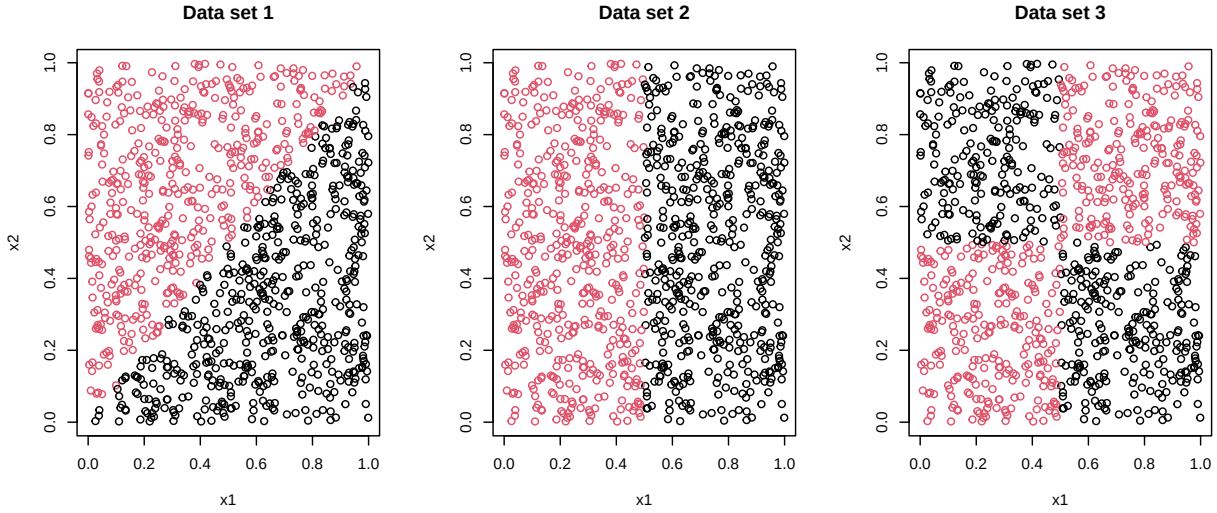
Computer Lab 1 block 2

Alan Cacique Tamariz (alaca734), Tugce Izci (tugiz798) and Kerstin Nilsson (kerni714)

2022-12-08

1. Ensemble methods

For this Ensemble methods assignment, 3 test datasets of dimension 1000 x 3 were created, as it can be seen below, each plots shows the two feature variables $X1$ and $X2$ with different divisions of the output variable Y. The objective of this task is to classify Y from $X1$ and $X2$ by fitting multiple random forests classifier models with 1000 training datasets of dimension 100 x 3 and compute the mean and variance of the misclassification rate of the test set. This is performed for different number of trees; 1,10 and 100 trees. The node size was set to 25 for the two first datasets and to 12 for the third one.



In ensemble methods like random forest, the idea is to train each base model (tree) in a slightly different way so as to avoid overfitting, and then that each base model will contribute to the learning. In random forests, the difference between the models is obtained by bagging and by decorrelating the prediction of the base models through a randomised selection of features to the individual trees. When the prediction result is averaged over the contributing models, it will result in decreased variance in the prediction, and which reduces the risk of overfitting. Thus, if more trees are used, more trees are contributing to learning the relationship between the input variables and the output, while at the same time averaging over several models (constructed by bagging and random variable selection) will reduce the risk of overfitting, and hence lead to lower error rate.

For the third dataset, we use nodesize=12 instead of nodesize=25. This parameter sets the minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown. Thus, larger trees will be grown with nodesize=12, which could be one explanation as to why the performance is better when using

sufficient trees in the random forest. It could also indicate that, given a sufficient number of trees, it is easier for decision trees to learn the type of classification problem that is represented as dataset 3, than the type as represented as data set 1. It is likely due to that in dataset 3, the decision boundaries correspond to a split along each of $X1$ and $X2$, rather than a combination of the two as in dataset 1. It is not immediately obvious how the tree finds those splits as $X1$ and $X2$ have uniform distributions, however we think the explanation could be that if the smallest allowed node size is sufficiently small, it is likely that it is possible to find an initial split along one of the variables due to random variability that can satisfy the splitting criterion. Having made this first split along one of the variables would then make the subsequent splits easier. This splitting can then continue until the splitting criteria stops the growing of the tree.

	1st Data set		2nd Data set		3rd Data set	
	mean	variance	mean	variance	mean	variance
1 Tree	21.0	34.1	9.3	172.9	25.1	132.6
10 Trees	13.5	9.6	1.4	4.6	12.3	32.0
100 trees	11.1	8.6	0.6	0.7	7.7	14.1

2. Mixure models

The EM algorithm was implemented for the Bernoulli mixture model. Note that the R code was adapted slightly from the provided template to produce output that better aligned with the R markdown format (though it produces the same results as using the provided template from the beginning for each value of M).

The Bernoulli mixture model is:

$$p(\boldsymbol{x}) = \sum_{m=1}^{M} \pi_m Bern(\boldsymbol{x}|\boldsymbol{\mu}_m)$$

where $\boldsymbol{x} = \{x_1, x_2, ..., x_D\}$ is a D-dimensional binary random vector, $\pi_m = p(y = m)$ and

$$Bern(\boldsymbol{x}|\boldsymbol{\mu}_m) = p(\boldsymbol{x}|y = m) = \prod_{d=1}^{D} \mu_{m,d}^{x_d}(1 - \mu_{m,d})^{(1-x_d)}$$

where $\boldsymbol{\mu} = (\mu_{m,1}, ..., \mu_{m,D})$ is a D-dimensional vector of probabilities.

The log likelihood of the dataset $\{\boldsymbol{x}_i^n\}$ is:

$$\sum_{i=1}^{n} \log p(\boldsymbol{x}_i)$$

The weights in the EM algorithm are calculated as $p(y = m|\boldsymbol{x}_i, \hat{\boldsymbol{\mu}})$

We have that $p(\boldsymbol{x}, y) = p(\boldsymbol{x}|y)p(y)$ and that $p(\boldsymbol{x}, y) = p(y|\boldsymbol{x})p(x)$, thus

$$p(y|\boldsymbol{x}) = \frac{p(y)p(\boldsymbol{x}|y)}{p(x)}$$

The weights then become:

$$p(y = m|\boldsymbol{x}_i) = \frac{p(y = m)p(\boldsymbol{x}|y = m)}{p(\boldsymbol{x})} = \frac{\hat{\pi}_m Bern(\boldsymbol{x_i}|\hat{\boldsymbol{\mu}}_m)}{\sum_{m=1}^{M} \hat{\pi}_m Bern(\boldsymbol{x_i}|\hat{\boldsymbol{\mu}}_m)}$$

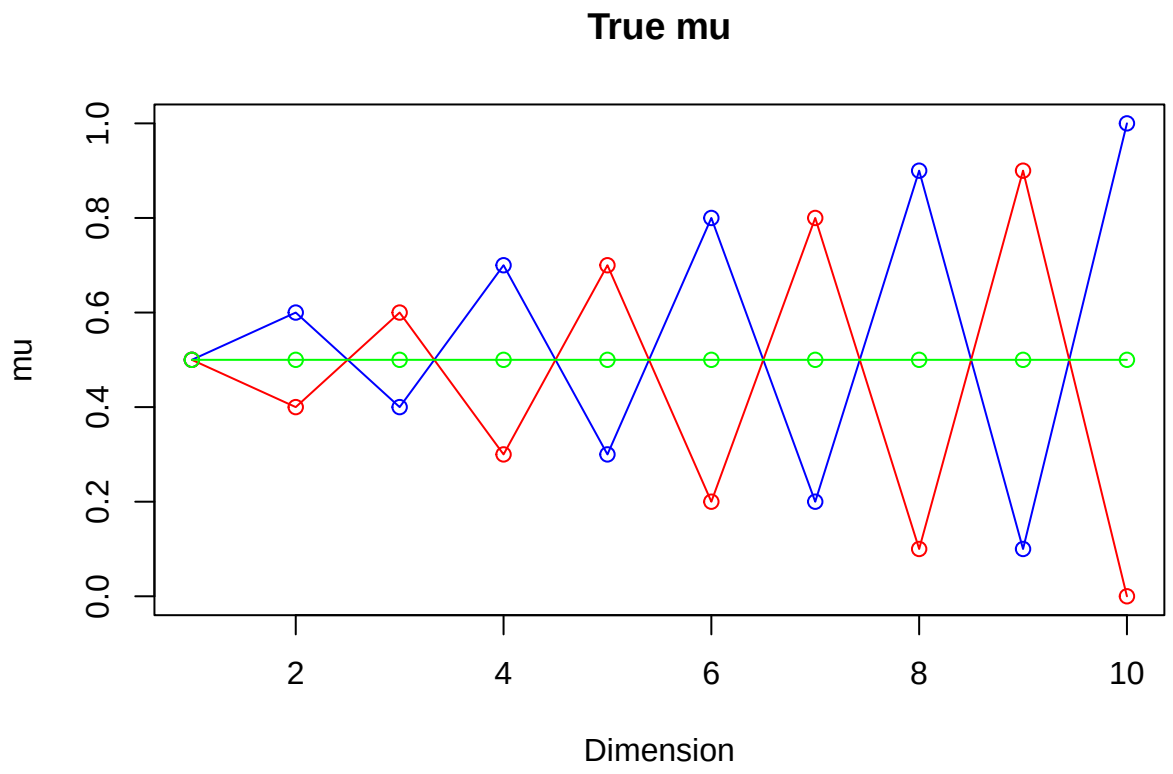
Finally, the parameters are updated according to:

$$\hat{\pi}_m = \frac{1}{n} \sum_{i=1}^{n} w_i(m),$$

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{m}} = \frac{1}{\sum_{i=1}^{n} w_i(m)} \sum_{i=1}^{n} w_i(m)\boldsymbol{x}_i$$

The true parameter values are displayed below.

True μ:

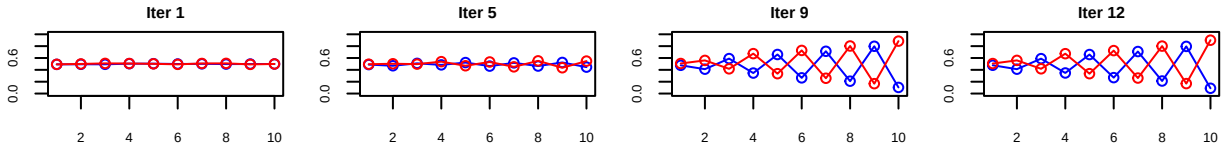


True π:

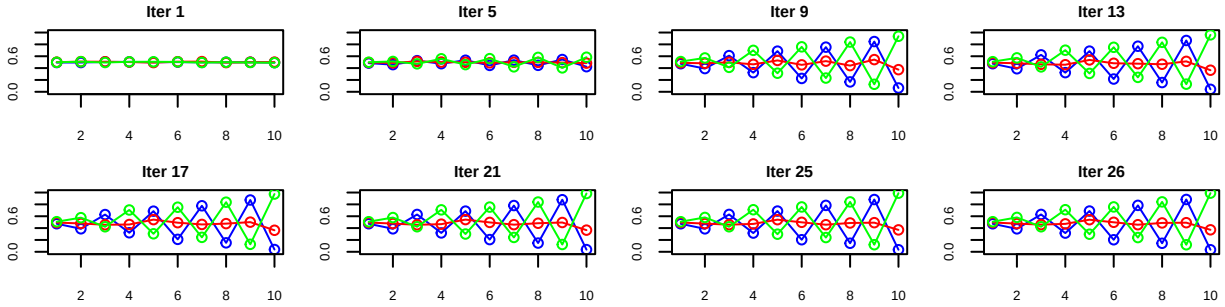
```
#> [1] 0.3333 0.3333 0.3333
```

The EM algorithm was run for $M = 2, 3$ and 4. Below, estimates of μ are displayed for the first, then every fourth iteration as well as after the final iteration. For overview and completeness, the final μ are also displayed after the iteration plots, along with the estimated values for π.

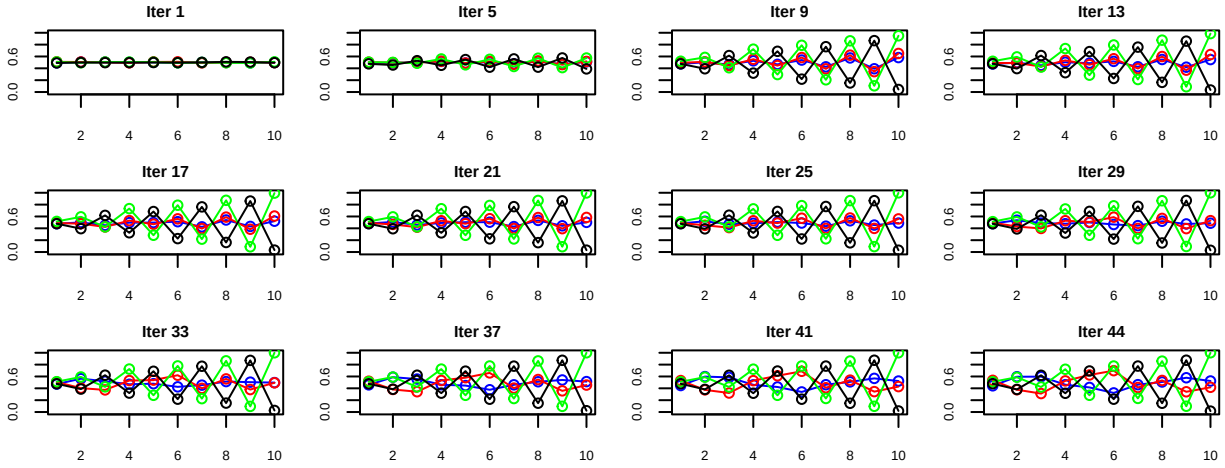
```
#> M = 2, start values of pi (rounded):0.49921, 0.50079
```



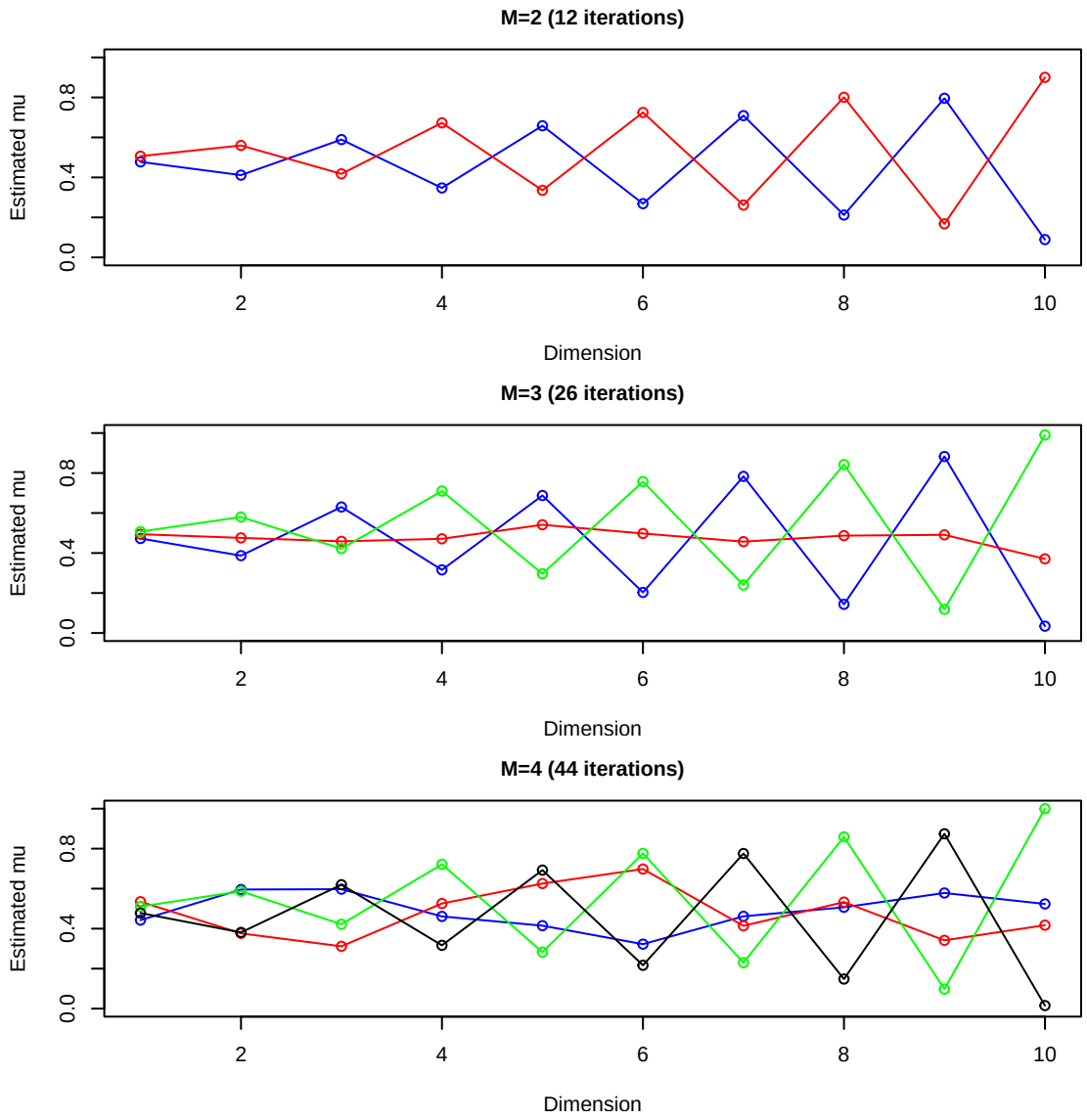

```
#> M = 3, start values of pi (rounded):0.33261, 0.33366, 0.33374
```




```
#> M = 4, start values of pi (rounded):0.24918, 0.24997, 0.25003, 0.25082
```



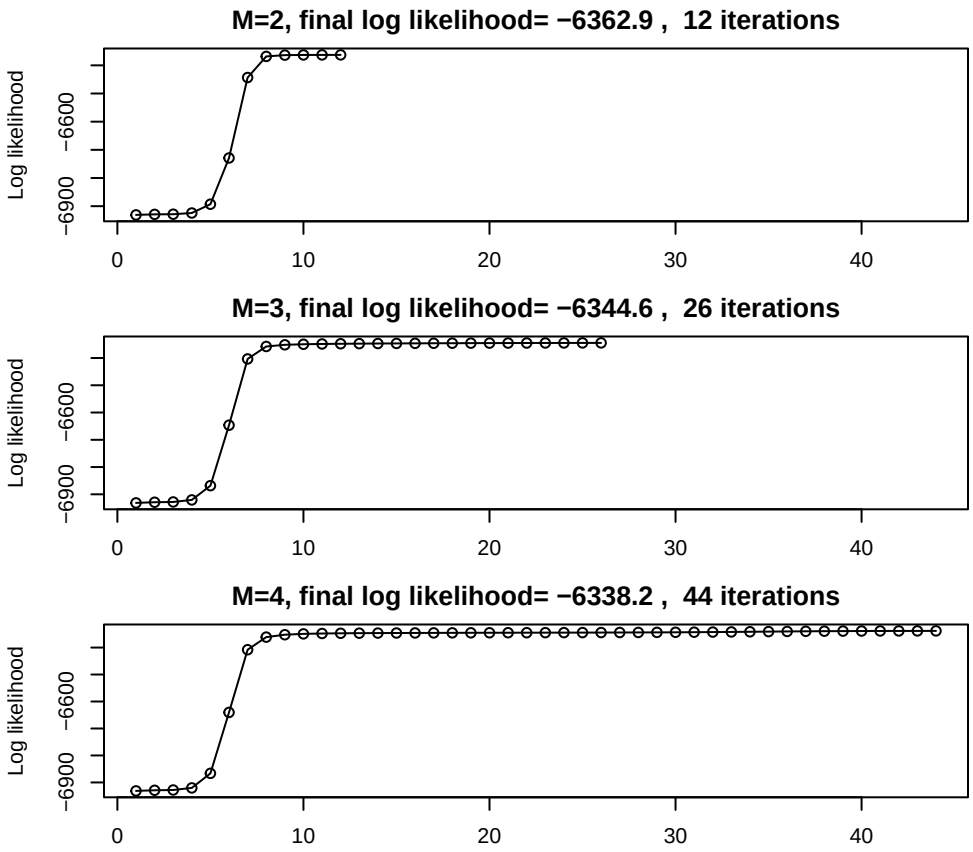
The values of the estimated μ and π are displayed below:



Estimated π:

```
#> Estimated pi M=2:

#> 0.4971 0.5029

#> Estimated pi M=3:

#> 0.3417 0.269 0.3893

#> Estimated pi M=4:

#> 0.1547 0.1419 0.3514 0.352
```

The log likelihood was also plotted versus the iteration number:



It can be seen that the number of iterations of the EM algorithm increased with increasing M, and that the maximum log likelihood was obtained for $M = 4$, even though the true $M = 3$. This can be explained by that the higher the number of clusters (M), the better fit can be obtained to the data.

Statement of Contribution

Assignment 1 was contributed by Alan Cacique and Tugce Izci. Assignment 2 was contributed by Kerstin Nilsson. Both assignments procedures and results were reviewed and discussed before the creation of the final report.

Appendix: All code for this report

```
knitr::opts_chunk$set(comment = "#>",
                      echo = FALSE)
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
#- Assignment 1
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#

library(randomForest)
library(dplyr)
library(ggplot2)
library(knitr)
library(kableExtra)
#-------------------------------------------------------------------------------#
# Functions
#-------------------------------------------------------------------------------#

generate_df1<- function(n){
x1<-runif(n)
x2<-runif(n)
tedata<-cbind(x1,x2)
y111<-as.numeric(x1<x2)
y<-as.factor(y111)
df1<-data.frame(tedata,y)
return(df1)
}

generate_df2 <- function(n){
  x1<-runif(n)
x2<-runif(n)
tedata<-cbind(x1,x2)
y111<-as.numeric(x1<0.5)
y<-as.factor(y111)
df2<-data.frame(tedata,y)
return(df2)
}

generate_df3 <- function(n){
x1<-runif(n)
x2<-runif(n)
tedata<-cbind(x1,x2)
y111<-as.numeric((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5))
y<-as.factor(y111)
df3<-data.frame(tedata,y)
return(df3)
}
#-------------------------------------------------------------------------------#
# Main program
#-------------------------------------------------------------------------------#

set.seed(1234)
```

```r
df_test1<-generate_df1(1000)
## Matrix to store the Misclassification values
MR_rate <- matrix(0,nrow = 1000, ncol = 3)
colnames(MR_rate)<- c("MR 1 tree","MR 10 tree","MR 100 tree")
  # Misclasification values Data set 2
MR_rate2 <- matrix(0,nrow = 1000, ncol = 3)
colnames(MR_rate2)<- c("MR 1 tree","MR 10 tree","MR 100 tree")
#   # Misclasification values Data set 3
MR_rate3 <- matrix(0,nrow = 1000, ncol = 3)
colnames(MR_rate3)<- c("MR 1 tree","MR 10 tree","MR 100 tree")
n_trees<-c(1,10,100)
for (i in 1:1000) {

  df_train1<-generate_df1(100)
  w<-1
  for (j in n_trees) {
    rf <- randomForest(y~x1+x2, data = df_train1, ntree=j ,nodesize = 25, keep.forest = TRUE )
    predicted_values <-predict(rf,df_test1)
    MR_rate[i,w] <- (mean(df_test1[,3] != predicted_values))*100
  w<-w+1
    }
}
set.seed(1234)
df_test2<-generate_df2(1000)
for (i2 in 1:1000) {

  df_train2<-generate_df2(100)
  w2<-1
  for (j2 in n_trees) {
    rf2 <- randomForest(y~x1+x2, data = df_train2, ntree=j2 ,nodesize = 25, keep.forest = TRUE )
    predicted_values2 <-predict(rf2,df_test2)
    MR_rate2[i2,w2] <- (mean(df_test2[,3] != predicted_values2))*100
  w2<-w2+1
    }
}
set.seed(1234)
df_test3<-generate_df3(1000)
for (i3 in 1:1000) {

  df_train3<-generate_df3(100)
  w3<-1
  for (j3 in n_trees) {
    rf3 <- randomForest(y~x1+x2, data = df_train3, ntree=j3 ,nodesize = 12, keep.forest = TRUE )
    predicted_values3 <-predict(rf3,df_test3)
    MR_rate3[i3,w3] <- (mean(df_test3[,3] != predicted_values3))*100
  w3<-w3+1
    }
}

par(mfrow=c(1,3))
plot(df_test1$x1,df_test1$x2,col=((df_test1$y)), xlab = "x1", ylab = "x2", main = "Data set 1")
plot(df_test2$x1,df_test2$x2,col=((df_test2$y)), xlab = "x1", ylab = "x2", main = "Data set 2")
plot(df_test3$x1,df_test3$x2,col=((df_test3$y)), xlab = "x1", ylab = "x2", main = "Data set 3")
```

```r
## Create a matrix to save the mean and the variance for 1, 10 and 100 trees
options(digits = 4)
miu_sigma <- matrix(0, nrow = 3, ncol = 6)
colnames(miu_sigma)<- c("mean","variance","mean","variance", "mean","variance")
rownames(miu_sigma)<- c("1   Tree", "10  Trees", "100 trees")

## Calculate mean and variances
miu_sigma[1,1]<- mean(MR_rate[,1])
miu_sigma[1,2]<- var(MR_rate[,1])
miu_sigma[2,1]<- mean(MR_rate[,2])
miu_sigma[2,2]<- var(MR_rate[,2])
miu_sigma[3,1]<- mean(MR_rate[,3])
miu_sigma[3,2]<- var(MR_rate[,3])
  # Test data 2
miu_sigma[1,3]<- mean(MR_rate2[,1])
miu_sigma[1,4]<- var(MR_rate2[,1])
miu_sigma[2,3]<- mean(MR_rate2[,2])
miu_sigma[2,4]<- var(MR_rate2[,2])
miu_sigma[3,3]<- mean(MR_rate2[,3])
miu_sigma[3,4]<- var(MR_rate2[,3])
  # Test data 3
miu_sigma[1,5]<- mean(MR_rate3[,1])
miu_sigma[1,6]<- var(MR_rate3[,1])
miu_sigma[2,5]<- mean(MR_rate3[,2])
miu_sigma[2,6]<- var(MR_rate3[,2])
miu_sigma[3,5]<- mean(MR_rate3[,3])
miu_sigma[3,6]<- var(MR_rate3[,3])

miu_sigma<-round(miu_sigma, digits = 1)

kable(miu_sigma)%>%
  kableExtra::add_header_above(c(" "=1,"1st Data set"=2, "2nd Data set"=2, "3rd Data set"=2))

#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
#- Assignment 2
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#

#-------------------------------------------------------------------------------#
# Functions
#-------------------------------------------------------------------------------#

#- Function to calculate p(x|y=m) for given class m, with param mu,
# Bernoulli distribution
#
# mu is D-dimensional vector
# x is matrix of dimension nxD
bern <- function(x, mu) {
  D <- dim(x)[2]
  stopifnot(length(mu)==D)
  n <- dim(x)[1]
  res <- numeric(n)
```

```
  for (i in 1:n) {
    prod_d <- 1
    for (d in 1:D) {
      term_d <- (mu[d]^x[i,d])*((1-mu[d])^(1-x[i,d]))
      prod_d <- term_d*prod_d
    }
    res[i] <- prod_d
  }
  return(res)
}

#- Function to calculate p(x)
# mu_v - vector of mu
# pi_v -vector with pi
px <- function(x, mu_v, pi_v) {
  M <- dim(mu_v)[1]
  sum_px <- 0
  for (m in 1:M) {
    sum_px <- sum_px + (bern(x, mu_v[m,])*pi_v[m])
  }
  return(sum_px)
}
#-------------------------------------------------------------------------------#
# Main program
#-------------------------------------------------------------------------------#

#set.seed(1234567890)

max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log lik between two consecutive 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)
#plot(true_mu[1,], type="o", col="blue", ylim=c(0,1), xlab = "Dimension",
#     ylab=expression(mu), main=expression(paste("True ",mu)))

#- The part with expression(paste("True ",mu))) does not work due to
# using the kableExtra library in assignment 1
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1), xlab = "Dimension",
     ylab="mu", main="True mu")

points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")

#- Note: this is done in loop to obtain same seed as would have been obtained
#  when running the original script separately for each M
```

```
# #- Producing the training data
# for(i in 1:n) {
#   m <- sample(1:3,1,prob=true_pi)
#   for(d in 1:D) {
#     x[i,d] <- rbinom(1,1,true_mu[m,d])
#   }
# }

true_pi

#M=3 # number of clusters
llik <- matrix(nrow=max_it, ncol=3) # log likelihood of the EM iterations
mu_list <- list() # save final mu for each M
pi_list <- list() # save final pi for each M
iter_list <- list() # save number of iter for each M

for (M in 2:4) {

  cat(paste0("M = ",M))
  set.seed(1234567890)
  #- Producing the training data
  for(i in 1:n) {
    m <- sample(1:3,1,prob=true_pi)
    for(d in 1:D) {
      x[i,d] <- rbinom(1,1,true_mu[m,d])
    }
  }

  w <- matrix(nrow=n, ncol=M) # weights
  pi <- vector(length = M) # mixing coefficients
  mu <- matrix(nrow=M, ncol=D) # conditional distributions
  #llik <- vector(length = max_it) # log likelihood of the EM iterations

  #- Random initialization of the parameters
  pi <- runif(M,0.49,0.51)
  pi <- pi / sum(pi)
  for(m in 1:M) {
    mu[m,] <- runif(D,0.49,0.51)
  }
  #pi
  #mu
  pi_str <- toString(round(pi,5))
  cat(paste0(", start values of pi (rounded):", pi_str))
  par(mfrow=c(3,4),mai = c(0.3, 0.25, 0.2, 0.15))
  for(it in 1:max_it) {
    #- Plot 1st then each 4th iter

    if(it==1 | ((it-1)%%4==0)) {
      plot(mu[1,], type="o", col="blue", ylim=c(0,1), main=paste0("Iter ", it),
           cex.main=0.8, cex.axis=0.6)
      points(mu[2,], type="o", col="red")
      if (dim(mu)[1] >= 3) {
        points(mu[3,], type="o", col="green")
```

```
    }
    if (dim(mu)[1] == 4) {
      points(mu[4,], type="o", col="black")
    }
  }
  Sys.sleep(0.5)

  #- E-step: Computation of the weights
  ############################################################################
  #- Your code here
  sum_px <- px(x=x, mu_v=mu, pi_v=pi)
  for (m in 1:M) {
    w[,m] <- (pi[m]*bern(x, mu[m,]))/sum_px
  }
  ############################################################################

  #- Log likelihood computation.
  ############################################################################
  #- Your code here
  llik[it,M-1] <- sum(log(sum_px))

  ############################################################################

  #cat("iteration: ", it, "log likelihood: ", llik[it,M-1], "\n")
  #flush.console()

  #-  Stop if the lok likelihood has not changed significantly
  ############################################################################
  #- Your code here
  # The value of the observed data log-likelihood increases at each iteration of
  # the procedure, unless it has reached a stationary point
  if (it > 1) {
    llik_ch <- llik[it,M-1]-llik[it-1,M-1]
    if (llik_ch < min_change) {
      break
    }
  }
  ############################################################################

  #- M-step: ML parameter estimation from the data and weights
  ############################################################################
  #- Your code here
  for (m in 1:M) {
    pi[m] <- (1/n)*sum(w[,m])
    mu[m,] <- (1/sum(w[,m]))*colSums(w[,m]*x)
  }
  ############################################################################
}

# - Plot likelihood vs iterations
#plot(llik[1:it,M-1], type="o")

#- Plot last iteration
```

```
  plot(mu[1,], type="o", col="blue", ylim=c(0,1), main=paste0("Iter ", it),
       cex.main=0.8, cex.axis=0.6)
  points(mu[2,], type="o", col="red")
  if (dim(mu)[1] >= 3) {
    points(mu[3,], type="o", col="green")
  }
  if (dim(mu)[1] == 4) {
    points(mu[4,], type="o", col="black")
  }

  #pi
  #mu

  mu_list[[M-1]] <- mu
  pi_list[[M-1]] <- pi
  iter_list[[M-1]] <- it
}

# - Plot final parameters
par(mfrow=c(3,1), mai=c(0.5,0.8,0.35,0.5))
for (M in 2:4) {
  mu <- mu_list[[M-1]]
  it <- iter_list[[M-1]]
  # expression does not work due to kableExtra
  #plot(mu[1,], type="o", col="blue", ylim=c(0,1),xlab="Dimension",
  #     ylab=expression(paste("Estimated ",mu)),
  #     main=paste0("M=", M, " (",  it, " iterations)"), cex.main=1)

  plot(mu[1,], type="o", col="blue", ylim=c(0,1),xlab="Dimension",
       ylab="Estimated mu",
       main=paste0("M=", M, " (",  it, " iterations)"), cex.main=1)

  points(mu[2,], type="o", col="red")
  if (dim(mu)[1] >= 3) {
    points(mu[3,], type="o", col="green")
  }
  if (dim(mu)[1] == 4) {
    points(mu[4,], type="o", col="black")
  }
}


cat("Estimated pi M=2:\n")
cat(pi_list[[1]],"\n")


cat("Estimated pi M=3:\n")
cat(pi_list[[2]],"\n")

cat("Estimated pi M=4:\n")
cat(pi_list[[3]],"\n")

# - Plot likelihood vs iterations
```

```
par(mfrow=c(3,1), mai=c(0.3,1,0.3,1))
plot(llik[1:it,1], type="o", xlab="Iteration", ylab="Log likelihood",
     main=paste("M=2, final log likelihood=",
                round(llik[iter_list[[1]],1],1), ", ",
                iter_list[[1]], "iterations"))
plot(llik[1:it,2], type="o", xlab="Iteration", ylab="Log likelihood",
     main=paste("M=3, final log likelihood=",
                round(llik[iter_list[[2]],2],1), ", ",
                iter_list[[2]], "iterations"))
plot(llik[1:it,3], type="o", xlab="Iteration", ylab="Log likelihood",
     main=paste("M=4, final log likelihood=",
                round(llik[iter_list[[3]],3],1), ", ",
                iter_list[[3]], "iterations"))
```