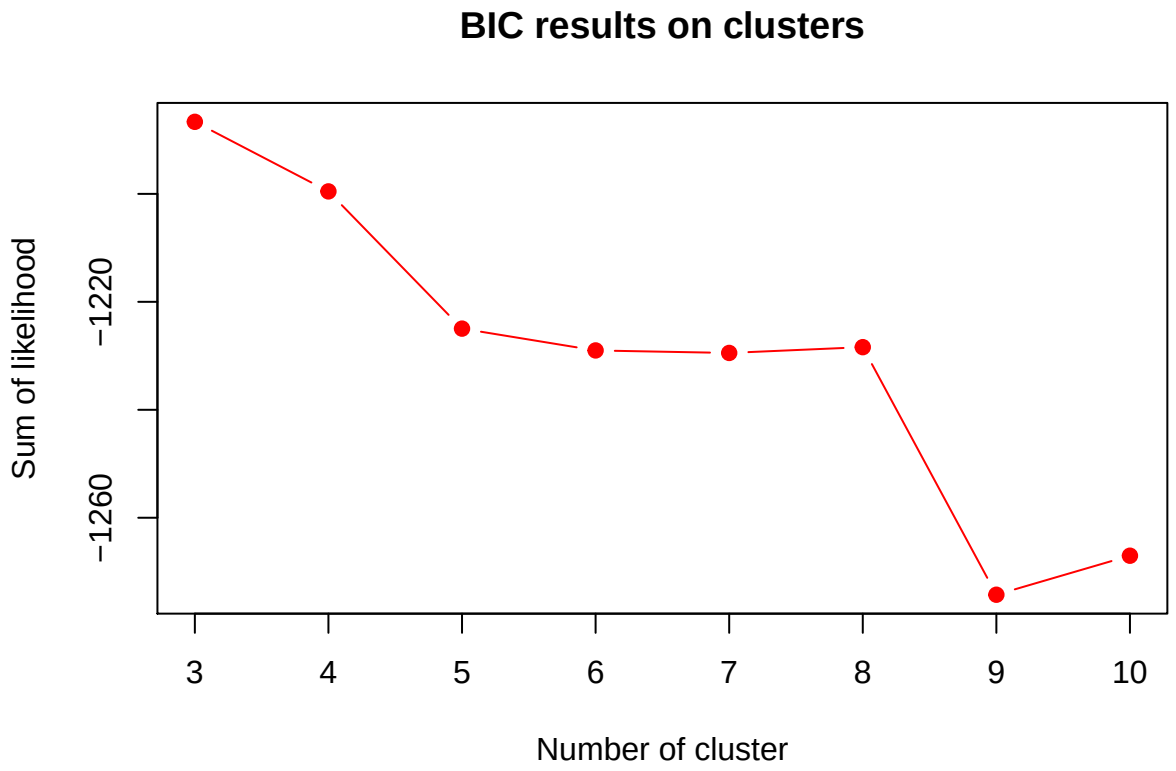
```
X_3_features = as.matrix(wine[,c(2,4,6)])
```

As we know in above part, BIC is also quite better than AIC in using to find the best possible number of clusters K. So we will use BIC to find the best number of cluster K for 3 features data. We don't use cross-validation k-folds because it takes too much time to find that.

```
nbClusterRange = 3:10
bic_3f_results = modelSelectionBIC(X_3_features, nbClusterRange)
```

```
## [1] "Cluster  3 --- Log-likelihood =  -1186.652"
## [1] "Cluster  4 --- Log-likelihood =  -1199.54"
## [1] "Cluster  5 --- Log-likelihood =  -1224.962"
## [1] "Cluster  6 --- Log-likelihood =  -1229.005"
## [1] "Cluster  7 --- Log-likelihood =  -1229.47"
## [1] "Cluster  8 --- Log-likelihood =  -1228.397"
## [1] "Cluster  9 --- Log-likelihood =  -1274.261"
## [1] "Cluster  10 --- Log-likelihood =  -1267.033"
## [1] "The best K value is  3  clusters (Based on BIC score)."
```

```
plot(nbClusterRange, bic_3f_results$result, type = "b", pch = 19,
     col = "red", xlab = "Number of cluster", ylab = "Sum of likelihood", main="BIC results on clusters
```



So now, we got the best K value is 3 clusters. we will train again EM model with 4 features data and 3 cluster. After that, we will visualize it to easy to see the result.

```
params_3f = EMOrigin(X_3_features, 3, 1000, showits = F)
params_3f
```

```
## $prob
## [1] 0.53385025 0.42880247 0.03734727
```

```
##
## $mu
##            X1       X2       X3
## [1,] 12.78735 97.18508 2.984710
## [2,] 13.27717 70.21885 1.601205
## [3,] 12.87396 97.75716 1.509068
##
## $sigma
## $sigma[[1]]
##                   Alcohol Fixed Acidity Malic Acid
## Alcohol       0.5288946      3.447949  0.3457571
## Fixed Acidity 3.4479490    279.100773 10.4653556
## Malic Acid    0.3457571     10.465356  1.3539491
##
## $sigma[[2]]
##                   Alcohol Fixed Acidity Malic Acid
## Alcohol       0.6096308      1.612237 0.12386585
## Fixed Acidity 1.6122366     47.135428 0.57537897
## Malic Acid    0.1238658      0.575379 0.08248408
##
## $sigma[[3]]
##                   Alcohol Fixed Acidity  Malic Acid
## Alcohol       1.4438310     1.6818572  0.12090461
## Fixed Acidity 1.6818572    17.3043098 -0.38679369
## Malic Acid    0.1209046    -0.3867937  0.02976898
##
##
## $gamma
##                [,1]          [,2]           [,3]
##   [1,] 0.0022995021 9.977005e-01 1.449253e-106
##   [2,] 0.0323785453 9.676215e-01  7.857664e-67
##   [3,] 0.8358303691 1.641696e-01  5.969897e-17
##   [4,] 0.0027767645 9.972232e-01  5.366359e-50
##   [5,] 0.9990166760 9.833240e-04  7.621102e-46
##   [6,] 0.0077559023 9.922441e-01  3.830794e-46
##   [7,] 0.0016612855 9.983387e-01 5.550800e-156
##   [8,] 0.0197002754 9.802997e-01  2.149017e-26
##   [9,] 0.0005679775 9.994320e-01 5.786361e-183
##  [10,] 0.0076673094 9.923327e-01 5.764812e-253
##  [11,] 0.0170461066 9.829539e-01  3.779582e-30
##  [12,] 0.0044256245 9.955744e-01 4.059432e-166
##  [13,] 0.0096568016 9.903432e-01  4.408275e-73
##  [14,] 0.0014349318 4.257888e-04  9.981393e-01
##  [15,] 0.0016953707 9.983046e-01  2.543963e-82
##  [16,] 0.0088922238 9.911078e-01  1.675366e-96
##  [17,] 0.0031231297 9.968769e-01  2.620546e-53
##  [18,] 0.0040410104 9.959590e-01 9.741119e-221
##  [19,] 0.0025740540 9.974259e-01 1.234815e-146
##  [20,] 0.9999999696 3.035001e-08 1.193802e-237
##  [21,] 0.0029129438 9.970871e-01 2.173288e-140
##  [22,] 1.0000000000 4.669851e-22  0.000000e+00
##  [23,] 0.0378783209 9.621217e-01  1.315803e-24
##  [24,] 0.0306671750 9.693328e-01 1.520575e-122
##  [25,] 0.0885418873 9.114581e-01  2.154880e-20
```

```
##  [26,] 0.4278709080 5.721291e-01  9.239690e-09
##  [27,] 0.0180581542 9.819418e-01  5.123822e-77
##  [28,] 0.0141188342 9.858812e-01 3.526274e-119
##  [29,] 0.0063336473 9.936664e-01  2.644808e-94
##  [30,] 0.0028207584 9.971792e-01 1.107447e-140
##  [31,] 0.0116519456 9.883481e-01 3.014449e-123
##  [32,] 0.0076075706 9.923924e-01 5.450431e-216
##  [33,] 0.0081829725 9.918170e-01 3.862689e-137
##  [34,] 0.0147210249 9.852790e-01  1.291112e-99
##  [35,] 0.0108662468 9.891338e-01 2.234582e-134
##  [36,] 0.0143191383 9.856809e-01  5.644693e-82
##  [37,] 0.0584520226 9.415480e-01  1.677752e-52
##  [38,] 0.0252779599 9.747220e-01 8.786003e-105
##  [39,] 0.0320854226 9.679146e-01 2.250110e-114
##  [40,] 1.0000000000 4.403660e-23  0.000000e+00
##  [41,] 0.0445504891 9.554495e-01  3.429552e-39
##  [42,] 1.0000000000 7.466212e-21  0.000000e+00
##  [43,] 0.0196504974 9.803495e-01  1.450651e-28
##  [44,] 1.0000000000 1.647632e-23  0.000000e+00
##  [45,] 0.1715921843 8.284078e-01  6.749413e-25
##  [46,] 1.0000000000 3.925008e-21  0.000000e+00
##  [47,] 1.0000000000 2.822799e-11  0.000000e+00
##  [48,] 0.0065789713 9.934210e-01  2.859251e-86
##  [49,] 0.0060218708 9.939781e-01  6.559647e-64
##  [50,] 0.0044165103 9.955835e-01  4.946537e-97
##  [51,] 0.0659041061 9.340959e-01  4.444508e-51
##  [52,] 0.0042779327 9.957221e-01 9.524244e-214
##  [53,] 0.0076135788 9.923864e-01  2.291808e-75
##  [54,] 0.0087967694 9.912032e-01 4.075828e-122
##  [55,] 0.0047764574 9.952235e-01 6.840569e-181
##  [56,] 0.0268204175 9.731796e-01  2.083476e-49
##  [57,] 0.0624665948 9.375334e-01  8.208794e-19
##  [58,] 0.2081915031 7.918085e-01  8.562984e-11
##  [59,] 0.0107742394 9.892258e-01 1.275284e-161
##  [60,] 0.9976018546 2.398145e-03  8.901106e-86
##  [61,] 0.1691070165 8.308930e-01 2.663200e-224
##  [62,] 0.9980357809 1.887716e-03  7.650306e-05
##  [63,] 0.0142918708 9.857081e-01  0.000000e+00
##  [64,] 0.1051480631 8.948519e-01 2.499086e-244
##  [65,] 0.0604098553 9.395901e-01 3.725291e-181
##  [66,] 0.0483730061 9.516270e-01 7.710181e-318
##  [67,] 0.6037461233 3.962539e-01 2.108804e-191
##  [68,] 0.0549627313 9.450373e-01 3.345467e-319
##  [69,] 0.2254019108 7.745981e-01  0.000000e+00
##  [70,] 0.9926798610 7.320139e-03  2.214599e-33
##  [71,] 0.1284731872 8.715268e-01 1.839821e-259
##  [72,] 0.0062476959 9.937523e-01 2.209462e-320
##  [73,] 0.0099233911 9.900766e-01 4.914302e-226
##  [74,] 0.1484644179 8.515356e-01  0.000000e+00
##  [75,] 0.1031206974 8.968793e-01 1.807223e-300
##  [76,] 0.7398344033 2.601656e-01  8.343104e-91
##  [77,] 0.7554850048 2.445150e-01 6.288958e-244
##  [78,] 1.0000000000 5.763783e-16  0.000000e+00
##  [79,] 0.1275408394 8.724592e-01  0.000000e+00
```

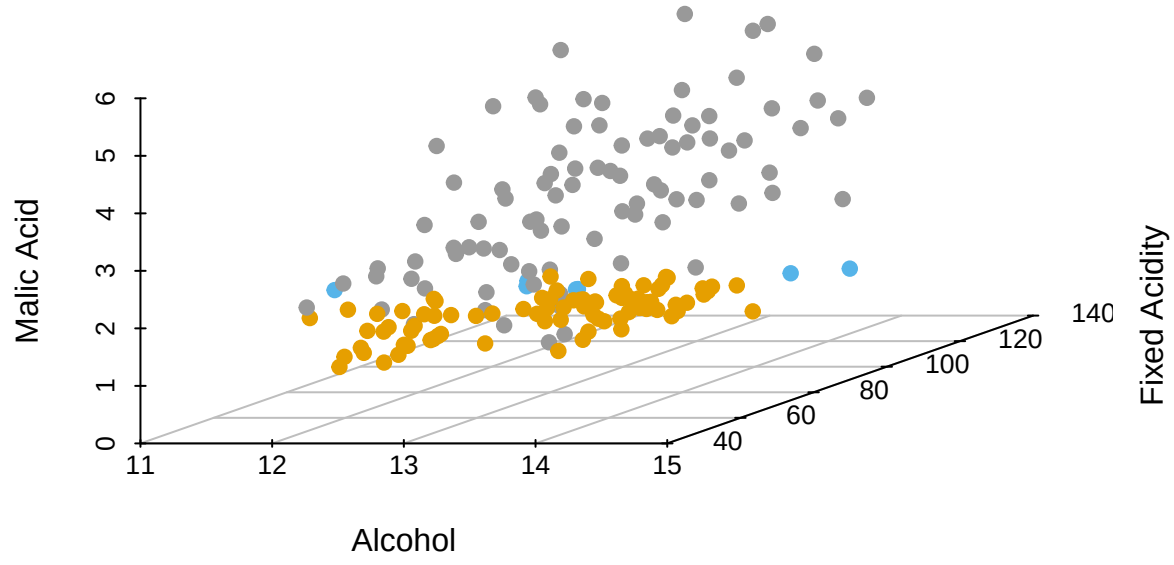
```
##  [80,] 1.0000000000 2.493347e-22  0.000000e+00
##  [81,] 0.1849375636 8.150624e-01  0.000000e+00
##  [82,] 0.1148711695 8.851288e-01 8.073706e-176
##  [83,] 0.0887295817 9.112704e-01  0.000000e+00
##  [84,] 1.0000000000 3.406363e-23  0.000000e+00
##  [85,] 0.4183446628 5.816553e-01 1.005263e-299
##  [86,] 0.3436535015 6.563465e-01 2.628315e-258
##  [87,] 0.4917819952 5.082180e-01  3.767870e-35
##  [88,] 0.3057587239 6.942413e-01 5.708317e-129
##  [89,] 0.9799873244 2.001268e-02  1.061166e-11
##  [90,] 0.0714728198 9.285272e-01 3.251156e-193
##  [91,] 0.8053899700 1.946100e-01  4.670196e-12
##  [92,] 0.3015274367 6.984726e-01  4.129810e-68
##  [93,] 0.0430580949 9.569419e-01 9.791444e-123
##  [94,] 0.9999999843 1.568341e-08 4.816314e-143
##  [95,] 0.9999659365 3.406349e-05  2.885757e-31
##  [96,] 0.0744697020 9.255303e-01 4.857873e-108
##  [97,] 0.9895476890 1.044867e-02  3.641752e-06
##  [98,] 0.0589459643 9.410540e-01 2.158442e-166
##  [99,] 0.0794769126 9.205231e-01  0.000000e+00
## [100,] 1.0000000000 7.491963e-19  0.000000e+00
## [101,] 0.9517760227 4.822393e-02  5.112221e-08
## [102,] 0.0306303995 9.693696e-01 2.370904e-263
## [103,] 0.9995965504 4.034496e-04  1.584196e-20
## [104,] 0.3303456599 6.696543e-01  9.035633e-85
## [105,] 0.0821889348 9.178111e-01 1.045462e-139
## [106,] 0.9999132900 8.671004e-05  6.495007e-33
## [107,] 0.3441943303 6.558057e-01  6.960385e-40
## [108,] 0.7662559920 2.337301e-01  1.386691e-05
## [109,] 0.0813968659 9.186031e-01 3.103440e-182
## [110,] 0.9257570290 7.424297e-02  1.695984e-34
## [111,] 1.0000000000 9.656870e-28  0.000000e+00
## [112,] 0.9999998841 1.159086e-07 3.512815e-161
## [113,] 0.9999999967 3.278766e-09 1.960623e-160
## [114,] 0.9999997809 2.191418e-07  1.046040e-36
## [115,] 0.5627640604 2.925176e-04  4.369434e-01
## [116,] 0.0657456598 1.551809e-05  9.342388e-01
## [117,] 0.3420798262 6.579202e-01  3.321195e-74
## [118,] 0.2473290487 5.481904e-04  7.521228e-01
## [119,] 1.0000000000 4.553241e-17  0.000000e+00
## [120,] 1.0000000000 1.946801e-17  0.000000e+00
## [121,] 1.0000000000 5.723766e-19  0.000000e+00
## [122,] 0.9994087474 5.912526e-04  5.542525e-11
## [123,] 1.0000000000 4.518436e-38  0.000000e+00
## [124,] 1.0000000000 1.446878e-70  0.000000e+00
## [125,] 1.0000000000 1.477773e-40  0.000000e+00
## [126,] 0.9999908332 9.166814e-06  4.561069e-64
## [127,] 0.2389372510 8.448654e-04  7.602179e-01
## [128,] 0.9999988447 1.155275e-06  4.402189e-78
## [129,] 0.9999896147 1.038528e-05  1.136900e-17
## [130,] 1.0000000000 7.706611e-38  0.000000e+00
## [131,] 0.8047347765 1.952652e-01  8.293691e-40
## [132,] 0.9999999125 8.751930e-08 9.360147e-105
## [133,] 0.9583669434 4.163245e-02  6.108725e-07
```

```
## [134,] 1.0000000000 6.743823e-17  0.000000e+00
## [135,] 0.1439037015 2.268466e-07  8.560961e-01
## [136,] 0.9993277731 6.722269e-04  1.870778e-26
## [137,] 1.0000000000 2.278258e-41  0.000000e+00
## [138,] 1.0000000000 7.170254e-62  0.000000e+00
## [139,] 1.0000000000 1.089330e-20  0.000000e+00
## [140,] 0.9999999256 7.441617e-08 6.567579e-124
## [141,] 0.9999938976 6.102442e-06  8.527025e-72
## [142,] 0.9999912638 8.736161e-06 3.619841e-142
## [143,] 0.9999999813 1.868060e-08 7.400973e-191
## [144,] 1.0000000000 6.231100e-40  0.000000e+00
## [145,] 1.0000000000 1.355503e-29  0.000000e+00
## [146,] 1.0000000000 1.289907e-22  0.000000e+00
## [147,] 1.0000000000 1.741122e-41  0.000000e+00
## [148,] 1.0000000000 6.134776e-37  0.000000e+00
## [149,] 1.0000000000 8.690376e-12  0.000000e+00
## [150,] 1.0000000000 7.990270e-29  0.000000e+00
## [151,] 1.0000000000 5.216490e-12  0.000000e+00
## [152,] 1.0000000000 1.693661e-14  0.000000e+00
## [153,] 0.9938804865 6.119513e-03  3.350199e-11
## [154,] 1.0000000000 4.760240e-20  0.000000e+00
## [155,] 0.0838147973 9.241218e-07  9.161843e-01
## [156,] 1.0000000000 3.775182e-58  0.000000e+00
## [157,] 1.0000000000 9.061195e-30  0.000000e+00
## [158,] 1.0000000000 1.322600e-14  0.000000e+00
## [159,] 0.0053689593 8.541233e-04  9.937769e-01
## [160,] 0.9995548627 4.451135e-04  2.384865e-08
## [161,] 1.0000000000 2.427473e-29  0.000000e+00
## [162,] 0.9999999726 2.743514e-08 5.099236e-115
## [163,] 1.0000000000 5.726428e-12 4.779850e-281
## [164,] 1.0000000000 7.410122e-16  0.000000e+00
## [165,] 0.9999998277 1.722989e-07 5.405001e-258
## [166,] 1.0000000000 1.331552e-26  0.000000e+00
## [167,] 1.0000000000 2.099267e-15  0.000000e+00
## [168,] 1.0000000000 3.789430e-22  0.000000e+00
## [169,] 1.0000000000 1.694431e-12  0.000000e+00
## [170,] 1.0000000000 1.834957e-47  0.000000e+00
## [171,] 1.0000000000 7.488069e-16  0.000000e+00
## [172,] 0.9999999998 2.487476e-10 3.507040e-272
## [173,] 0.9999999993 7.189265e-10  0.000000e+00
## [174,] 1.0000000000 1.544433e-61  0.000000e+00
## [175,] 1.0000000000 8.048951e-30  0.000000e+00
## [176,] 1.0000000000 2.488480e-29  0.000000e+00
## [177,] 1.0000000000 2.369470e-11  0.000000e+00
## [178,] 1.0000000000 2.854534e-29  0.000000e+00
##
## $logLik
## [1] -1108.847
```

Look at the result of clustering, with 3 cluster, we can see the probabilities of each cluster are 0.53385025, 0.42880247, 0.03734727

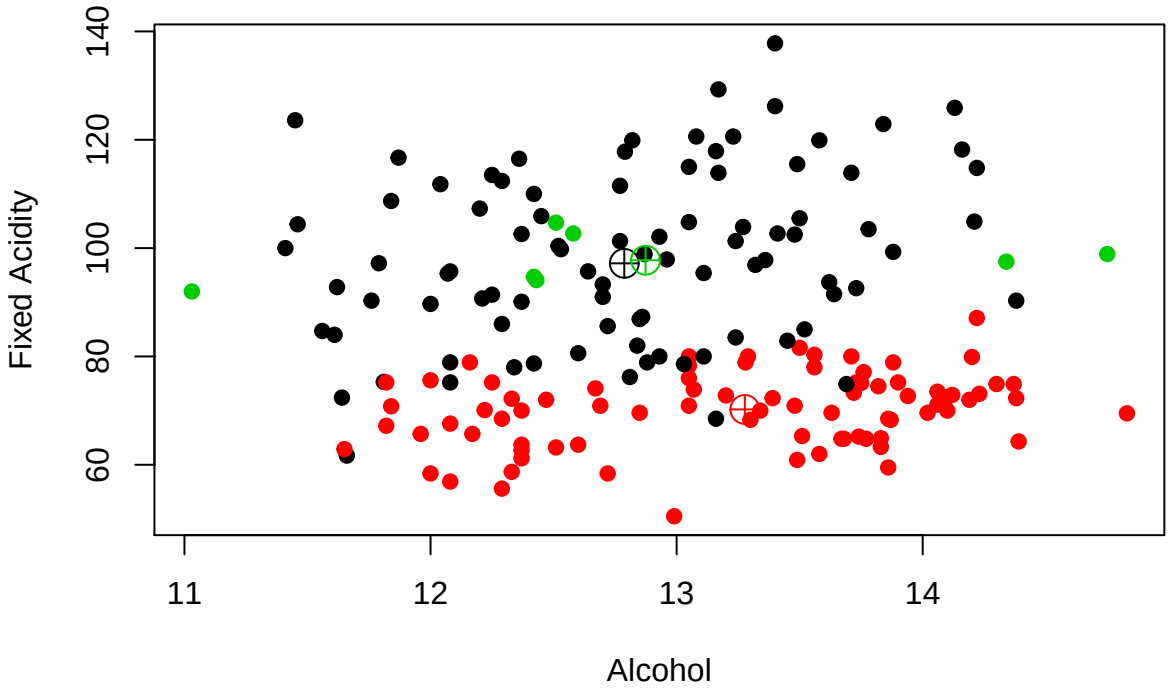
We will plot 3D graph to see all data with 3-dimensions of data.

```
cluster_3f_result = EMOrigin.findCluster(params_3f$gamma)
colors <- c("#999999", "#E69F00", "#56B4E9")
colors <- colors[as.numeric(cluster_3f_result)]
scatterplot3d(X_3_features, pch = 19, color=colors,box=FALSE)
```



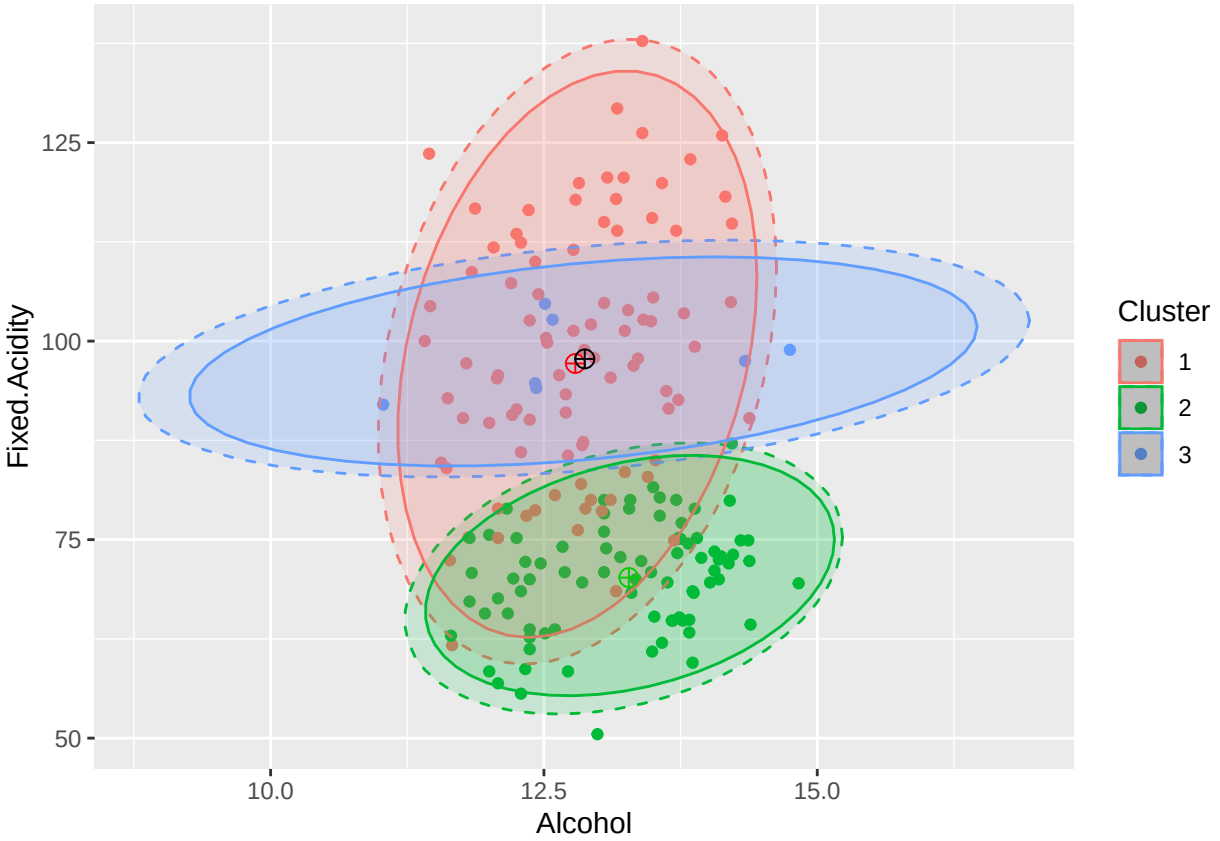
Now we can draw a plot on 2 variables we used in above part to visualize the impact when we increase number of data dimension.

```
plot(X_3_features[,c(1,2)], col=cluster_3f_result, pch=19)
points(params_3f$mu[,c(1,2)], col = 1:3, pch = 10, cex=2)
```



Plot cluster area with data:

```
plot_cluster_area(X_3_features[,c(1,2)], params_3f$mu[,c(1,2)], cluster_3f_result)
```



We can comoute the classification error rate by classError() function in mclust library:

```
classError(cluster_3f_result, y)
```

```
## $misclassified
##  [1]   3   5  14  20  22  40  42  44  46  47  60  61  62  63  64  65  66  67  68
## [20]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87
## [39]  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106
## [58] 107 108 109 110 111 112 113 114 115 117 119 120 121 122 123 124 125 126 128
## [77] 129 130 135 155 159
##
## $errorRate
## [1] 0.4550562
```

Comment: After many times of experimenting with changing input features of the data, we have come to a conclusion that: Increasing the number of dimensions of the data sometimes actually helps to increase the cluster ratio. However, there are many cases where the added features are not useful enough in data clustering. When adding data feature that has no utility or impact on another features, the result is that one cluster will be very small compared to the other two. This situation leads to when using BIC, AIC, or Cross-validation to find the best number of clusters is no longer accurate. The fact that a cluster is almost invisible in the data set will cause uncertainty in the calculation of BIC, AIC, and average log-likelihood from the training results because it is too small to be estimated.

5.2. Generate random higher-dimension data

To have an objective and overall view, instead of manually randomly taking the features of the data, we will automatically generate random for 3 cases 3-Dimensions data, 4-Dimensions data and 5-Dimensions data. Each case, we will create random 5 samples to see the best number of cluster and classification error on this

```r
set.seed(110)
#110
generateHigherDimSample <-function(data, n){
  max_features = ncol(data)
  results = lapply(1:5, function(i) sort(sample(1:max_features, n, replace=F)))
  return(results)
}
list_features_5d = generateHigherDimSample(wine[,-1], 5)
list_features_4d = generateHigherDimSample(wine[,-1], 4)
list_features_3d = generateHigherDimSample(wine[,-1], 3)
```

Now, we will analyze in 3-Dimensions, 4-Dimensions and 5-Dimensions data sets which generated above.

5.2.1. 3-Dimensions datasets

```r
nbClusterRange = 3:10
headers = colnames(wine[,-1])
list_K_3d_result = c()
for (i in 1:length(list_features_3d)){
  x_3d = as.matrix(wine[, list_features_3d[[i]]])
  cat("Feature names: ", paste( unlist(headers[list_features_3d[[i]]]), collapse=' ______ '), "\n")
  resultBIC = modelSelectionBIC(x_3d, nbClusterRange, print_steps=F)
  resultAIC = modelSelectionAIC(x_3d, nbClusterRange, print_steps=F)

  list_K_3d_result = append(list_K_3d_result, resultBIC$bestK)
  print("-----------------------")
}
```

```
## Feature names:  Uronic Acids ______ Ash ______ Methanol
## [1] "The best K value is  3  clusters (Based on BIC score)."
## [1] "The best K value is  10  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  Sugar-free Extract ______ Tartaric Acid ______ Methanol
## [1] "The best K value is  9  clusters (Based on BIC score)."
## [1] "The best K value is  9  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  Uronic Acids ______ Alcalinity of Ash ______ Chloride
## [1] "The best K value is  3  clusters (Based on BIC score)."
## [1] "The best K value is  9  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  pH ______ OD280/OD315 of Diluted Wines ______ Total Nitrogen
## [1] "The best K value is  3  clusters (Based on BIC score)."
## [1] "The best K value is  9  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  Fixed Acidity ______ Malic Acid ______ Hue
## [1] "The best K value is  3  clusters (Based on BIC score)."
## [1] "The best K value is  10  clusters (Based on AIC score)."
## [1] "-----------------------"
```

We can see that if we based on BIC scores to find the best number of cluster, the result will be almost around 3 or 4. However, some cases we will get some high number of cluster. This is unstable because the selected features are not related to others and make clustering more difficult. The high number of clusters shows that the input data is difficult to cluster, so it has to be divided into many different small clusters. Besides, achieved number

Let's see the result of 4-Dimensions datasets.

5.2.2. 4-Dimensions datasets

```
list_K_4d_result = c()
for (i in 1:length(list_features_4d)){
  x_4d = as.matrix(wine[, list_features_4d[[i]]])
  cat("Feature names: ", paste( unlist(headers[list_features_4d[[i]]]), collapse=' ______ '), "\n")
  resultBIC = modelSelectionBIC(x_4d, nbClusterRange, print_steps=F)
  resultAIC = modelSelectionAIC(x_4d, nbClusterRange, print_steps=F)
  list_K_4d_result = append(list_K_4d_result, resultBIC$bestK)
  print("-----------------------")
}
```

```
## Feature names:  Chloride ______ Proanthocyanins ______ Glycerol ______ Proline
## [1] "The best K value is  7  clusters (Based on BIC score)."
## [1] "The best K value is  7  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  Fixed Acidity ______ Alcalinity of Ash ______ Magnesium ______ Chloride
## [1] "The best K value is  4  clusters (Based on BIC score)."
## [1] "The best K value is  4  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  pH ______ Calcium ______ OD280/OD315 of Diluted Wines ______ 2-3-Butanediol
## [1] "The best K value is  4  clusters (Based on BIC score)."
## [1] "The best K value is  4  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  Tartaric Acid ______ Potassium ______ Non-flavanoid Phenols ______ Total Nitrogen
## [1] "The best K value is  10  clusters (Based on BIC score)."
## [1] "The best K value is  10  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  Alcohol ______ Malic Acid ______ Alcalinity of Ash ______ 2-3-Butanediol
## [1] "The best K value is  5  clusters (Based on BIC score)."
## [1] "The best K value is  5  clusters (Based on AIC score)."
## [1] "-----------------------"
```

5.2.3. 5-Dimensions datasets

```
list_K_5d_result = c()
for (i in 1:length(list_features_5d)){
  x_5d = as.matrix(wine[, list_features_5d[[i]]])
  cat("Feature names: ", paste( unlist(headers[list_features_5d[[i]]]), collapse=' ______ '), "\n")
  resultBIC = modelSelectionBIC(x_5d, nbClusterRange, print_steps=F)
  resultAIC = modelSelectionAIC(x_5d, nbClusterRange, print_steps=F)
  list_K_5d_result = append(list_K_5d_result, resultBIC$bestK)
  print("-----------------------")
}
```

```
## Feature names:  Uronic Acids ______ Alcalinity of Ash ______ Color Intensity ______ Hue ______ 2-3-Bu
## [1] "The best K value is  9  clusters (Based on BIC score)."
## [1] "The best K value is  9  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  Fixed Acidity ______ Tartaric Acid ______ pH ______ Alcalinity of Ash ______ Flavano
## [1] "The best K value is  8  clusters (Based on BIC score)."
## [1] "The best K value is  8  clusters (Based on AIC score)."
```

```
## [1] "-----------------------"
## Feature names:  Alcohol ______ Potassium ______ Color Intensity ______ 2-3-Butanediol ______ Methanol
## [1] "The best K value is  8  clusters (Based on BIC score)."
## [1] "The best K value is  8  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  pH ______ Alcalinity of Ash ______ OD280/OD315 of Diluted Wines ______ Total Nitrogen
## [1] "The best K value is  8  clusters (Based on BIC score)."
## [1] "The best K value is  8  clusters (Based on AIC score)."
## [1] "-----------------------"
## Feature names:  Potassium ______ Magnesium ______ OD280/OD315 of Flavanoids ______ Proline ______ Me
## [1] "The best K value is  6  clusters (Based on BIC score)."
## [1] "The best K value is  6  clusters (Based on AIC score)."
## [1] "-----------------------"
```

5.3. Observation and comments

As we have seen the results, increasing the data dimension by adding features will sometimes fragment the data and make clustering difficult. On the printed results we can see that for each dataset with the same dimension we get a very different number of clusters: At a dimensional count of 3, we get some 3 clusters results. - the difference is not too big with the dimension of 3. In the cases of 4 and 5, the results are completely different and the number of clusters is high.

In the process of adding features to train the clustering, some features accidentally fragmented the data, breaking the consistency of the data. Thereby showing us the importance of data analysis and selection, which has a great influence on the results of clustering.

6. Conclusion

In this project, we have successfully built EM for GMM algorithm and applied it to wine. Through analyzing the obtained results as well as comparing with the K-means algorithm, we have come to some conclusions as follows:

With the application of the above two algorithms to cluster alcohol based on Alcohol and Fixed Acidity of alcohol, the EM algorithm gives better clustering results than K-means. The results obtained for the given Wine type from the data are more similar than the results from K-means.

Second, The use of criteria to select the number of clusters is very important. In this project, we tested on 3 criteria which are AIC score, BIC score and cross-validation on likelihood. The obtained results show that the BIC score and cross-validation are stable and are not affected too much when the number of clusters increases. The AIC score shows instability and is not really reliable.

Third, increasing the number of dimensions of the data will sometimes fragment the data and cause difficulties in the clustering process. The number of clusters increases when we use unhelpful features from the data. Care should be taken in selecting and analyzing features before adding input data.