Laboratory Block 2 Report

Johan Lundin (johlu023) Greeshma Jeev Koothuparambil (greko370)
Daisuke Ronald Ssegwanyi Yamashita (ronss490)
Sangeeth Sankunny Menon (sansa237)

2024-01-15

STATEMENT OF CONTRIBUTION

The first assignment was coded, interpreted, and analysed by Greeshma Jeev Koothuparambil and Sangeeth Sankunny Menon. The second assignment was coded, and analysed by Johan Lundin. Interpretation was done by Daisuke Ronald Ssegwanyi Yamashita.

ASSIGNMENT 1: ENSEMBLE METHODS

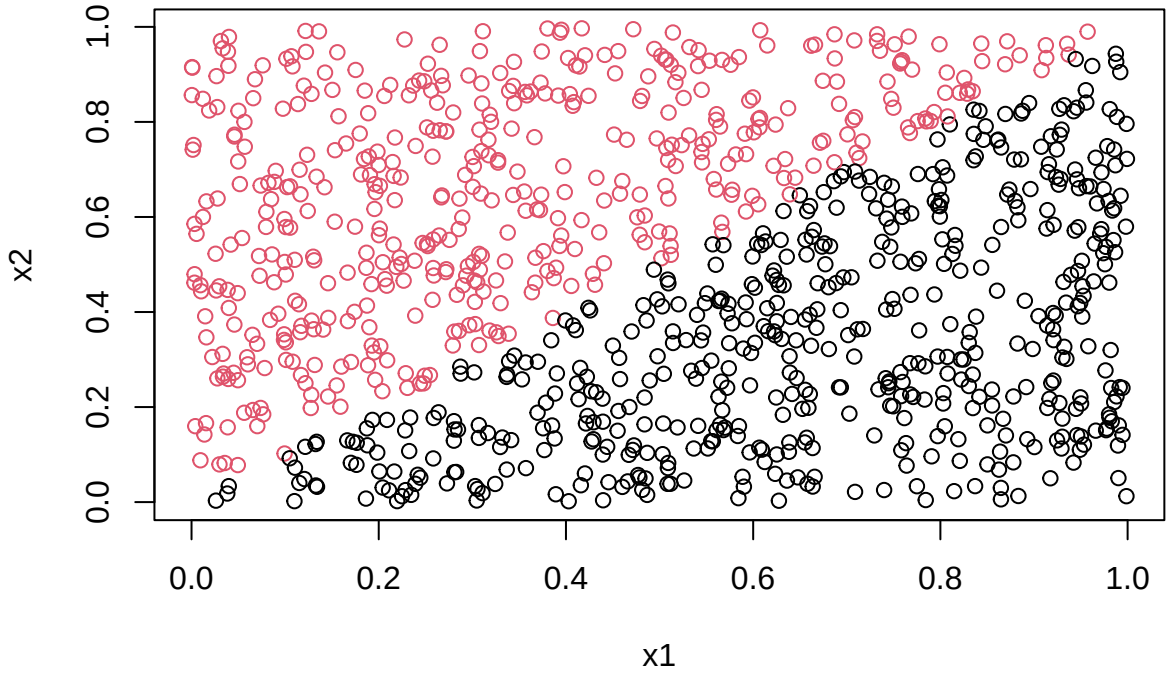
Your task is to learn some random forests using the function randomForest from the R package randomForest. The training data is produced by running the following R code.

```
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)
```

The task is therefore classifying Y from X1 and X2, where Y is binary and X1 and X2 continuous. You should learn a random forest with 1, 10 and 100 trees, which you can do by setting the argument ntree to the appropriate value. Use nodesize = 25 and keep.forest = TRUE. The latter saves the random forest learned. You need it because you should also compute the misclassification error in the following test dataset (use the function predict for this purpose).

```
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
telabels<-as.factor(y)
plot(x1,x2,col=(y+1))
```

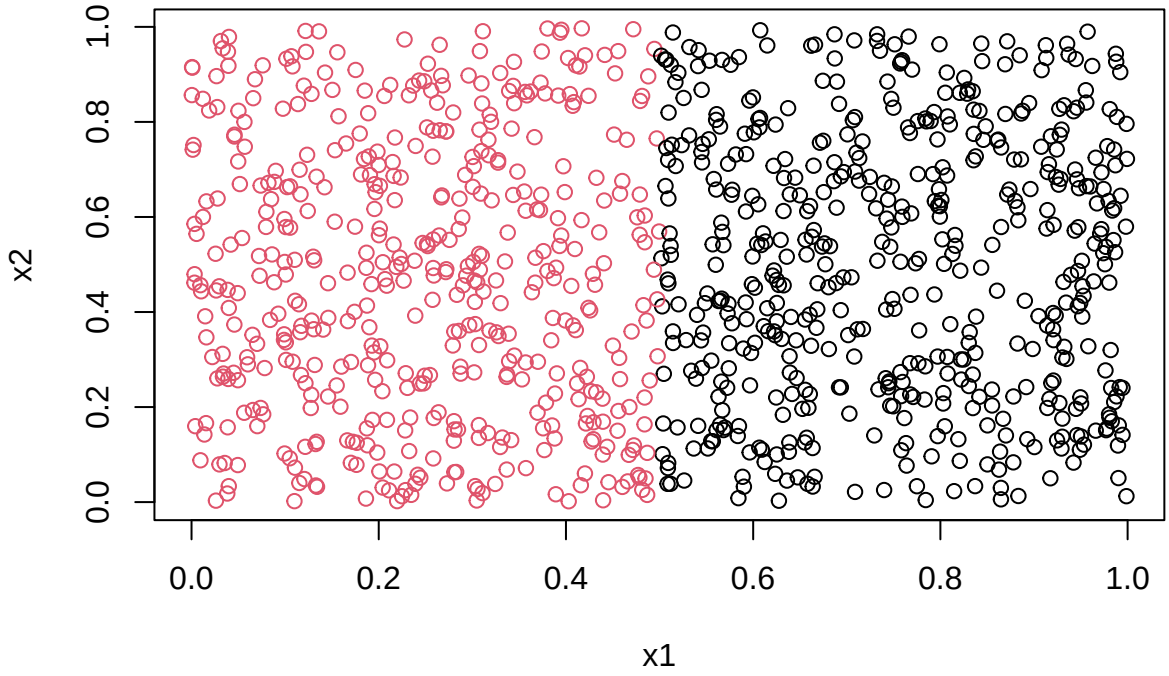
Answers



Repeat the procedure above for 1000 training datasets of size 100 and report the mean and variance of the misclassification errors. In other words, create 1000 training datasets of size 100, learn a random forest from each dataset, and compute the misclassification error in the same test dataset of size 1000. Report results for when the random forest has 1, 10 and 100 trees.

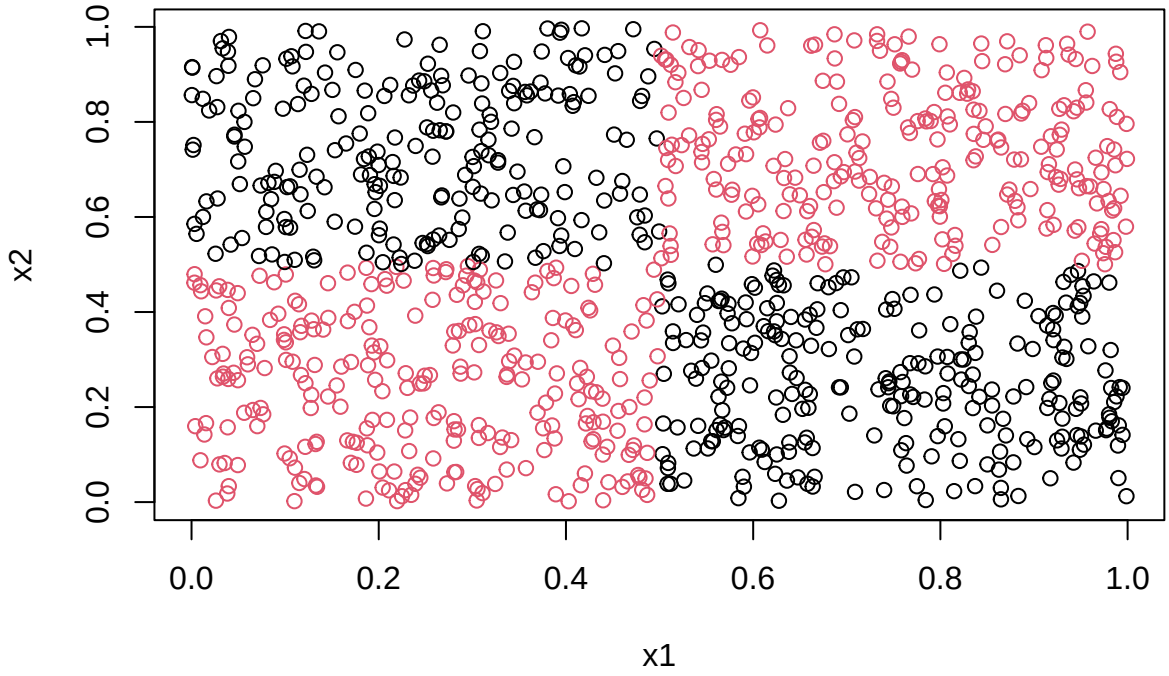
Misclassification error mean for model with 1 tree: 0.12906
Misclassification error variance for model with 1 tree: 0.0012463
Misclassification error mean for model with 10 tree: 0.075232
Misclassification error variance for model with 10 tree: 2.7648266×10^{-4}
Misclassification error mean for model with 100 tree: 0.065308
Misclassification error variance for model with 100 tree:, 2.3877791×10^{-4}

Repeat the exercise above but this time use the condition (x1<0.5) instead of (x1<x2) when producing the training and test datasets.



Misclassification error mean for model with 1 tree: 0.038341
Misclassification error variance for model with 1 tree: 0.0021858
Misclassification error mean for model with 10 tree: 0.007921
Misclassification error variance for model with 10 tree: 6.2509268×10^{-5}
Misclassification error mean for model with 100 tree: 0.006014
Misclassification error variance for model with 100 tree:, 4.6230034×10^{-5}

Repeat the exercise above but this time use the condition ((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5)) instead of (x1<x2) when producing the training and test datasets. Unlike above, use nodesize = 12 for this exercise



Misclassification error mean for model with 1 tree: 0.16879
Misclassification error variance for model with 1 tree: 0.0059838
Misclassification error mean for model with 10 tree: 0.066355
Misclassification error variance for model with 10 tree: 7.057267×10^{-4}
Misclassification error mean for model with 100 tree: 0.04656
Misclassification error variance for model with 100 tree:, 3.847011×10^{-4}

Answer the following questions:

What happens with the mean error rate when the number of trees in the random forest grows? Why?*

The mean error rate seems to decrease when the number of trees in the random forest grows. Lower error rates are due to reduced overfitting and capturing diverse patterns.

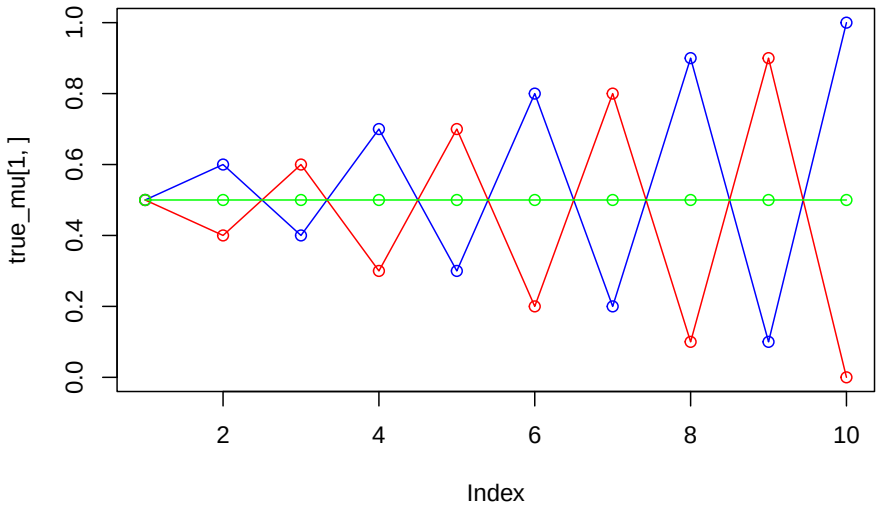
The third dataset represents a slightly more complicated classification problem than the first one. Still, you should get better performance for it when using sufficient trees in the random forest. Explain why you get better performance.

Smaller nodesize leads to larger decision trees in the random forest, reducing bias in average predicted values. As the number of trees B increases, the bagging effect diminishes variance, boosting prediction accuracy. While a single tree (B = 1) initially leads to higher bias and variance, increasing B significantly improves accuracy by reducing both bias and variance in the random forest's predictions.

ASSIGNMENT 2: BERNOULLI MIXTURE MODELS

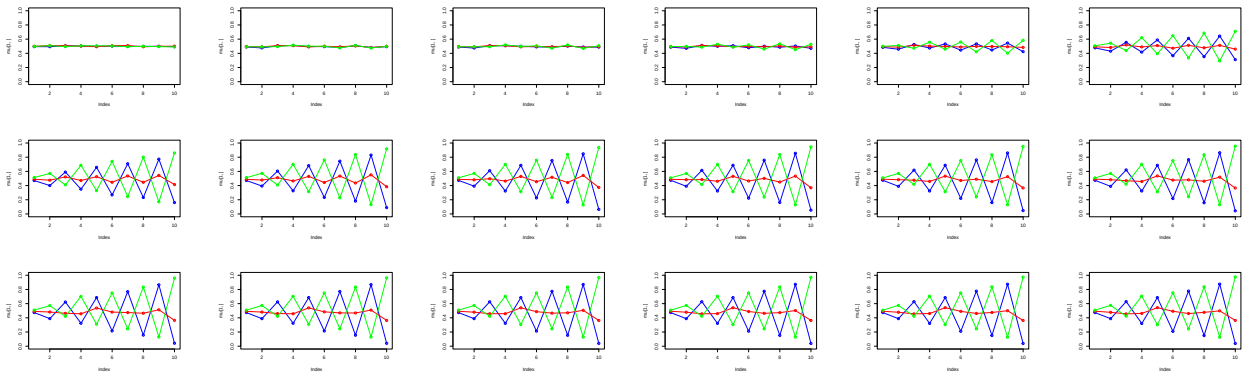
Your task is to implement the EM algorithm for Bernoulli mixture model. Please use the R template given to solve the assignment. Then, use your implementation to show what happens when your mixture model has too few and too many clusters, i.e. set M = 2, 3, 4 and compare results. Please provide a short explanation as well.

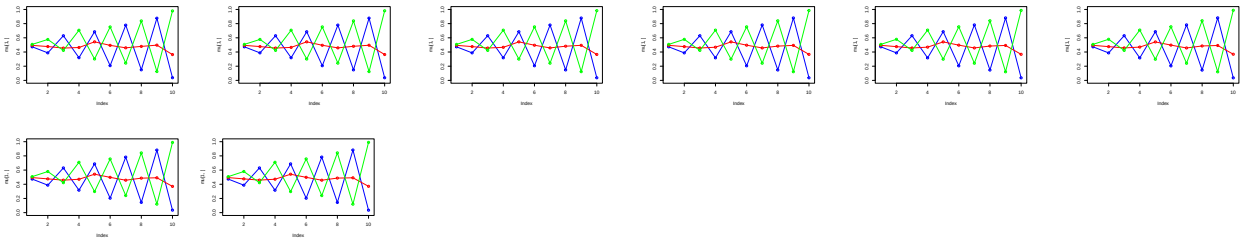
The template mentioned above is creating the training data (1000 points) and three clusters of data with a μ for each dimension, the true μ is shown in the plot below



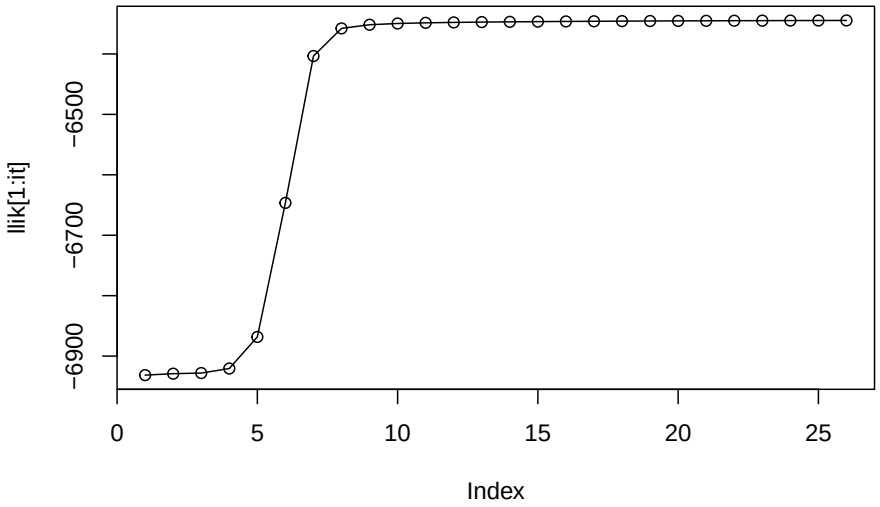
The Expectation-Maximization (EM) algorithm is iterative, and thus starts with some initial (μ) values and then calculates the log-liklihood for that set of values (E-step). Then it alters those values (M-step) and once again calculates the log-likelihood which should have improved. Then the iteration preocess continues until the change in log-likelihood for a iteration is significantly small to assume that we are close to a maximum (local or global).

In the plots below it can be seen how μ changes with each iteration.

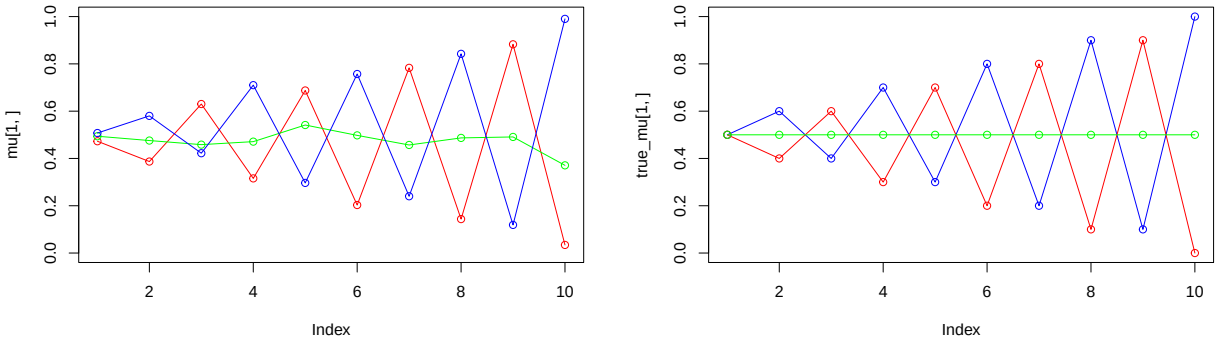




For each iteration the log-liklihood increases. It can be seen that the largest increase in the log-likelihood happens between iteration 4-8, and this can also be seen in the iteration plots were it μ starts to look very similar to the true μ around iteration plot 6-7.

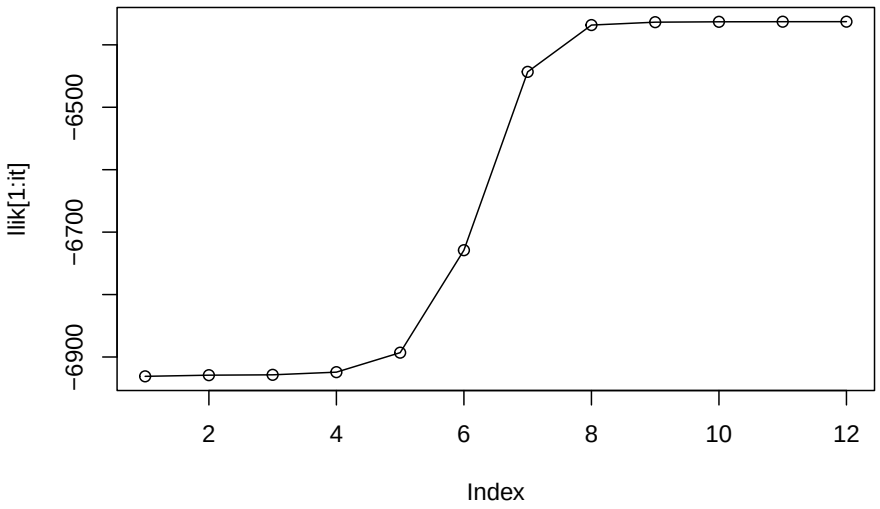


The calcualted μ (left) could then be compared with the true μ (right) in the plots below. The most obvious difference can be seen for the third true cluster (green, all μ values of 0.5 and thus a straight line in the plot) compared with the calculated μ where the third cluster is not a straight line. Note that the colors for this plot have been changed compared to the iteration-plots to make it easier to visually compare the clusters.

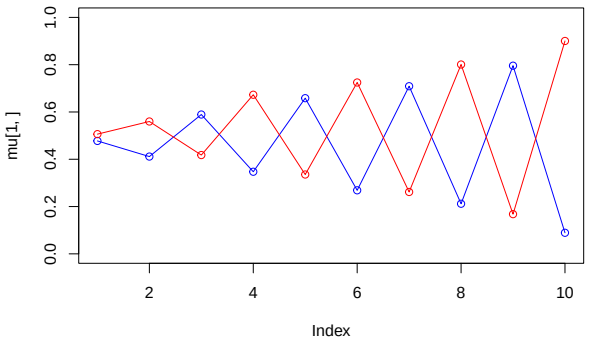


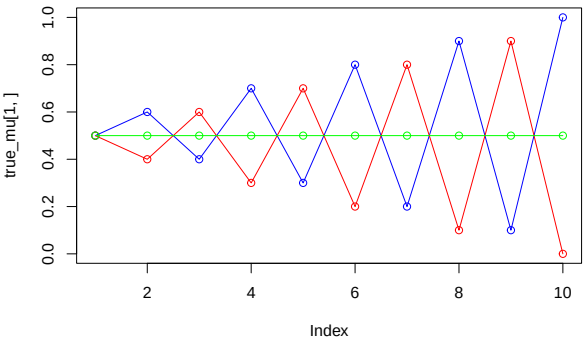
Two clusters (M=2)

Repeating the same procedure and algorithm as above, but this time we assume there are only two clusters gives the following log-likelihood plot, and between iteration 4 and 8 the likelihood increses significantly but with only 12 iterations before reaching the stopping critera.



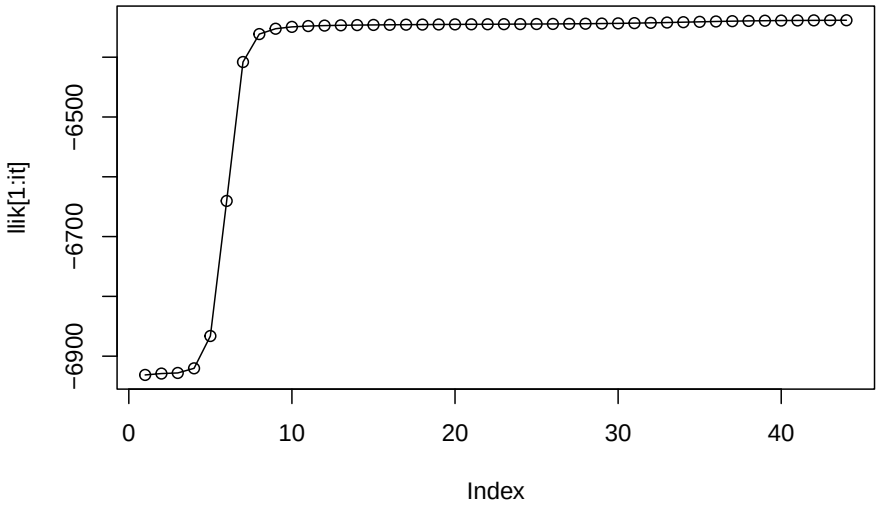
Comparing the calculated μ were only two clusters is assumed with the true μ which has three clusters, it can be seen that the first two are found, but that the third true cluster is affecting the two clusters we have assumed (with lower peaks).



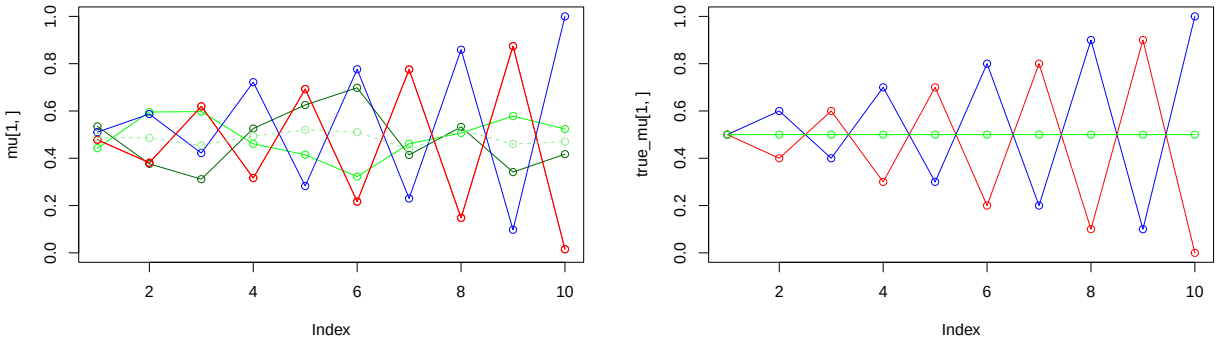


Four clusters (M=4)

Assuming four clusters give the following results



When there is assumed a fourth cluster it seems like the first two clusters are identified reasonably well. The colors have been changed manually to match the true μ for easier visual comparison. The dashed line is also the mean of the the two green clusters, just to point out the fact that the true third cluster is actually in a large extent a combination of the third and the fourth assumed clusters.



Question.

Use your implementation to show what happens when your mixture model has too few and too many clusters, i.e. set M = 2, 3, 4 and compare results. Please provide a short explanation as well.

In this mixture model when there are too few clusters, it may lead to the following consequences:

1. The model might exhibit high bias and low variance potentially leading to underfitting. The model may oversimplify and fail to distinguish between different patterns or subgroups in the data. As a result, the clusters formed may be too broad and not representative of the true underlying structure.
2. The model might exhibit merged or oversimplified clusters leading to loss of information and poor representation of the underlying data structure.

On the other hand, having too many clusters may lead to the following consequences:

1. The model might exhibit high variance and low bias leading to overfitting. This indicates high sensitivity to variations in the training data.

2. The model might capture noise and small variations in the data, leading to a less interpretable and less generalizable model. Having too many clusters may lead to an increased risk of finding spurious patterns that do not generalize well to new, unseen data.

CODE APPENDIX

##Assignment 1(ENSEMBLE METHODS)##

```
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)
```

```
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
telabels<-as.factor(y)
plot(x1,x2,col=(y+1))
```

```
library(randomForest)

#train data:
traingen1 <- function(){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<x2)
  trlabels<-as.factor(y)
  retval <- list(train = trdata, yval = trlabels)
  return(retval)
}


#test data:
testgen1 <- function(){
  set.seed(1234)
  x1<-runif(1000)
  x2<-runif(1000)
  tedata<-cbind(x1,x2)
  y<-as.numeric(x1<x2)
  telabels<-as.factor(y)
  plot(x1,x2,col=(y+1))

  retval <- list(test = tedata, yval = telabels)
  return(retval)

}


#misclass error:
misclass <- function(ar1, ar2){
  error <- length(which( ar1 != ar2))/length(ar1)
  return(error)
}
```

```
train <- traingen1()
test <- testgen1()
#random forest with ntree =1
random1 <- randomForest(x = train$train, y= train$yval, ntree=1, nsize= 25,
                        keep.forest = TRUE)
predict1 <- predict(random1, test$test)
missclasserror1 <- misclass(predict1, test$yval)

#random forest with ntree =10
random10 <- randomForest(x = train$train, y= train$yval, ntree=10, nsize= 25,
                         keep.forest = TRUE)
predict10 <- predict(random10,  test$test)
missclasserror10 <- misclass(predict10, test$yval)

#random forest with ntree =100
random100 <- randomForest(x = train$train, y= train$yval, ntree=100, nsize= 25,
                          keep.forest = TRUE)
predict100 <- predict(random100,  test$test)
missclasserror100 <- misclass(predict100, test$yval)
```

```
#1.1

testing <- function(n, test, traingen){
  missclassarray1 <- c()
  missclassarray10 <- c()
  missclassarray100 <- c()

  for (i in 1:n) {
    train <- traingen()
    #random forest with ntree =1
    random1 <- randomForest(x = train$train, y= train$yval, ntree=1, nsize= 25,
                            keep.forest = TRUE)
    predict1 <- predict(random1, test$test)
    missclassarray1[i] <- misclass(predict1, test$yval)

    #random forest with ntree =10
    random10 <- randomForest(x = train$train, y= train$yval, ntree=10,
                             nsize= 25, keep.forest = TRUE)
    predict10 <- predict(random10, test$test)
    missclassarray10[i] <- misclass(predict10, test$yval)

    #random forest with ntree =100
    random100 <- randomForest(x = train$train, y= train$yval, ntree=100,
                              nsize= 25, keep.forest = TRUE)
    predict100 <- predict(random100, test$test)
    missclassarray100[i] <- misclass(predict100, test$yval)


  }
   retvals <- list(missclass1_mean = mean(missclassarray1),
                   missclass10_mean = mean(missclassarray10),
                   missclass100_mean = mean(missclassarray100),
                   missclass1_var = var(missclassarray1),
  missclass10_var = var(missclassarray10),
```

```
  missclass100_var = var(missclassarray100))
  return(retvals)

}


testing1 <- testing(1000, test, traingen1)
```

```
#1.2

#train data:
traingen2 <- function(){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<0.5)
  trlabels<-as.factor(y)
  retval <- list(train = trdata, yval = trlabels)
  return(retval)
}

#train data
testgen2 <- function(){
  set.seed(1234)
  x1<-runif(1000)
  x2<-runif(1000)
  tedata<-cbind(x1,x2)
  y<-as.numeric(x1<0.5)
  telabels<-as.factor(y)
  plot(x1,x2,col=(y+1))

  retval <- list(test = tedata, yval = telabels)
  return(retval)

}

test2 <- testgen2()

testing2 <- testing(1000, test2, traingen2)
```

```
#1.3
testing_12 <- function(n, test, traingen){
  missclassarray1 <- c()
  missclassarray10 <- c()
  missclassarray100 <- c()

  for (i in 1:n) {
    train <- traingen()
    #random forest with ntree =1
    random1 <- randomForest(x = train$train, y= train$yval, ntree=1, nsize= 12,
                            keep.forest = TRUE)
    predict1 <- predict(random1, test$test)
    missclassarray1[i] <- misclass(predict1, test$yval)
```

```
    #random forest with ntree =10
    random10 <- randomForest(x = train$train, y= train$yval, ntree=10, nsize= 12,
                             keep.forest = TRUE)
    predict10 <- predict(random10, test$test)
    missclassarray10[i] <- misclass(predict10, test$yval)

    #random forest with ntree =100
    random100 <- randomForest(x = train$train, y= train$yval, ntree=100,
                              nsize= 12,
                              keep.forest = TRUE)
    predict100 <- predict(random100, test$test)
    missclassarray100[i] <- misclass(predict100, test$yval)


  }
  retvals <- list(missclass1_mean = mean(missclassarray1),
                  missclass10_mean = mean(missclassarray10),
                  missclass100_mean = mean(missclassarray100),
                  missclass1_var = var(missclassarray1),
                  missclass10_var = var(missclassarray10),
                  missclass100_var = var(missclassarray100))
  return(retvals)

}

#train data:
traingen3 <- function(){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric((x1<0.5& x2<0.5)|(x1>0.5& x2>0.5))
  trlabels<-as.factor(y)
  retval <- list(train = trdata, yval = trlabels)
  return(retval)
}

#train data
testgen3 <- function(){
  set.seed(1234)
  x1<-runif(1000)
  x2<-runif(1000)
  tedata<-cbind(x1,x2)
  y<-as.numeric((x1<0.5& x2<0.5)|(x1>0.5& x2>0.5))
  telabels<-as.factor(y)
  plot(x1,x2,col=(y+1))

  retval <- list(test = tedata, yval = telabels)
  return(retval)

}

test3 <- testgen3()
```

```
testing3 <- testing_12(1000, test3, traingen3)
```

##Assignment 2(BERNOULLI MIXTURE MODELS)##

Note that only the code for M=3 is given, the code for M=2 and M=4 is identical with the exception that the M-values have been changed.

The code for plotting , and checking for convergence with M = 2 and M = 4 is the same as M = 3 provided at the bottom of the appendix. Just that the number of clusters differ.

Code for M = 3

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

# 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])
  }
}
```

```
M=3                             # number of clusters
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

# Plotting the assumed number of clusters (mu)
```

```
for(it in 1:max_it) {
  plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  points(mu[2,], type="o", col="red")

  # If we assume 3 clusters, add a third line
  if (M >= 3){
  points(mu[3,], type="o", col="green")
  }

  # If we assume 4 clusters, add a fourth line
  if (M >= 4){
  points(mu[4,], type="o", col="yellow")
  }

  Sys.sleep(0.5)



  # E-step: Computation of the weights

   # Calculate the probability that a data point (x) belongs to a cluster

   # Loop1: Go through every data point
   for (p in 1:n)
   {
     #print(paste(p,"DATA POINT")) - USED FOR DEBUGGING

     # Initialize the sum of probabilities for each data point (p)
     prob.sum = 0

     #Loop2: Go through every cluster
     for (c in 1:M)

     {
        # print(paste(c,"CLUSTER")) - USED FOR DEBUGGING

        #Loop3: Go through every dimension of the data points

        # Initialize the Bernoulli probability for each cluster (c)
        bernxum.tot = 1

          for (d in 1:D)

          {
          # Equation for Bernoulli distribution (from 2nd equation in Lab description)
          bernxum.d = mu[c, d]^x[p, d] * (1 - mu[c, d])^(1 - x[p, d])

          # print(paste("",d,paste("dim, prob: ",bernxum.d)))  - USED FOR DEBUGGING

          # Product of all the probabilities for all dimensions
          bernxum.tot = bernxum.tot * bernxum.d
          }
```

```
        # print(paste("",c,paste("cluster, total prob: ",pro.sum)))  - USED FOR DEBUGGING

        # Put the calculated total sum into the w matrix
        w[p,c] = bernxum.tot * pi[c]

        # Sum up the probabilities to use in the log-likl vector
        prob.sum = prob.sum + w[p,c]
      }

      #Normalize the weights
      w[p,] = w[p,] / sum( w[p,] )

      # Add to the log-likl vector
      llik[it] = llik[it] + log ( prob.sum )

    }


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


  # Stop if the log likelihood has not changed significantly (min_change)
  if ( it > 1)
   {
    if (llik[it]-llik[it-1] < min_change)
      {
      print("Min change achieved. Stopping iterating")
      break
      }
    }


  #M-step: ML parameter estimation from the data and weights


  # Go through every cluster and update pi and mu
  for (c in 1:M)

  {
    # Equation 10.16a from the lecture slides
    pi[c] = 1/n * sum(w[,c])

    for (d in 1:D)

      {
      # Equation 10.16B from the lecture slides
      mu[c,d] = 1/sum(w[,c]) * sum(w[,c]* x[,d])
      }

  }

}
```

```
plot(llik[1:it], type="o")
```

```
  plot(mu[1,], type="o", col="red", ylim=c(0,1))
  points(mu[2,], type="o", col="green")
  points(mu[3,], type="o", col="blue")

  mu.3 = mu
    llik.3 = llik

  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")
```

Code for $M = 2$, and $M = 4$

The code for plotting , and checking for convergence with M = 2 and M = 4 is the same as M = 3 provided above. Just that the number of clusters differ.