# Computer lab 1 block 2

*Laura Julià Melis*

*12/04/2019*

## Contents

## Assignment 1. Ensemble Methods.

Before starting, it is necessary to load the `mboost` and `randomForest` packages, import the file to R and then split the dataset into training and hold-out sets. See *Appendix* to see the code used.

### 1.1. Adaboost classification tree.

To fit an Adaboost classification tree the function `blackboost()` from the **mboost** package has ben used. Also, the `family` argument has been used to specify the desired loss function which, in our case, has been the `AdaExp()` because we want the function to perform the AdaBoost algorithm and for that, the exponential loss function is needed. Finally, with the `control` argument, it has been indicated the number of trees to be considered $(10, 20, ..., 100)$.

Once the model has been fitted, predictions and error rates for the hold-out and the training data sets have been calculated.

### 1.2. Random forest model.

In this section, the `randomForest()` from the **randomForest** package has been used to fit the Random forest models, specifying with the `ntrees` argument that the number of trees to grow in each case are $10, 20, ..., 100$.

Then, for each fitted model, predictions and misclassification rates have been calculated.

### 1.3. Performance evaluation.

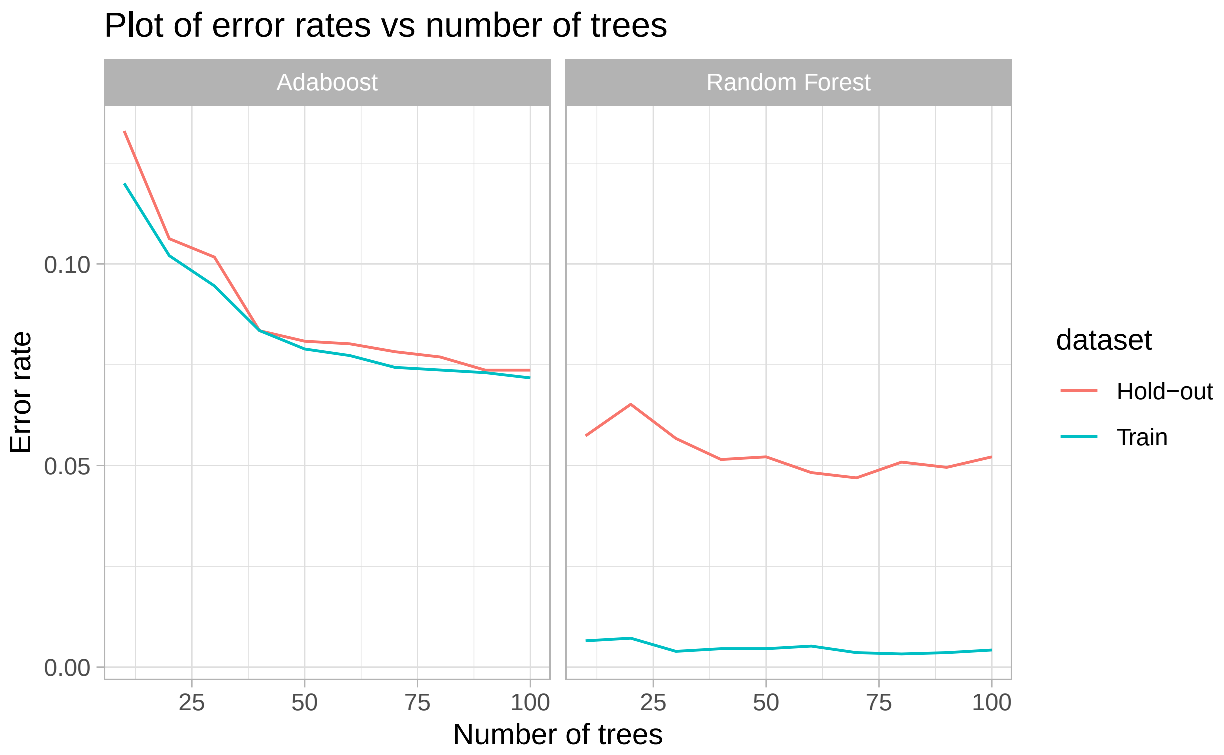
- Error rates for the Adaboost model:

```
##    ntrees rate_ada_holdout rate_ada_train
## 1      10       0.13298566     0.11998696
## 2      20       0.10625815     0.10205412
## 3      30       0.10169492     0.09455494
## 4      40       0.08344198     0.08346919
## 5      50       0.08083442     0.07890447
## 6      60       0.08018253     0.07727421
## 7      70       0.07822686     0.07433975
## 8      80       0.07692308     0.07368764
## 9      90       0.07366362     0.07303554
## 10    100       0.07366362     0.07173133
```

- Error rates for the Random Forest model:

```
##    ntrees rate_randomforest_holdout rate_randomforest_train
## 1      10                0.05736636             0.006521030
## 2      20                0.06518905             0.007173133
## 3      30                0.05671447             0.003912618
## 4      40                0.05149935             0.004564721
## 5      50                0.05215124             0.004564721
## 6      60                0.04823990             0.005216824
## 7      70                0.04693611             0.003586567
## 8      80                0.05084746             0.003260515
## 9      90                0.04954368             0.003586567
## 10    100                0.05215124             0.004238670
```

- Plot showing the error rates by model.



From the plots above we can observe that the Adaboost model has higher error rates than the Random forest model for all the tree numbers considered. Also, the erro rates for the hold-out and the training data sets are more similar in the Adaboost model while in Random Forest the error rate for the training data set is really small compared to the rate for the hold-out data. Finally, we can also comment that the error rate in Adaboost decreases steadily as the number of trees to consider increases and the errors for the Random Forest model remain more or less the same (around 0.05 for the hold-out data), especially at from 40 number of trees.

# Assignment 2. Mixture models.

## 2.1. EM ALGORITHM EXPLANATION.[1]

Let $\boldsymbol{z}$ be a latent variable that denotes from which distribution the sample $\boldsymbol{x} = (x_1, x_2, \cdots, x_D)^T$ is coming from and the probability of observing $x$ given $\mu$ for a multivariate Bernoulli distribution:

$$p(\boldsymbol{x}|\boldsymbol{\mu}) = \prod_{d=1}^{D} \mu_{kd}^{x_d}(1-\mu_{kd})^{(1-x_d)}$$

Then, the mixture model for a multivariate Bernoulli distribution is

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k \cdot (\boldsymbol{x}|\boldsymbol{\mu}_k)$$

where:

- $\pi$ are the mixing coefficients ($0 \leq \pi_k \leq 1$).
- $\mu$ are the Bernoulli parameters indicating probability of success ($0 \leq \mu_k \leq 1$).

And the log-likelihood function for a sample of size N is:

$$\ln p(\boldsymbol{x}|\mu, \pi) = \sum_{n=1}^{N} \ln \left[ \sum_{k=1}^{K} \pi_k \cdot p(\boldsymbol{x}_n|\boldsymbol{\mu}_k) \right]$$

Our goal is to find maximum likelihood estimates for the parameters in the mixture model above, and in order to achieve that, the expectation–maximization (EM) algorithm will be implemented.

The EM Algorithm has different steps:

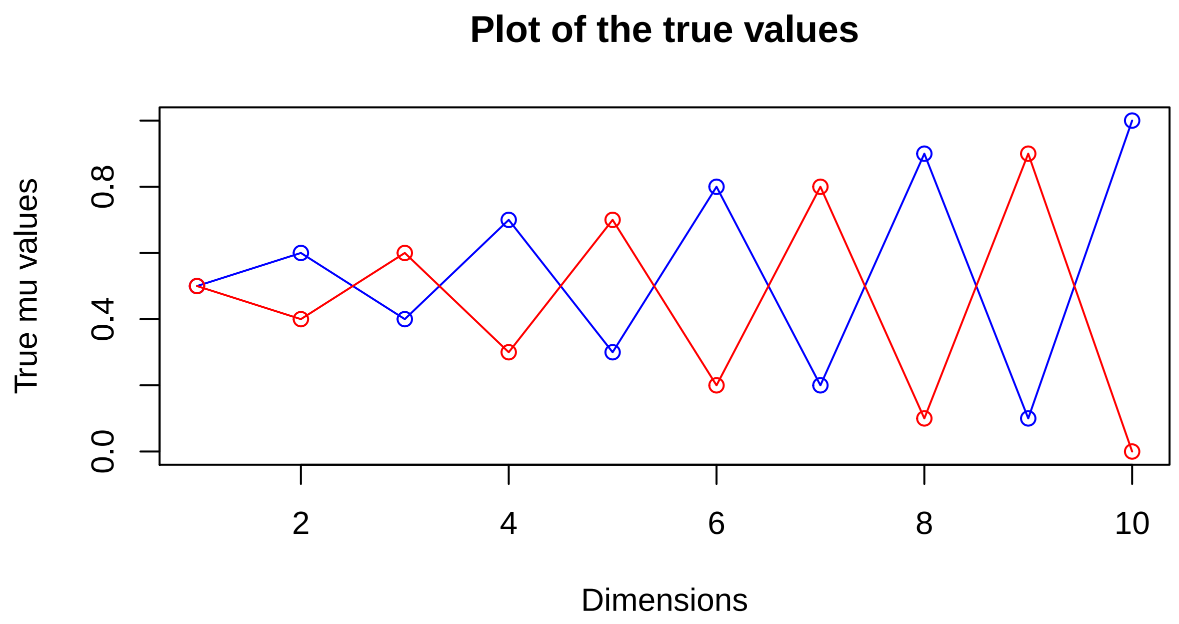
1. Set $\boldsymbol{\pi}$ and $\boldsymbol{\mu}$ to some initial values.
2. *E step*: Compute the posterior distribution $p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})$ for each point.
   2.1. This is calculated using the Bayes's rule: $p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}) = \frac{\pi_k \cdot p(\boldsymbol{x}_n|\boldsymbol{\mu}_k)}{\sum_{k=1}^{K} \pi_k \cdot p(\boldsymbol{x}_n|\boldsymbol{\mu}_k)}$\$.
   2.2. Compute the ML estimation (log-likelihood function shown above).
3. *M step*: Adjust $\pi$ and $\mu$ to fit points assigned to them:
   3.1. Set $\pi_k$ to $\pi_k^{ML} = \frac{\sum_{n=1}^{N} p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{N}$.
   3.2. Set $\mu_{ki}$ to $\mu_k^{ML} = \frac{\sum_{n=1}^{N} x_n \cdot p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_{n=1}^{N} p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}$.
4. Iterate until it converges (repeat until $\boldsymbol{\pi}$ and $\boldsymbol{\mu}$ don't change).

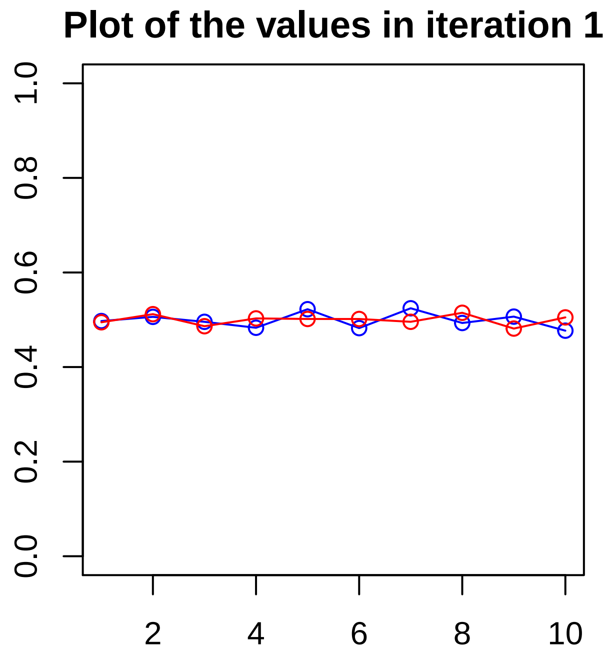
## 2.2. RESULTS FOR K=2 COMPONENTS.

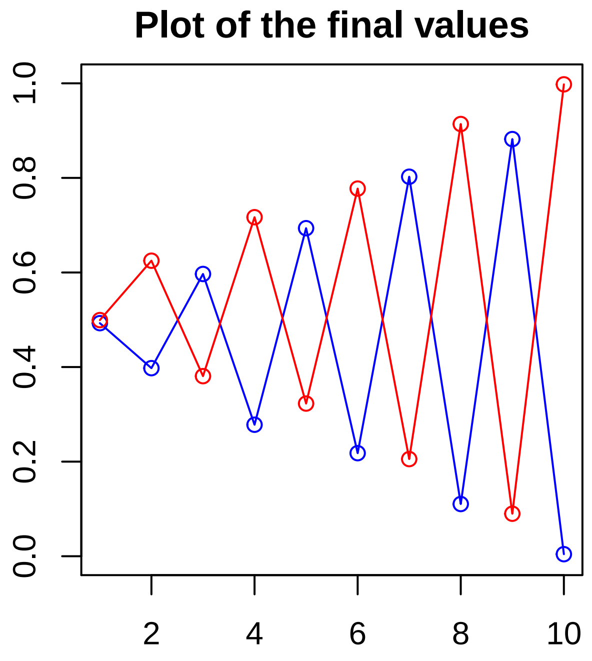
```
## iteration:  1 log likelihood:  -6931.384
## iteration:  2 log likelihood:  -6924.849
## iteration:  3 log likelihood:  -6859.799
## iteration:  4 log likelihood:  -6358.375
## iteration:  5 log likelihood:  -5700.168
## iteration:  6 log likelihood:  -5641.57
## iteration:  7 log likelihood:  -5638.102
```

[1]Source: Chapter 9 "Mixture models and EM" from the book "Pattern Recognition and Machine Learning" of Christopher M. Bishop.

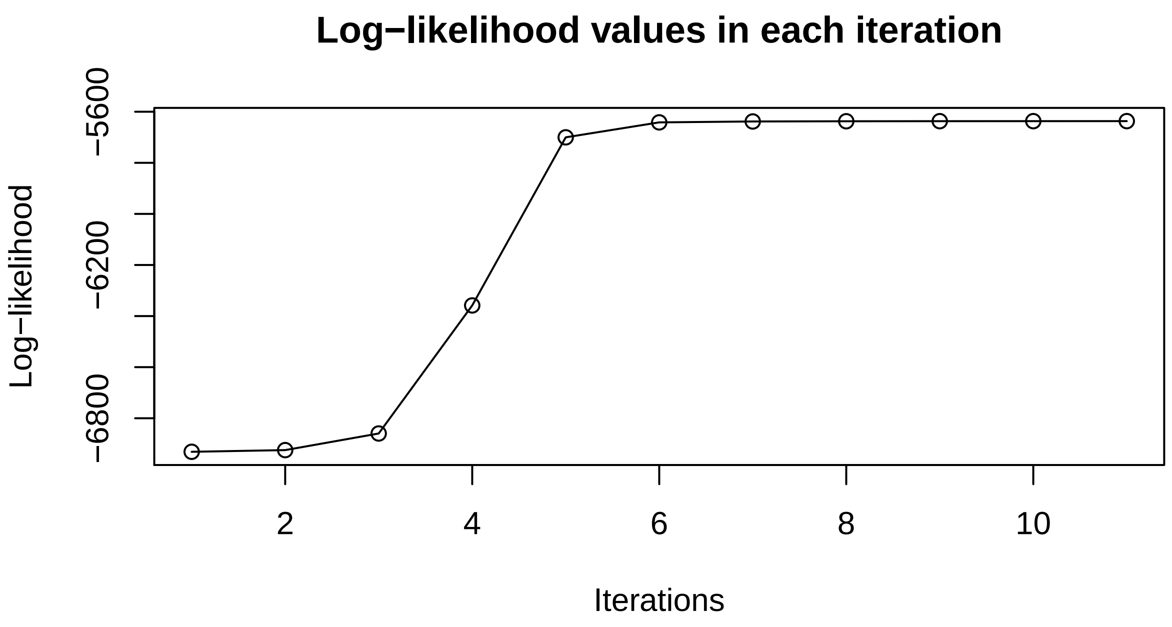
```
## iteration:  8 log likelihood:  -5637.271
## iteration:  9 log likelihood:  -5636.952
## iteration:  10 log likelihood:  -5636.796
## iteration:  11 log likelihood:  -5636.708
```





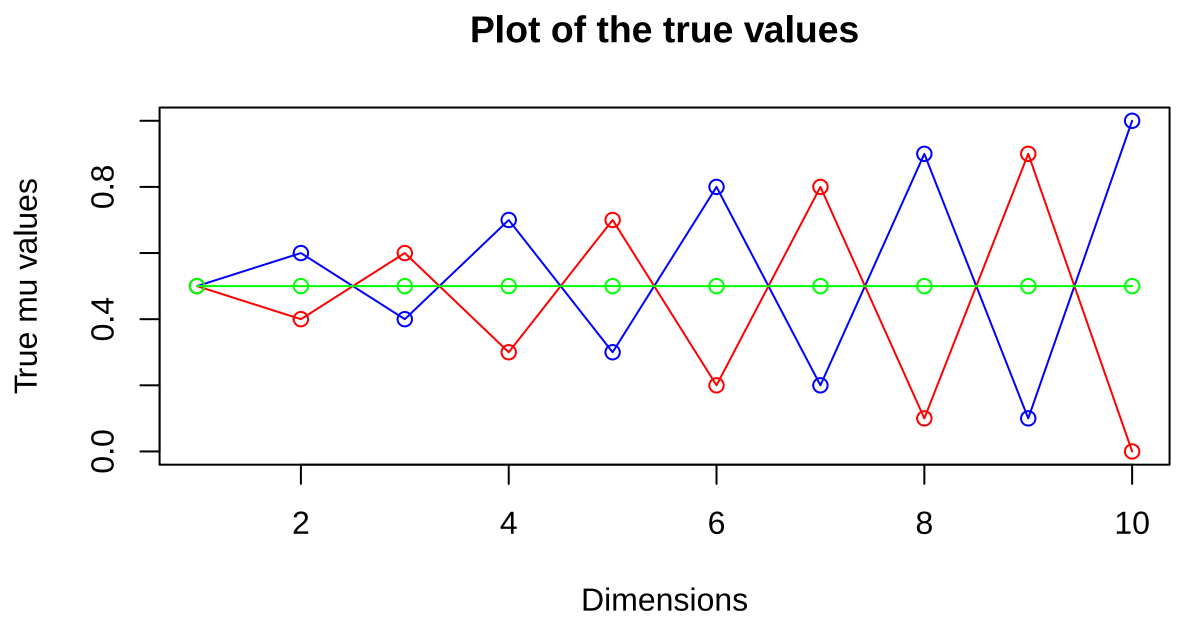


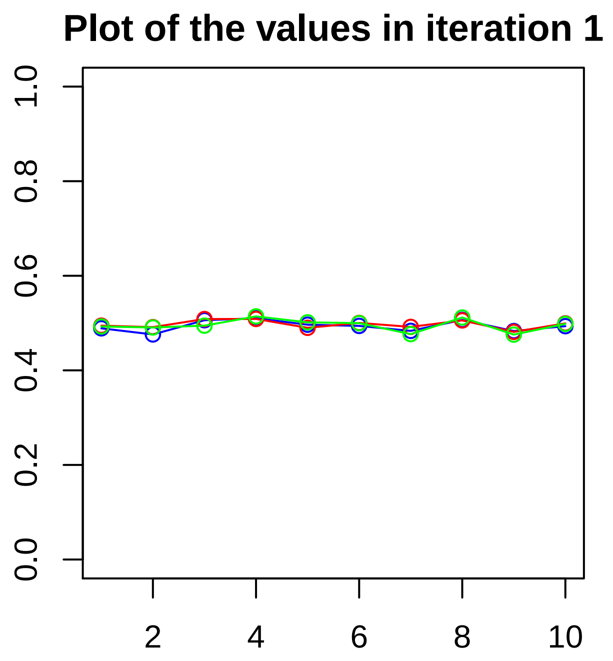
```
## Pi values for K=2:

## [1] 0.5101402 0.4898598

## Mu values for K=2:

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4928148 0.3976832 0.5968198 0.2780429 0.6936406 0.2179508 0.8024894
## [2,] 0.4993170 0.6249254 0.3807992 0.7168564 0.3228394 0.7773950 0.2054015
##           [,8]       [,9]       [,10]
## [1,] 0.1103907 0.88203975 0.004305102
## [2,] 0.9139049 0.08989523 0.997844285
```

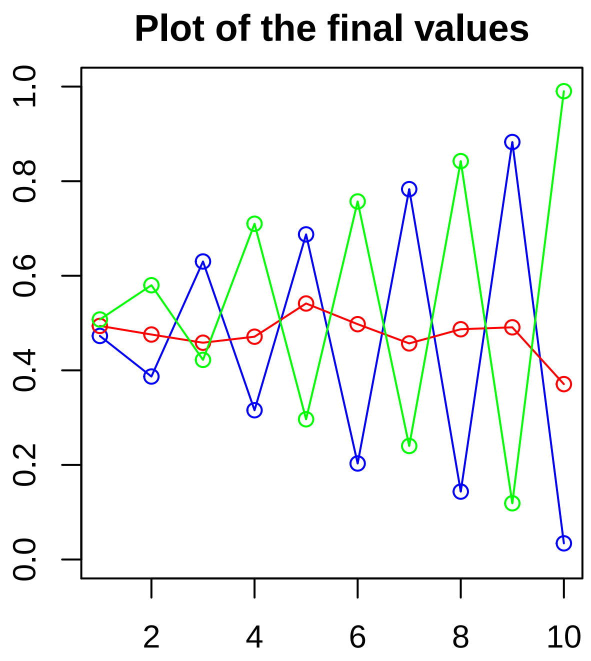


**2.3. RESULTS FOR K=3 COMPONENTS.**

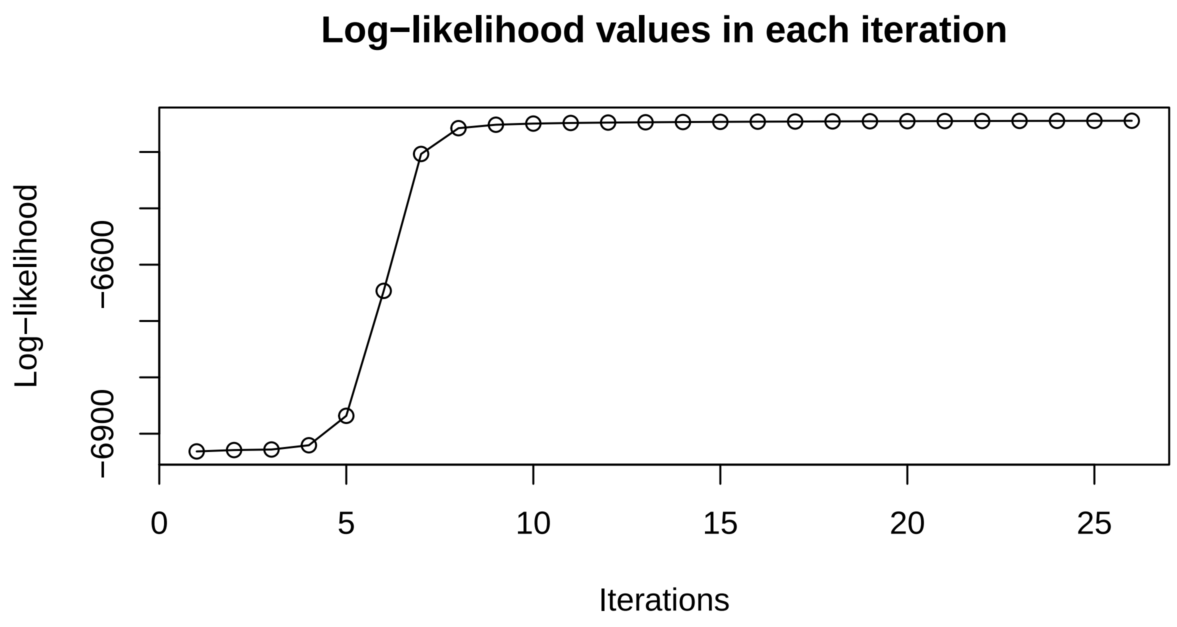
```
## iteration:  1 log likelihood:  -6931.482
## iteration:  2 log likelihood:  -6929.074
## iteration:  3 log likelihood:  -6928.081
## iteration:  4 log likelihood:  -6920.57
## iteration:  5 log likelihood:  -6868.29
## iteration:  6 log likelihood:  -6646.505
## iteration:  7 log likelihood:  -6403.476
## iteration:  8 log likelihood:  -6357.743
## iteration:  9 log likelihood:  -6351.637
## iteration:  10 log likelihood:  -6349.59
## iteration:  11 log likelihood:  -6348.513
## iteration:  12 log likelihood:  -6347.809
## iteration:  13 log likelihood:  -6347.284
## iteration:  14 log likelihood:  -6346.861
## iteration:  15 log likelihood:  -6346.506
## iteration:  16 log likelihood:  -6346.2
## iteration:  17 log likelihood:  -6345.934
## iteration:  18 log likelihood:  -6345.699
## iteration:  19 log likelihood:  -6345.492
## iteration:  20 log likelihood:  -6345.309
## iteration:  21 log likelihood:  -6345.147
## iteration:  22 log likelihood:  -6345.003
## iteration:  23 log likelihood:  -6344.875
## iteration:  24 log likelihood:  -6344.762
## iteration:  25 log likelihood:  -6344.66
## iteration:  26 log likelihood:  -6344.57
```





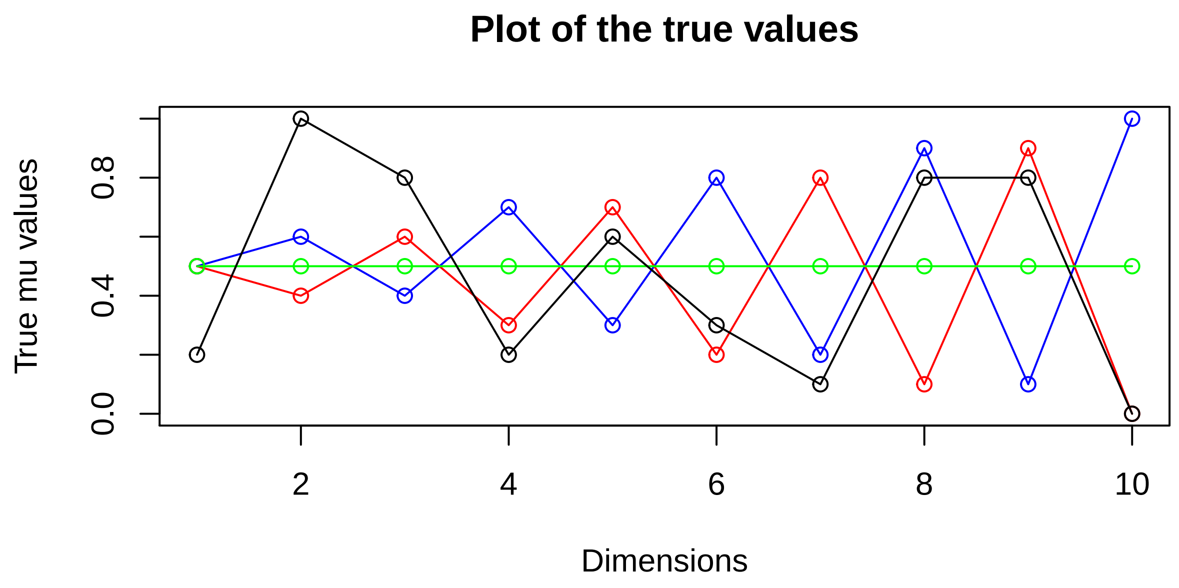


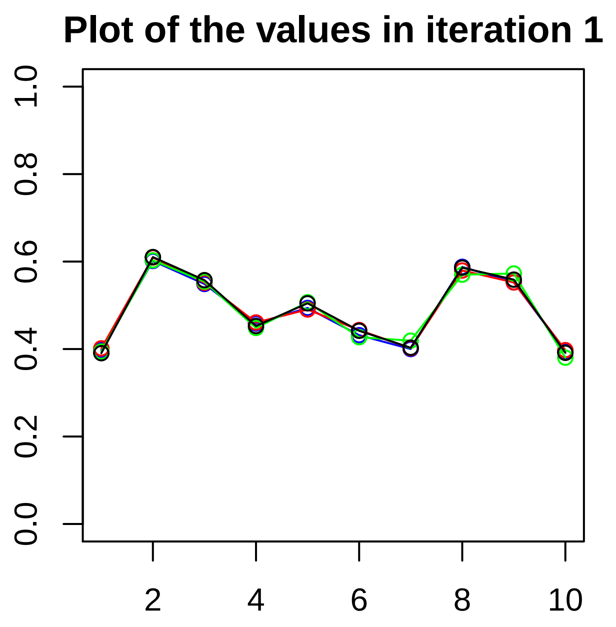
```
## Pi values for K=3:

## [1] 0.3416794 0.2690298 0.3892909

## Mu values for K=3:

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4727544 0.3869396 0.6302224 0.3156325 0.6875038 0.2030173 0.7832090
## [2,] 0.4939501 0.4757687 0.4584644 0.4711358 0.5413928 0.4976325 0.4569664
## [3,] 0.5075441 0.5800156 0.4221148 0.7100227 0.2965478 0.7571593 0.2400675
##           [,8]      [,9]      [,10]
## [1,] 0.1435650 0.8827796 0.03422816
## [2,] 0.4869015 0.4909904 0.37087402
## [3,] 0.8424441 0.1188864 0.99033611
```

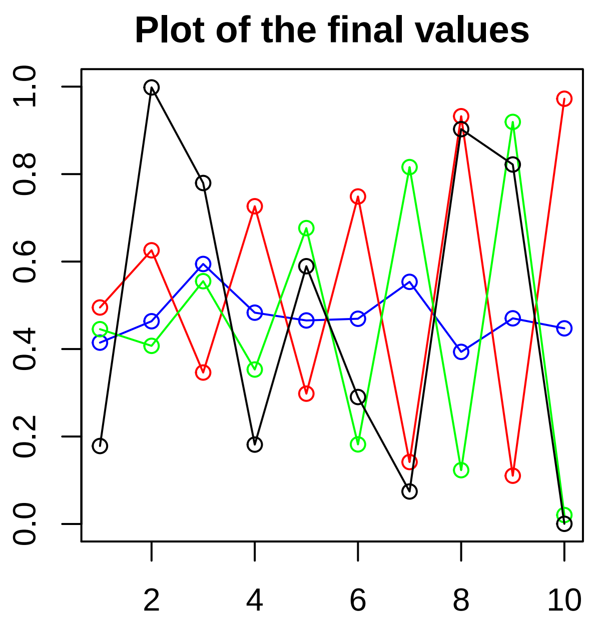


### 2.4. RESULTS FOR K=4 COMPONENTS.

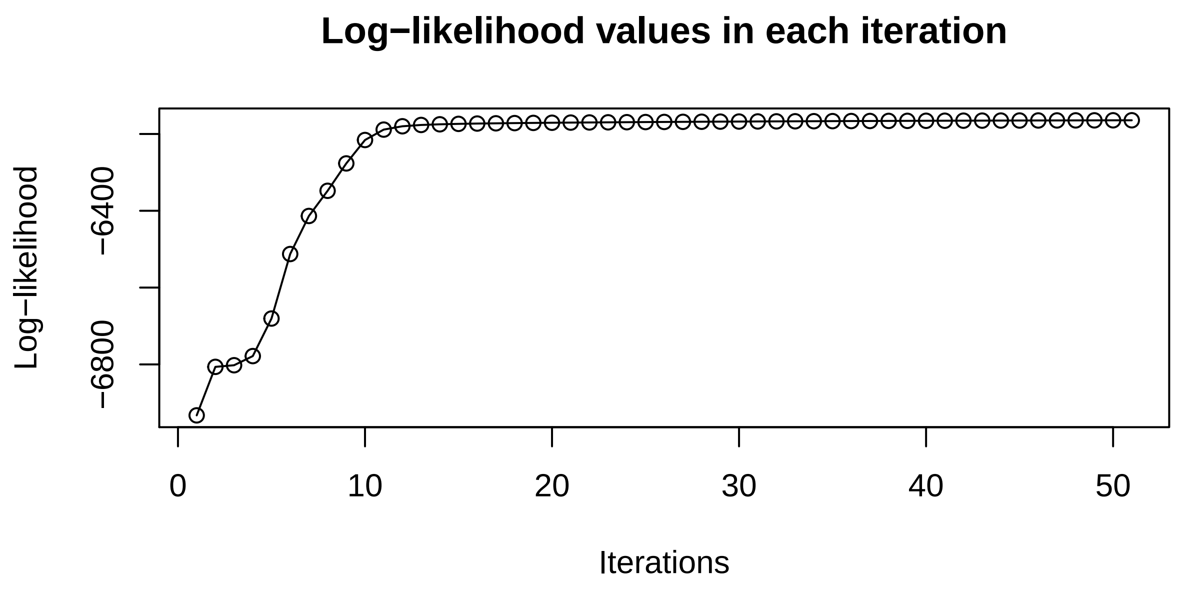
```
## iteration:  1 log likelihood:  -6932.752
## iteration:  2 log likelihood:  -6806.446
## iteration:  3 log likelihood:  -6802.194
## iteration:  4 log likelihood:  -6778.386
## iteration:  5 log likelihood:  -6680.6
## iteration:  6 log likelihood:  -6512.619
## iteration:  7 log likelihood:  -6413.564
## iteration:  8 log likelihood:  -6347.98
## iteration:  9 log likelihood:  -6276.69
## iteration:  10 log likelihood:  -6215.718
## iteration:  11 log likelihood:  -6188.657
## iteration:  12 log likelihood:  -6179.953
## iteration:  13 log likelihood:  -6176.552
## iteration:  14 log likelihood:  -6174.793
## iteration:  15 log likelihood:  -6173.679
## iteration:  16 log likelihood:  -6172.874
## iteration:  17 log likelihood:  -6172.235
## iteration:  18 log likelihood:  -6171.695
## iteration:  19 log likelihood:  -6171.218
## iteration:  20 log likelihood:  -6170.785
## iteration:  21 log likelihood:  -6170.386
## iteration:  22 log likelihood:  -6170.012
## iteration:  23 log likelihood:  -6169.661
## iteration:  24 log likelihood:  -6169.329
## iteration:  25 log likelihood:  -6169.013
## iteration:  26 log likelihood:  -6168.713
## iteration:  27 log likelihood:  -6168.427
## iteration:  28 log likelihood:  -6168.154
## iteration:  29 log likelihood:  -6167.893
## iteration:  30 log likelihood:  -6167.643
## iteration:  31 log likelihood:  -6167.405
## iteration:  32 log likelihood:  -6167.177
## iteration:  33 log likelihood:  -6166.959
## iteration:  34 log likelihood:  -6166.75
## iteration:  35 log likelihood:  -6166.55
## iteration:  36 log likelihood:  -6166.359
## iteration:  37 log likelihood:  -6166.176
## iteration:  38 log likelihood:  -6166
## iteration:  39 log likelihood:  -6165.832
## iteration:  40 log likelihood:  -6165.672
## iteration:  41 log likelihood:  -6165.518
## iteration:  42 log likelihood:  -6165.371
## iteration:  43 log likelihood:  -6165.23
## iteration:  44 log likelihood:  -6165.096
## iteration:  45 log likelihood:  -6164.968
## iteration:  46 log likelihood:  -6164.845
## iteration:  47 log likelihood:  -6164.728
## iteration:  48 log likelihood:  -6164.616
## iteration:  49 log likelihood:  -6164.509
## iteration:  50 log likelihood:  -6164.407
## iteration:  51 log likelihood:  -6164.31
```







```
## Pi values for K=4:

## [1] 0.2502517 0.2814170 0.2599507 0.2083805

## Mu values for K=4:

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.4146163 0.4635748 0.5944980 0.4831440 0.4653533 0.4692804
## [2,] 0.4950136 0.6257836 0.3462818 0.7265150 0.2980083 0.7488649
## [3,] 0.4454147 0.4073089 0.5551794 0.3531776 0.6765153 0.1820397
## [4,] 0.1782815 0.9981893 0.7796168 0.1815423 0.5893990 0.2903202
##            [,7]      [,8]      [,9]         [,10]
## [1,] 0.55365920 0.3935478 0.4702613 0.4473205630
## [2,] 0.14161678 0.9323994 0.1103956 0.9723441021
## [3,] 0.81601012 0.1230776 0.9193068 0.0204963820
## [4,] 0.07423937 0.9028041 0.8219321 0.0004558875
```



### 2.5. ANALYSIS OF RESULTS.

The number of iterations for K= 2,3,4 has been 11, 26 and 54, respectively. Also, comparing the plots of the true and the final $\mu$ values in each case, we can observe that with $K = 2$ both plots are quite similar (almost the same) while when increasing $K$'s, the final results are every time more different than the true values. So we see that when $K = 4$, the $\mu$ estimations obtained with the EM algorithm are not as good as when $K$ is smaller.

# Appendix.

## Assignment 1. Ensemble Methods.

### 1.0. Initialization.

```r
# Changing the version:
RNGversion('3.5.1')

# Loading the packages
library(mboost)
library(randomForest)
library(ggplot2)

# Importing the data:
sp <- read.csv2("spambase.csv")
sp$Spam <- as.factor(sp$Spam)

# Spliting the data in training and hold-out datasets:
n=dim(sp)[1]
set.seed(12345)
id=sample(1:n, floor(n*(2/3)))
train=sp[id,]
holdout=sp[-id,]
```

### 1.1. Adaboost classification tree.

```r
# Number of trees to consider.
ntrees <- seq(10, 100, by = 10)

rate_ada_holdout <- vector()
rate_ada_train <- vector()
for(i in ntrees){
  # Fitting the model with the training dataset
  fit_ada <- blackboost(Spam~., data = train, family = AdaExp(), control=boost_control(mstop = i))
          # "AdaExp() uses the exponential loss, which essentially leads to the AdaBoost algorithm"
          # "mstop = an integer giving the number of initial boosting iterations."

  # Predictions
  fitted_ada_holdout <- predict(fit_ada, newdata = holdout, type = "class")
  fitted_ada_train <- predict(fit_ada, newdata = train, type = "class")

  # Error rates
  rate_ada_holdout[(i/10)] <- mean(fitted_ada_holdout != holdout$Spam)
  rate_ada_train[(i/10)] <- mean(fitted_ada_train != train$Spam)
}
```

### 1.2. Random forest model.

```r
rate_randomforest_holdout <- vector()
rate_randomforest_train <- vector()
for(i in ntrees){
  # Fitting the model with the training dataset
  fit_randomforest <- randomForest(Spam~., data = train, ntree=i) #ntree="Number of trees to grow."

  # Predictions
  fitted_randomforest_holdout <- predict(fit_randomforest, newdata = holdout, type = "class")
  fitted_randomforest_train <- predict(fit_randomforest, newdata = train, type = "class")

  # Error rates
  rate_randomforest_holdout[(i/10)] <- mean(fitted_randomforest_holdout != holdout$Spam)
  rate_randomforest_train[(i/10)] <- mean(fitted_randomforest_train != train$Spam)
}
```

### 1.3. Performance evaluation.

```r
# Table with results.
# Adaboost model:
as.data.frame(cbind(ntrees, rate_ada_holdout, rate_ada_train))

# Random forest model:
as.data.frame(cbind(ntrees, rate_randomforest_holdout, rate_randomforest_train))

# Dataframe to use ggplot()
df <- data.frame(ntrees, error_rates= c(rate_ada_holdout, rate_ada_train,
                                        rate_randomforest_holdout, rate_randomforest_train),
                 dataset  = rep(c("Hold-out", "Train", "Hold-out", "Train"), each=10),
                 model = rep(c("Adaboost", "Adaboost",  "Random Forest", "Random Forest"), each=10))

# Plot of error_rates vs ntrees:
ggplot(df, aes(x = ntrees, y = error_rates, group = dataset, color = dataset)) +
  geom_line() + ggtitle("Plot of error rates vs number of trees") +
  xlab("Number of trees") + ylab("Error rate") +
  theme_light() + facet_grid(cols = vars(model))
```

## Assignment 2. Mixture models.

### FOR K=2.

```r
# ----- K=2 ----- #
### STEP 0. INITIALIZATING THE DATA
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

true_pi <- vector(length = 2) # true mixing coefficients
true_mu <- matrix(nrow=2, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3)
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)
```

```r
# 0.1. Producing the training data
for(n in 1:N) {
  k <- sample(1:2,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

K=2 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

### STEP 1. Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}

for(it in 1:max_it){
  Sys.sleep(0.5)

  ###  STEP 2. E-step: Computation of the fractional component assignments

   # 2.1. We have to compute bayes rule: p(z,x|mu,pi)/ sum(p(z,x|mu,pi))

          prob_x <- exp(x%*%log(t(mu))+(1-x)%*%log(t(1-mu)))
          pi_prob_x <- prob_x * matrix(rep(pi, N), nrow=N, byrow =T)
          sum_pi_prob_x <- rowSums(pi_prob_x)


          z <- pi_prob_x/sum_pi_prob_x


   # 2.2. Log likelihood computation. sum_N ln(sum_K pi * bernoulli)

          llik[it] <- sum(log(sum_pi_prob_x))
          llik

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

    # 2.2.1. Stop if the lok likelihood has not changed significantly
        if(it > 1){
          change <- abs(llik[it]-llik[it-1])
          if(change < min_change){
            break
          }
        }

  ###  STEP 3. M-step: ML parameter estimation from the data and fractional component assignments
```

```
    # 3.1 Setting new mu. mu_ML= sum_kn(x*z)/sum_k(z)
            mu <- (t(z)%*%x)/colSums(z)

    # 3.2. Setting new pi: pi_ML= sum_k(z)/N (of all n)
            pi <- colSums(z)/N

  ### Plot of the initial values:
   if(it==1){
    plot_mu <- mu
    plot_pi <- pi
  }
}

{plot(true_mu[1,], type="o", col="blue", ylim=c(0,1), main="Plot of the true values",
      xlab="Dimensions", ylab="True mu values")
points(true_mu[2,], type="o", col="red")}

par(mfrow=c(1,2), mar=c(3,2,2,1)+0.1)
{plot(plot_mu[1,], type="o", col="blue", ylim=c(0,1),
      main = "Plot of the values in iteration 1", xlab="Dimensions", ylab="mu values")
points(plot_mu[2,], type="o", col="red")}

{plot(mu[1,], type="o", col="blue", ylim=c(0,1),
      main = "Plot of the final values", xlab="Dimensions", ylab="mu values")
points(mu[2,], type="o", col="red")}
cat("Pi values for K=2: ", "\n")
pi
cat("Mu values for K=2: ", "\n")
mu
par(mfrow=c(1,1), mar=c(5, 4, 4, 2) + 0.1)
plot(llik[1:it], type="o", main="Log-likelihood values in each iteration",
     ylab="Log-likelihood", xlab="Iterations")
```

**FOR K=3.**

```
# ----- K=3 ----- #
### STEP 0. INITIALIZATING THE DATA
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3, 1/3)
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)

# 0.1. Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
```

```r
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

### STEP 1. Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}

for(it in 1:max_it){
  Sys.sleep(0.5)

  ###  STEP 2. E-step: Computation of the fractional component assignments

   # 2.1. We have to compute bayes rule: p(z,x|mu,pi)/ sum(p(z,x|mu,pi))

          prob_x <- exp(x%*%log(t(mu))+(1-x)%*%log(t(1-mu)))
          pi_prob_x <- prob_x * matrix(rep(pi, N), nrow=N, byrow =T)
          sum_pi_prob_x <- rowSums(pi_prob_x)


          z <- pi_prob_x/sum_pi_prob_x


   # 2.2. Log likelihood computation. sum_N ln(sum_K pi * bernoulli)

          llik[it] <- sum(log(sum_pi_prob_x))
          llik

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

    # 2.2.1. Stop if the lok likelihood has not changed significantly
        if(it > 1){
          change <- abs(llik[it]-llik[it-1])
          if(change < min_change){
            break
          }
        }

  ###  STEP 3. M-step: ML parameter estimation from the data and fractional component assignments
     # 3.1 Setting new mu. mu_ML= sum_kn(x*z)/sum_k(z)
            mu <- (t(z)%*%x)/colSums(z)
     # 3.2. Setting new pi: pi_ML= sum_k(z)/N (of all n)
             pi <- colSums(z)/N

  ### Plot of the initial values:
   if(it==1){
    plot_mu <- mu
    plot_pi <- pi
  }
}

{plot(true_mu[1,], type="o", col="blue", ylim=c(0,1), main="Plot of the true values",
      xlab="Dimensions", ylab="True mu values")
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")}

par(mfrow=c(1,2), mar=c(3,2,2,1)+0.1)
{plot(plot_mu[1,], type="o", col="blue", ylim=c(0,1),
      main = "Plot of the values in iteration 1", xlab="Dimensions", ylab="mu values")
points(plot_mu[2,], type="o", col="red")
points(plot_mu[3,], type="o", col="green")}

{plot(mu[1,], type="o", col="blue", ylim=c(0,1),
      main = "Plot of the final values", xlab="Dimensions", ylab="mu values")
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")}

cat("Pi values for K=3: ", "\n")
pi
cat("Mu values for K=3: ", "\n")
mu

par(mfrow=c(1,1), mar=c(5, 4, 4, 2) + 0.1)
plot(llik[1:it], type="o", main="Log-likelihood values in each iteration",
     ylab="Log-likelihood", xlab="Iterations")
```

**FOR K=4.**

```
# ----- K=4 ----- #
### STEP 0. INITIALIZATING THE DATA
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

true_pi <- vector(length = 4) # true mixing coefficients
true_mu <- matrix(nrow=4, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3, 1/3, 1/3)
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(0.2,1.0,0.8,0.2,0.6,0.3,0.1,0.8,0.8,0.0)
```

```r
# 0.1. Producing the training data
for(n in 1:N) {
  k <- sample(1:4,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

K=4 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

### STEP 1. Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}

for(it in 1:max_it){
  Sys.sleep(0.5)

  ###  STEP 2. E-step: Computation of the fractional component assignments

   # 2.1. We have to compute bayes rule: p(z,x|mu,pi)/ sum(p(z,x|mu,pi))

          prob_x <- exp(x%*%log(t(mu))+(1-x)%*%log(t(1-mu)))
          pi_prob_x <- prob_x * matrix(rep(pi, N), nrow=N, byrow =T)
          sum_pi_prob_x <- rowSums(pi_prob_x)


          z <- pi_prob_x/sum_pi_prob_x


   # 2.2. Log likelihood computation. sum_N ln(sum_K pi * bernoulli)

          llik[it] <- sum(log(sum_pi_prob_x))
          llik

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

    # 2.2.1. Stop if the lok likelihood has not changed significantly
        if(it > 1){
          change <- abs(llik[it]-llik[it-1])
          if(change < min_change){
            break
          }
        }

  ###  STEP 3. M-step: ML parameter estimation from the data and fractional component assignments
```

```r
     # 3.1 Setting new mu. mu_ML= sum_kn(x*z)/sum_k(z)

            mu <- (t(z)%*%x)/colSums(z)
     # 3.2. Setting new pi: pi_ML= sum_k(z)/N (of all n)

            pi <- colSums(z)/N

  ### Plot of the initial values:
   if(it==1){
    plot_mu <- mu
    plot_pi <- pi
  }
}

{plot(true_mu[1,], type="o", col="blue", ylim=c(0,1), main="Plot of the true values",
      xlab="Dimensions", ylab="True mu values")
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
points(true_mu[4,], type="o", col="black")}

par(mfrow=c(1,2), mar=c(3,2,2,1)+0.1)
{plot(plot_mu[1,], type="o", col="blue", ylim=c(0,1),
      main = "Plot of the values in iteration 1", xlab="Dimensions", ylab="mu values")
points(plot_mu[2,], type="o", col="red")
points(plot_mu[3,], type="o", col="green")
points(plot_mu[4,], type="o", col="black")}

{plot(mu[1,], type="o", col="blue", ylim=c(0,1),
      main = "Plot of the final values", xlab="Dimensions", ylab="mu values")
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
points(mu[4,], type="o", col="black")}

cat("Pi values for K=4: ", "\n")
pi
cat("Mu values for K=4: ", "\n")
mu

par(mfrow=c(1,1), mar=c(5, 4, 4, 2) + 0.1)
plot(llik[1:it], type="o", main="Log-likelihood values in each iteration",
     ylab="Log-likelihood", xlab="Iterations")
```