LAB 1 BLOCK 2: ENSEMBLE METHODS AND MIXTURE MODELS

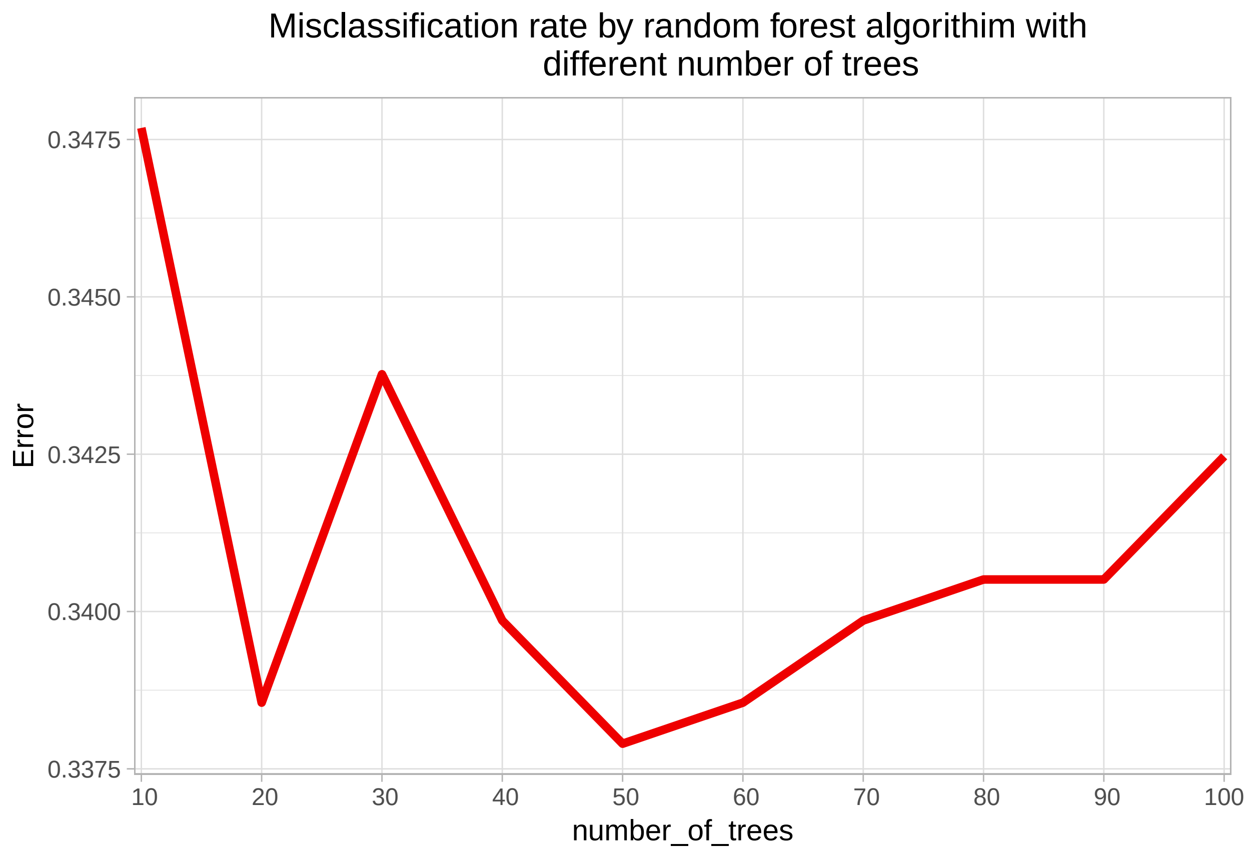
Nahid Farazmand (nahfa911)

December 2, 2018

1. ENSEMBLE METHODS

In this assignment we want to compare the performance of two different ensemble learning algorithms which use decision trees as as the weak learners. For this purpose, at first we tried random forest algorithm with different number of trees and investigated the optimum number of trees for getting the best fitted model.

Random forest



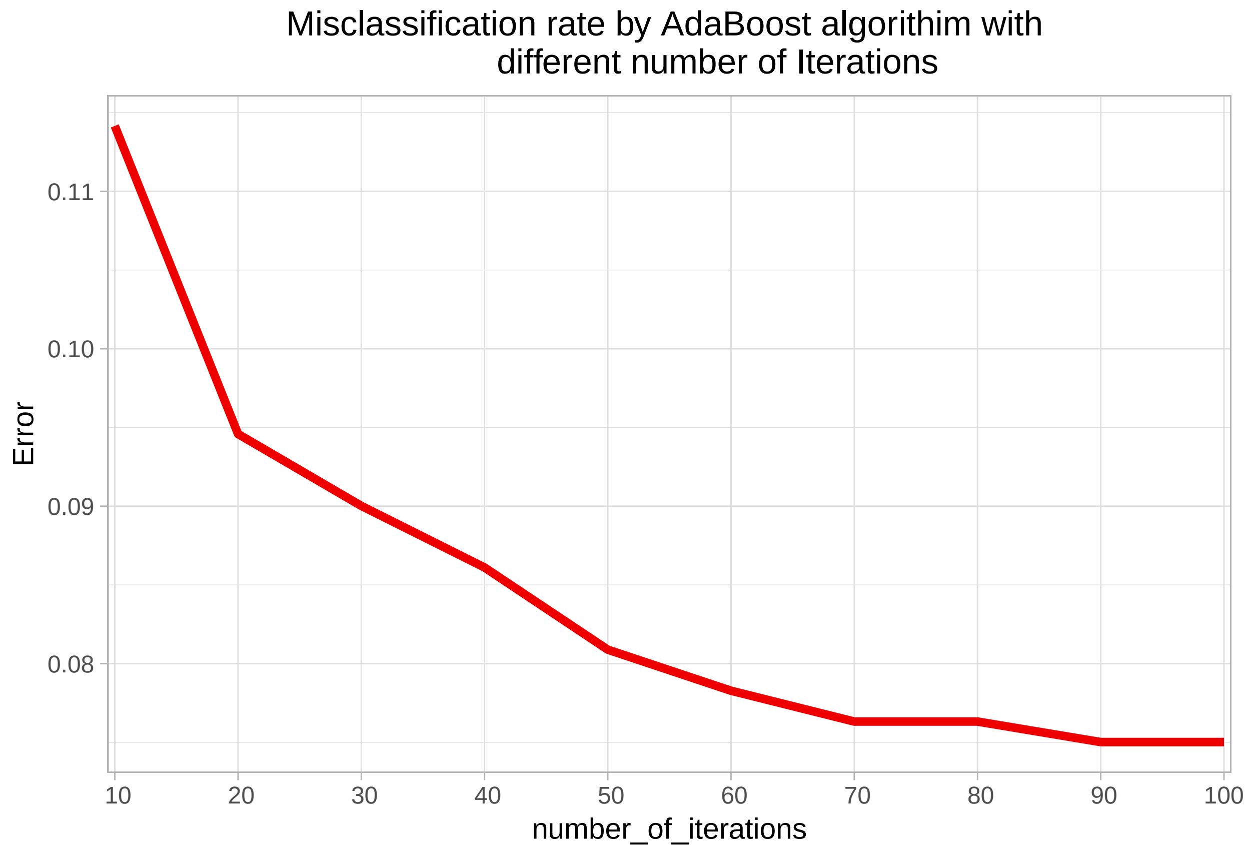
As can be clearly seen in this plot, when the number of trees is equal to 70, the algorithm has the lowest minclassification rate; so we will use this model for comparing the performance of the random forest versus AdaBoost.

The least rate of misclassification rate for the test dataset by using random forest (number of trees are equal to 50)

I increased the number of trees to 500 and the result did not improved significantly and the error rate stayed around 0.3.

```
## [1] 0.3378995
```

Adaboost

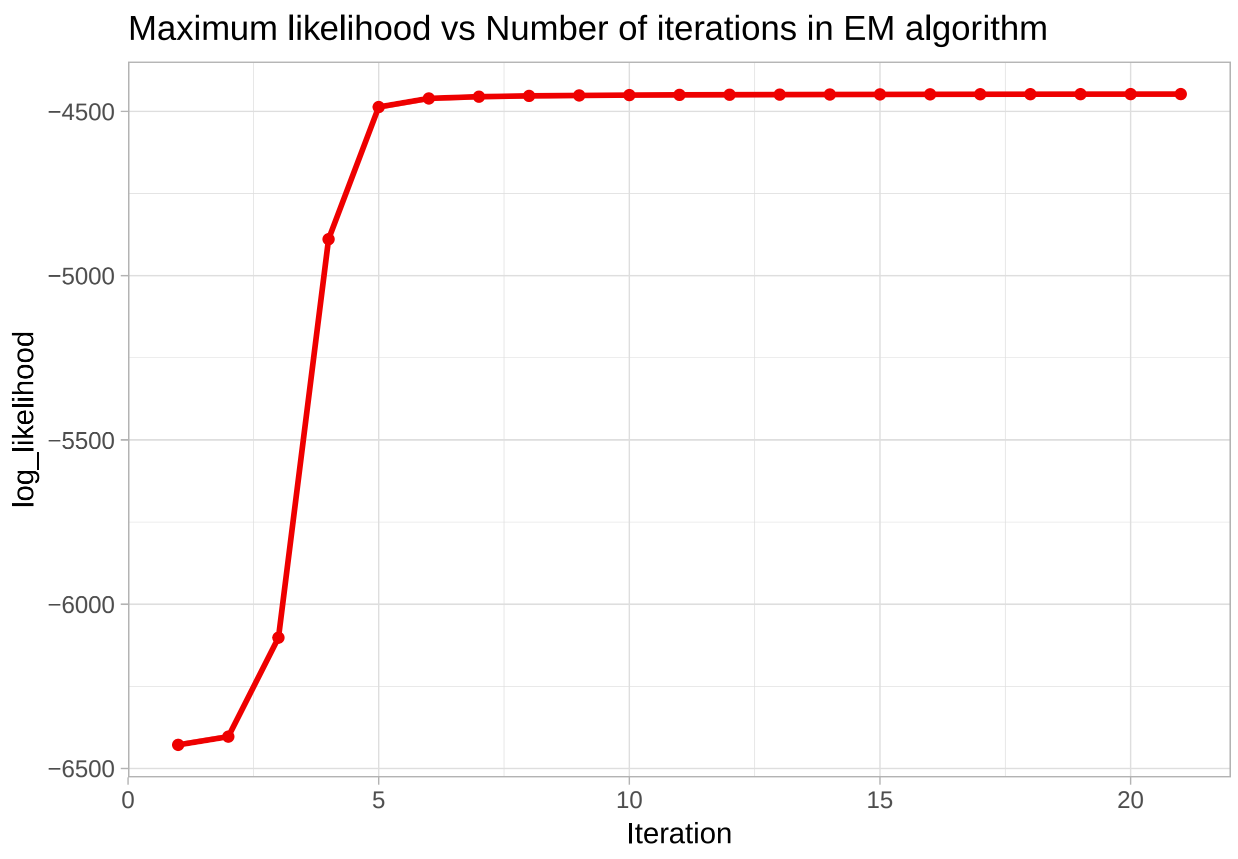


By using Adaboost algorithm, the misclassification rate was as follow. We can see that this rate is far less that the least error that we got by using random forest algorithm (even with 10 iteration); So it seems that this algorithm is more trustable! Moreover, we can see thet without setting a seed, the result of the random forest - in terms of the best number of trees - changes; however, the error rate does not change significantly and the error for all of the number of trees is around 34%.

2. MIXTURE MODELS

In this assignment we want to implement the EM algorithm for mixtures of multivariate Benoulli distributions. And to be able to investigate the performance of the algorithm, we will use the algprithm to fit the data to a mixture of k = 2, k = 33 and k = 4 Bernoulli distributions.

k = 2



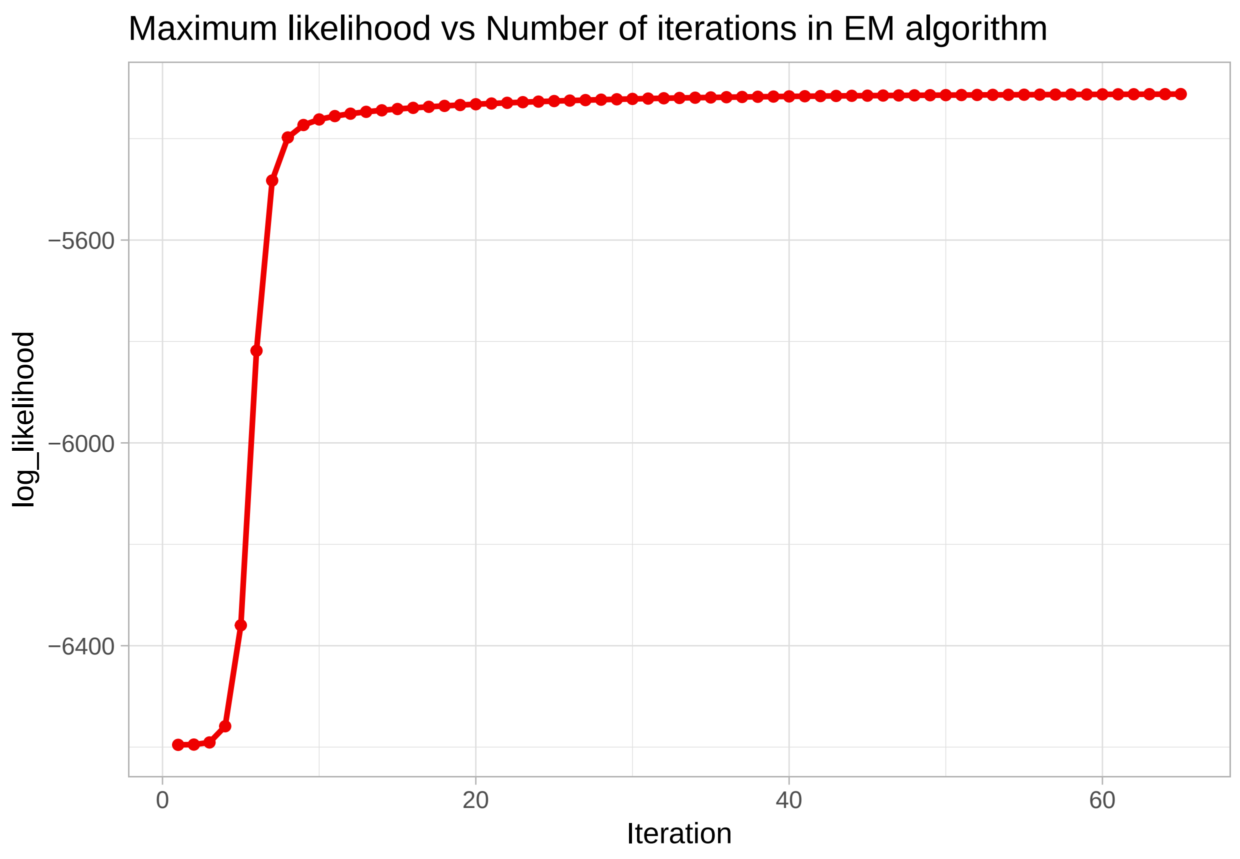
The proportion of distributions is as follow:

```
## [1] 0.5110531 0.4889469
```

The probability of being 1, for each variable of each observation in different distributions ie as follow:

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4931735 0.3974606 0.5967811 0.2785480 0.6927917 0.2184957 0.8018491
## [2,] 0.4989543 0.6255823 0.3804363 0.7171478 0.3230343 0.7778699 0.2049559
##           [,8]       [,9]       [,10]
## [1,] 0.1116477 0.88054439 0.004290353
## [2,] 0.9140913 0.08997919 0.999714736
```

k = 3



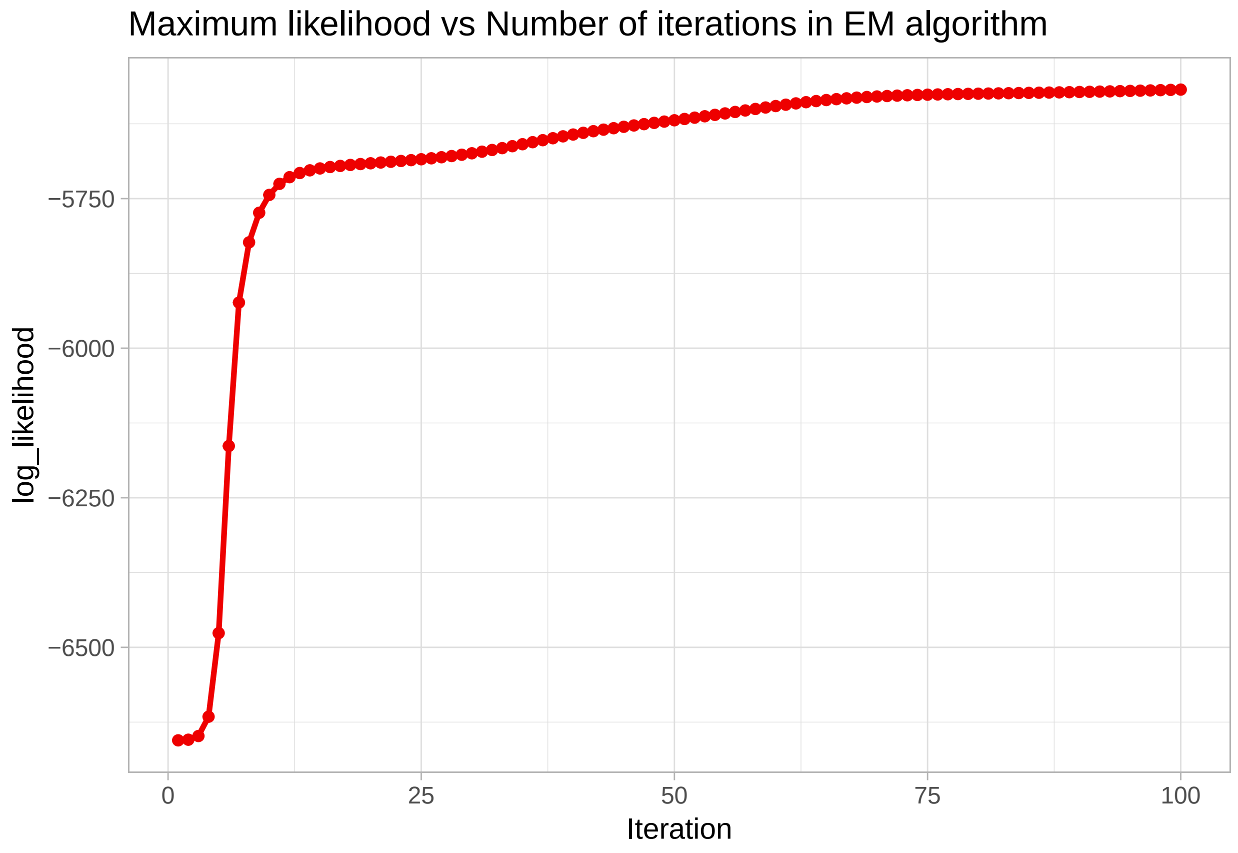
The proportion of distributions is as follow:

```
## [1] 0.3253079 0.3054648 0.3692273
```

The probability of being 1, for each variable of each observation in different distributions ie as follow:

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4737641 0.3814787 0.6287272 0.3083381 0.6946440 0.1979384 0.7880918
## [2,] 0.4908688 0.4794407 0.4696385 0.4792674 0.5328539 0.4924305 0.4649473
## [3,] 0.5090026 0.5835152 0.4198282 0.7158080 0.2904951 0.7669771 0.2318423
##           [,8]      [,9]      [,10]
## [1,] 0.1347064 0.8915450 0.01871995
## [2,] 0.4899193 0.4926427 0.39851709
## [3,] 0.8518488 0.1069488 0.99986452
```

k = 4



The proportion of distributions is as follow:

```
## [1] 0.2880470 0.2533761 0.2933710 0.1652060
```

The probability of being 1, for each variable of each observation in different distributions ie as follow:

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.3714855 0.3899958 0.4790260 0.5731886 0.5022651 0.5108478 0.2835691
## [2,] 0.5199997 0.6135841 0.3891214 0.7132736 0.2722448 0.7785461 0.2168891
## [3,] 0.4383456 0.4042497 0.5489526 0.3298363 0.6578057 0.2049012 0.7825505
## [4,] 0.3428531 0.7784238 0.5591637 0.6319621 0.5167044 0.4629058 0.7311279
##           [,8]       [,9]      [,10]
## [1,] 0.3519184 0.36924863 0.48252239
## [2,] 0.9337959 0.08504806 0.99916297
## [3,] 0.1703330 0.80517853 0.04500171
## [4,] 0.6601375 0.46532151 0.48814639
```

Comaring the results with different amount of k, we can see that when the number of distributoins increases the log of likelihood decreases; so in this problem, it seems that the model with k = 2 can be a better representative of the final distribution.

Another point that we can see here is the fact that when the number of mixture distributions increases, the convergence to the best likelihood needs more number of iterations which shows that the computational complexity has increased by k amount growth.

All in all, in this problem when the complexity increases, we will move away from the best model.

Appendix

```
#-- Loading Packages and Data
library(caTools)
library(mboost)
library(randomForest)
library(ggplot2)
sp <- read.csv2("spambase.csv")
sp$Spam <- as.factor(sp$Spam)
set.seed(1234567890)
split <- sample.split(Y = sp$Spam, SplitRatio = 2/3)
trainig_set <- subset(x = sp, subset = split == TRUE)
test_set <- subset(x = sp, subset = split == FALSE)
################# Random Forest ###################
ntree <- seq(from = 10, to = 100, by = 10)
output <- data.frame(number_of_trees = 0, Error = 0)
row <- 1
for (i in ntree) {
  set.seed(1234567890)
  classifier <-  randomForest(x = trainig_set[--1]
                              ,y = trainig_set$Spam
                              ,ntree = i)
  Y_hat <- predict(object = classifier, newdata = test_set[--1])
  cm <- table(test_set$Spam, Y_hat)
  Error <- (cm[2,1]+cm[1,2])/sum(cm)
  output[row,] <- c(i,Error)
  row <- row + 1
}

#----- Plot
ggplot() +
  geom_line(data = output, mapping = aes(x = number_of_trees
                                         , y = Error)
            , size = 1.5, color = 'red2') +
  theme_light() +
  scale_x_discrete(limits = seq(from = 10, to = 100, by = 10)) +

  ggtitle('Misclassification rate by random forest algorithim with
          different number of trees') +
  theme(plot.title = element_text(hjust = 0.5))
#---- Best one is when ntree = 50
output[5,2]
################# AdaBoost with decision tree ###################
nIter <- seq(from = 10, to = 100, by = 10)
output <- data.frame(number_of_iterations = 0, Error = 0)
row <- 1
for (i in nIter) {
  set.seed(1234567890)
  classifier <- blackboost(formula = Spam~.
                           , data = trainig_set
                           , family = Binomial(type = c("adaboost"))
                           ,control = boost_control(mstop = i))
  Y_hat <- predict(object = classifier
                   , newdata = test_set
```

```r
                    , type = 'class')

  cm <- table(test_set$Spam, Y_hat)
  Error <- (cm[1,2]+cm[2,1])/sum(cm)
  output[row,] <- c(i,Error)
  row <- row + 1
}
#----- Plot
ggplot() +
  geom_line(data = output, mapping = aes(x = number_of_iterations
                                         , y = Error)
            , size = 1.5, color = 'red2') +
  theme_light() +
  scale_x_discrete(limits = seq(from = 10, to = 100, by = 10)) +

  ggtitle('Misclassification rate by AdaBoost algorithim with
          different number of Iterations') +
  theme(plot.title = element_text(hjust = 0.5))
## 2. MIXTURE MODELS
################# Creating EM_algorithm function ##################
EM_algorithm <- function(k){
set.seed(1234567890)
# max number of EM iterations
max_it <- 100

# min change in log likelihood between two consecutive EM iterations
min_change <- 0.1

#------------- Producing Training data and Initialization ----------------#
# number of training points
N <- 1000

# number of dimensions
D <- 10

# training data
x <- matrix(nrow=N, ncol=D)

# true mixing coefficients
true_pi <- vector(length = k)
true_pi <- rep(1/k, k)

# true conditional distributions
true_mu <- matrix(nrow = k, ncol = D)
if(k == 2){
  plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
  points(true_mu[2,], type="o", col="red")

  true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
  true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
}else if(k == 3){
  plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
  points(true_mu[2,], type="o", col="red")
  points(true_mu[3,], type="o", col="green")

  true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
  true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
  true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
}else {
  plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
  points(true_mu[2,], type="o", col="red")
  points(true_mu[3,], type="o", col="green")
  points(true_mu[4,], type="o", col="yellow")

  true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
  true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
  true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
  true_mu[4,]=c(0.3,0.5,0.5,0.7,0.5,0.5,0.5,0.5,0.4,0.5)}


# Producing the training data
for(n in 1:N) {
  l <- sample(1:k,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[l,d])
  }
}

# fractional component assignments
z <- matrix(nrow = N, ncol = k)

# mixing coefficients
pi <- vector(length = k)

# conditional distributions
mu <- matrix(nrow = k, ncol = D)

# log likelihood of the EM iterations
llik <- vector(length = max_it)

# Random initialization of the paramters
pi <- runif(k,0.49,0.51)
pi <- pi / sum(pi)

for(i in 1:k) {
  mu[i,] <- runif(D,0.49,0.51)
}

#-----------------------  Iteration stage ----------------------#
for(it in 1:max_it) {

  if(k == 2){
    plot(mu[1,], type="o", col="blue", ylim=c(0,1))
    points(mu[2,], type="o", col="red")
  }else if(k == 3){
    plot(mu[1,], type="o", col="blue", ylim=c(0,1))
    points(mu[2,], type="o", col="red")
    points(mu[3,], type="o", col="green")
  }else{
    plot(mu[1,], type="o", col="blue", ylim=c(0,1))
    points(mu[2,], type="o", col="red")
    points(mu[3,], type="o", col="green")
    points(mu[4,], type="o", col="yellow")}


  Sys.sleep(0.5)

# E-step: Computation of the fractional component assignments
# Updating z matrix
p_Xn_MUn <- exp(x %*% log(t(mu)) + (1 - x) %*% log(1 - t(mu)))
numerator <- matrix(rep(pi,N), ncol = k, byrow = TRUE) * p_Xn_MUn
denominator <- rowSums(numerator)
Z_nk <- numerator/denominator

# Updating pi
pi <- colSums(Z_nk)/N

# Updating mu
mu <-  (t(Z_nk) %*% x)/colSums(Z_nk)


#Log likelihood computation.

llik[it] <- sum(Z_nk * ((x %*% log(t(mu)) + (1 - x) %*% log(1 - t(mu))
                         ) + matrix(rep(pi,N), ncol = k, byrow = TRUE)))


cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()

# Stop if the log likelihood has not changed significantly
if(it >= 2){
if((llik[it] - llik[it-1]) < min_change){break()}
}

#M-step: ML parameter estimation from the data and fractional component assignments
# pi_ML
pi_ML <- pi

#mu_ML
mu_ML <- mu

}

#-----------------------  output stage ----------------------#
df <- data.frame(Iteration = 1:length(llik[which(llik != 0.000)])
                 , log_likelihood = llik[which(llik != 0.000)])

require(ggplot2)
```

```
plot <- ggplot(data = df) +
geom_point(mapping = aes(x = Iteration, y = log_likelihood),
           color = 'red2') +
geom_line(mapping = aes(x = Iteration, y = log_likelihood),
             color = 'red2', size = 1) +
ggtitle('Maximum likelihood vs Number of iterations in EM algorithm') +
theme(plot.title = element_text(hjust = 0.5)) +
theme_light()
output <- list(pi_ML = pi_ML,
               mu_ML = mu_ML,
               plot = plot
               )
output
}
EM_2 <- EM_algorithm(2)
EM_2$plot
EM_2$pi_ML
EM_2$mu_ML
EM_3 <- EM_algorithm(3)
EM_3$plot
EM_3$pi_ML
EM_3$mu_ML
EM_4 <- EM_algorithm(4)
EM_4$plot
EM_4$pi_ML
EM_4$mu_ML
```