Computer lab 1 block 2

Feras Faez Elias, Thalgamuwe Gedara Ashen Akalanka Weligalle, Olayemi Morrison

2023-12-06

Statement Of Contribution:

Assignment 1: Completed by Olayemi Morrison

Assignment 2: Completed by Thalgamuwe Gedara Ashen Akalanka Weligalle

For the report and analysis, all 3 group members went through the outputs and the analysis to make our own suggestions to the results in order to complete the report successfully.

The RMD file was designed together by Feras Faez and coded by Olayemi.

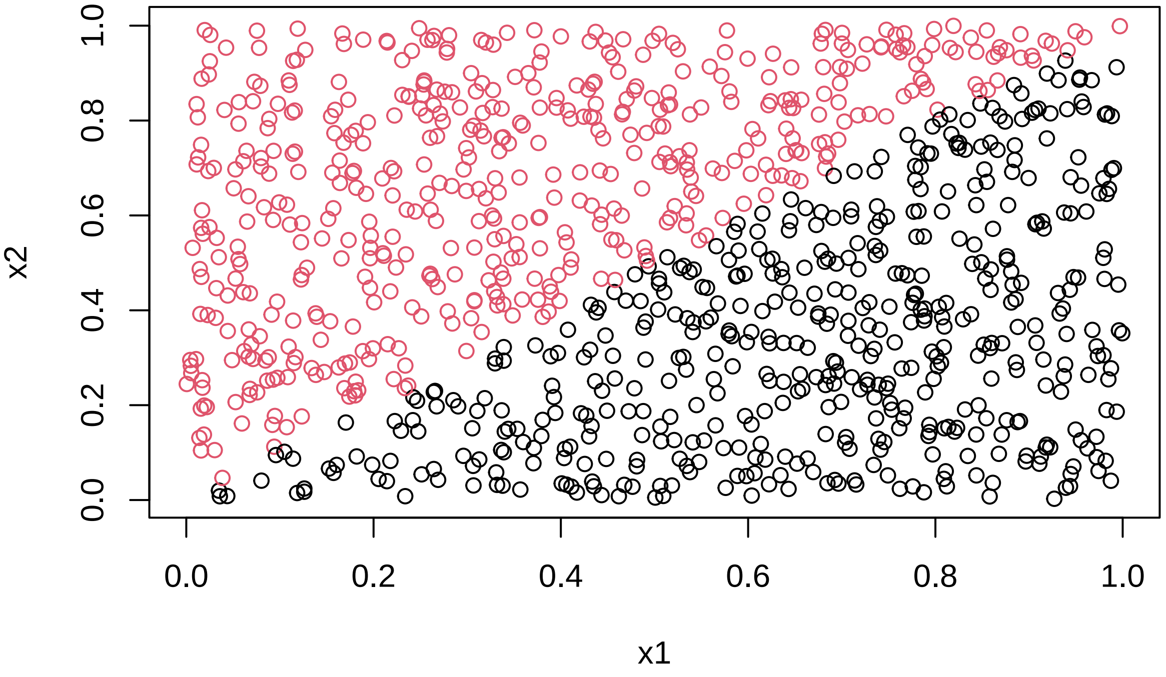
Assignment 1. ENSEMBLE METHODS

All codes are given in the appendix.

Task 1a: Classify Y from X1 and X2, where Y is binary and X1 and X2 continuous.

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

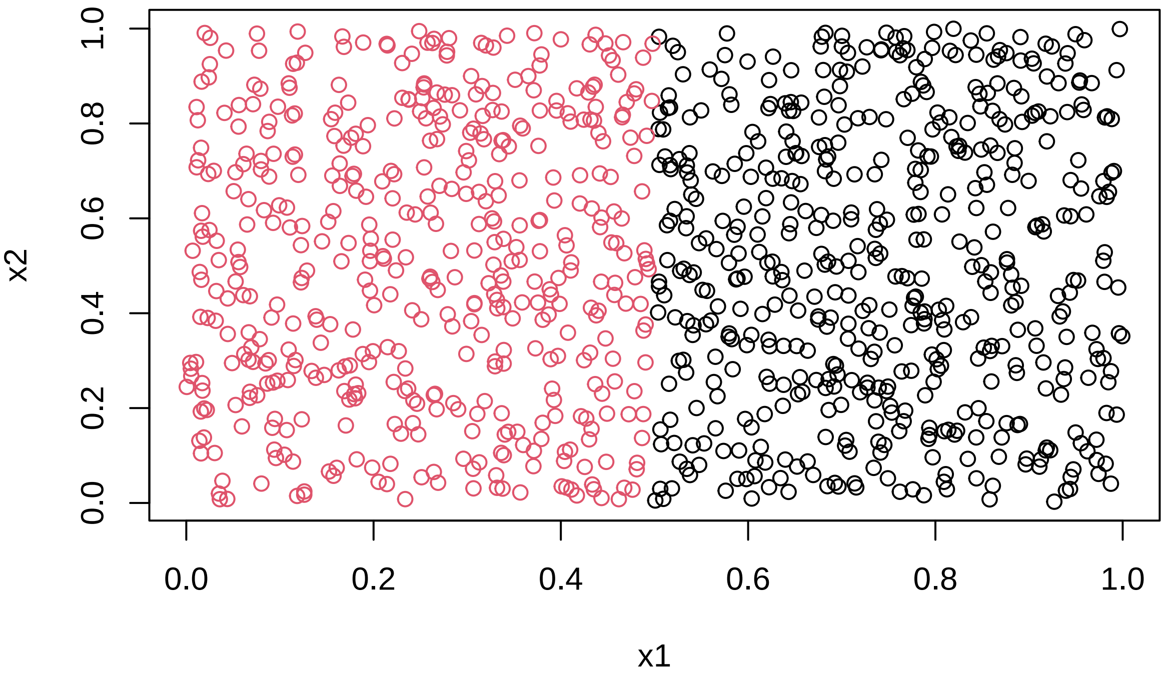


```
## Misclassification errors:
##  0.191 0.168 0.114
```

Task 1b: Repeat the procedure above for 1000 training datasets of size 100 and report the mean and variance of the misclassification errors.

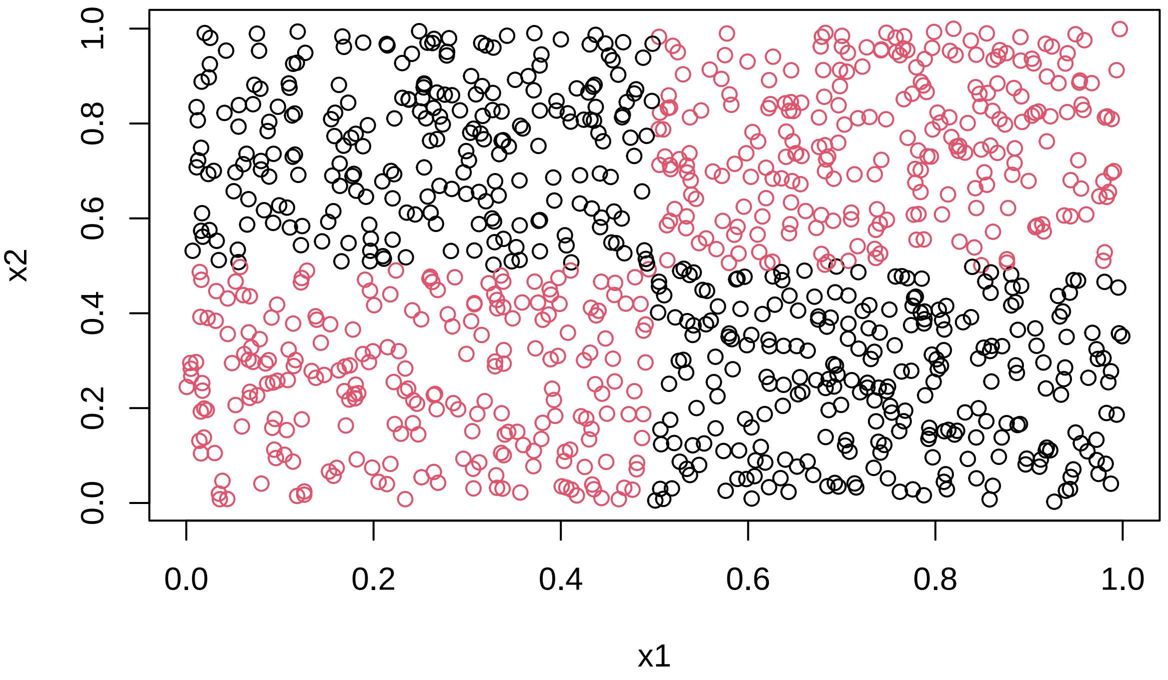
```
##        mean1       mean10      mean100         var1        var10       var100
## 0.2133750000 0.1409400000 0.1131460000 0.0036153377 0.0006881586 0.0001577785
```

Task 1c: Repeat the exercise above but this time use the condition (x1<0.5) instead of (x1<x2) when producing the training and test datasets.



```
##        mean1       mean10      mean100         var1        var10       var100
## 0.2133090000 0.1409870000 0.1131440000 0.0036139294 0.0006884913 0.0001577710
```

Task 1d: 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.



```
##        mean1        mean10       mean100          var1         var10        var100
## 0.1716520000 0.1044540000 0.0955980000 0.0030693683 0.0003405945 0.0000686030
```

Task 1e:

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

The mean error rates are as follows:

	mean1	mean10	mean100	var1	var10	var100
mis_values1	0.213375	0.140940	0.113146	0.0036153	0.0006882	0.0001578
mis_values2	0.213309	0.140987	0.113144	0.0036139	0.0006885	0.0001578
mis_values3	0.171652	0.104454	0.095598	0.0030694	0.0003406	0.0000686

From the table above, it is observed that as the number of trees increases, the mean error rate decreases. This is because with a larger number of trees, the variance in predictions tends to decrease, leading to a more stable and accurate model.

ii. 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.

The third dataset, being a more complex classification problem, benefits from a smaller node size in the decision tree, reducing bias but increasing variance. With a larger ntree, the bagging effect of the random forest helps mitigate variance, leading to improved prediction accuracy compared to the case with a single tree (ntree = 1). As ntree increases, both bias and variance decrease, resulting in enhanced overall prediction accuracy that aligns well with the dataset.

Assignment 2: MIXTURE MODELS

First we have to generate

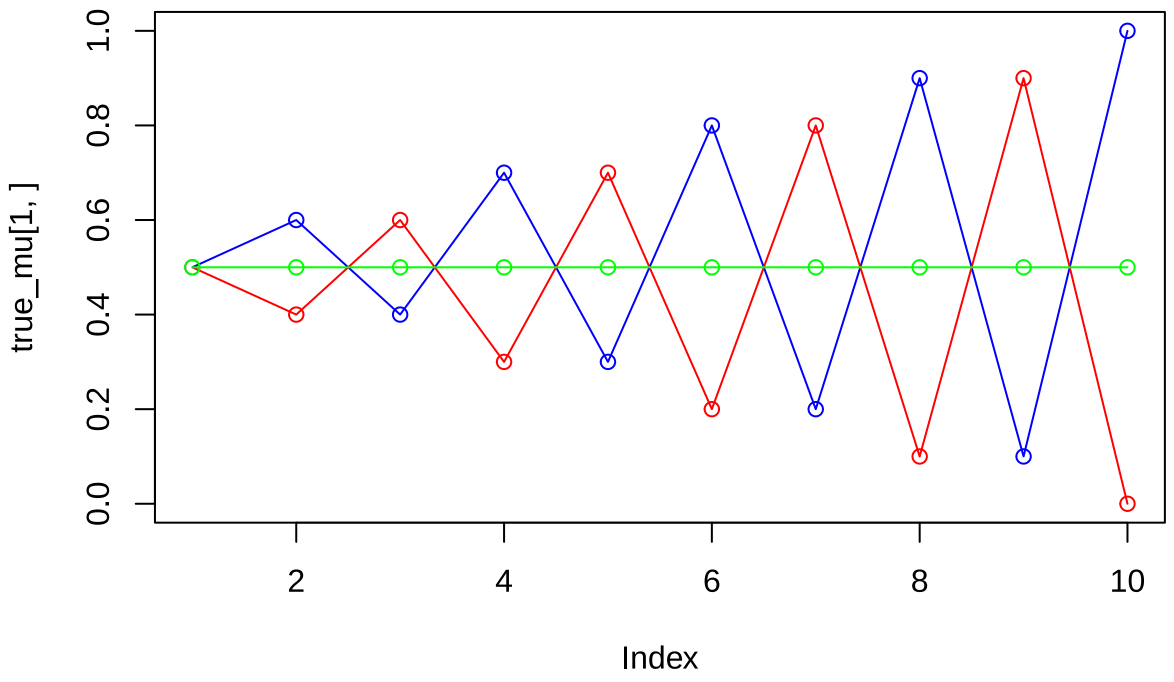
$$\pi$$

and

$$\mu$$

of the Bernoulli distribution.

```r
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")
```



According to above plot, There are 3

$$\mu$$

s. we are using same

$$\pi = 1/3$$

for each

$$\mu$$

.

```r
# 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])
  }
}
```

First we start with cluster M = 2.

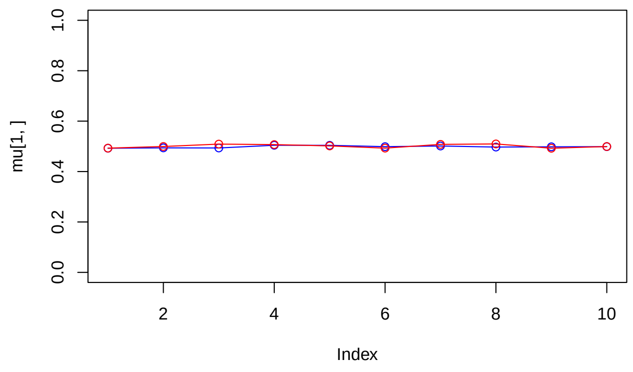
```r
M=2 # number of clusters
w <- matrix(nrow=n, ncol=M) # weights
pi <- vector(length = M) # mixing coefficients
mu <- matrix(nrow=M, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the parameters
pi <- runif(M,0.49,0.51)
pi <- pi / sum(pi)
for(m in 1:M) {
  mu[m,] <- runif(D,0.49,0.51)
}
pi
```

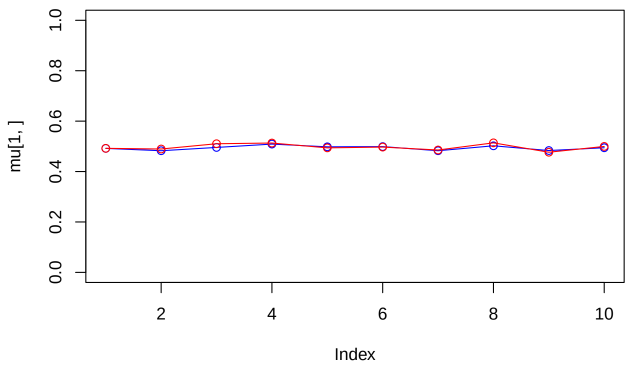
```
## [1] 0.4992145 0.5007855
```

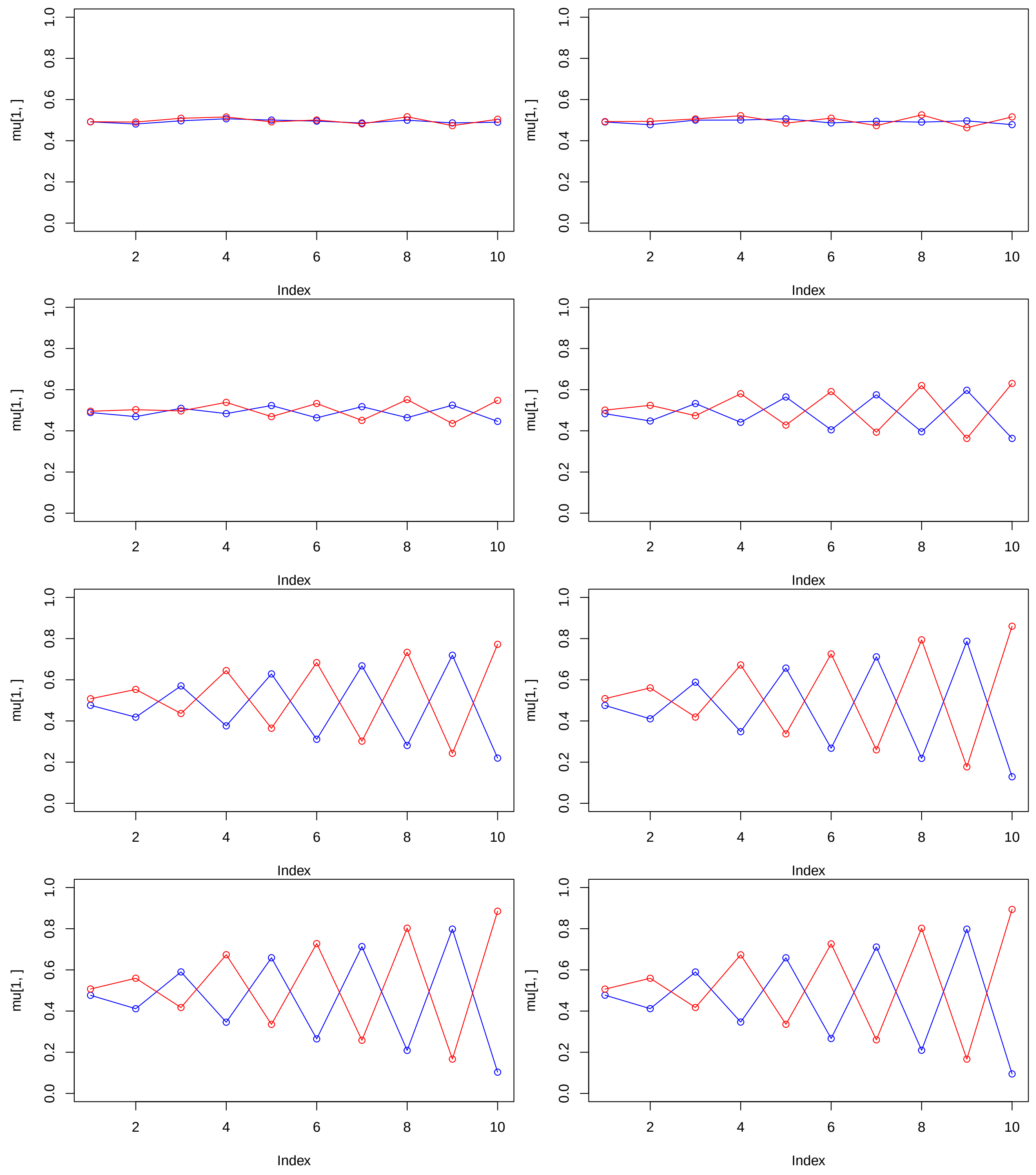
```r
mu
```

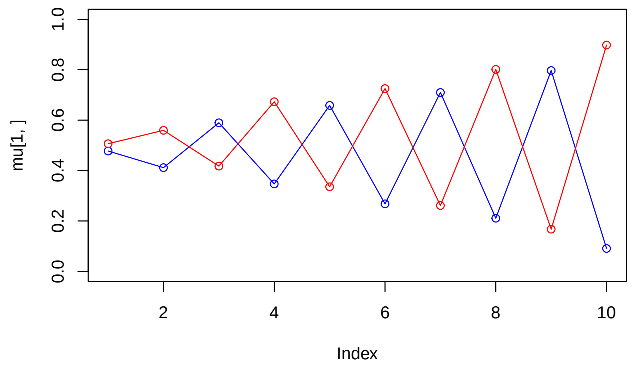
```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4924255 0.4939877 0.4935375 0.5042511 0.5040286 0.4987810 0.5012754
## [2,] 0.4929075 0.4993719 0.5088453 0.5068730 0.5016720 0.4929275 0.5077146
##           [,8]      [,9]     [,10]
## [1,] 0.4971036 0.4982144 0.4987654
## [2,] 0.5095075 0.4924574 0.4992470
```

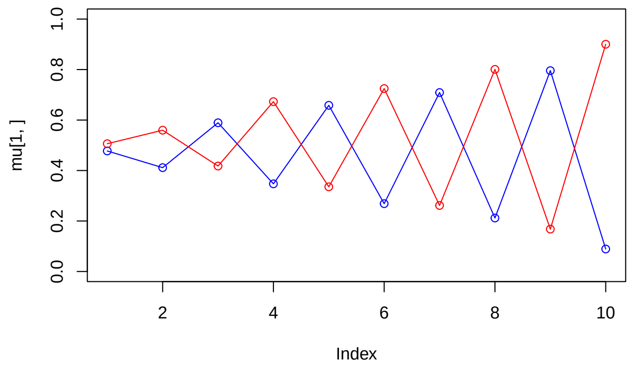
```r
set.seed(1234567890)
iterLog <- vector(length = max_it)

for(it in 1:max_it) {
  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 weights

  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) *(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
# print(paste(i,m,d,pxi))
      }
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
 #print(paste(i,m,d,bernXMum))
        }
      w[i,m] <- (bernXMum * pi[m]) / pxi
      }
    w[i, ] <- w[i,] /sum(w[i,])
    }

  # Log likelihood computation.
  llik[it] <- 0
  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <-bernXMum*(mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
      }
    llik[it] <- llik[it] + log(pxi)
  }

  iterLog[it] <- paste("iteration: ", it, "log likelihood: ", llik[it])

  flush.console()

  # Stop if the look likelihood has not changed significantly

if(it > 1 && (llik[it] - llik[it - 1]) < min_change) break

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

pi<- apply(w, 2, mean)
mu<- t(w) %*% x / colSums(w)

}
```



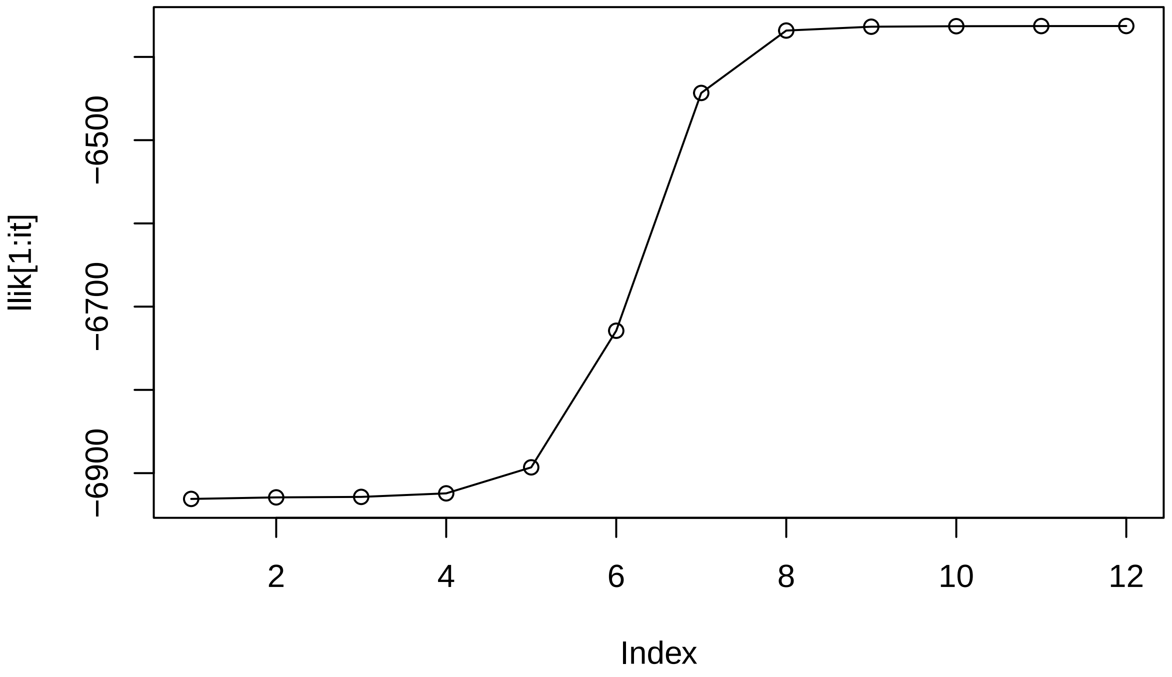



mu[1,]
Index
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10





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



```r
print(iterLog[1:it])
```

```
##  [1] "iteration:  1 log likelihood:  -6930.97504223822"
##  [2] "iteration:  2 log likelihood:  -6929.12473628598"
##  [3] "iteration:  3 log likelihood:  -6928.56238318311"
##  [4] "iteration:  4 log likelihood:  -6924.28062067272"
##  [5] "iteration:  5 log likelihood:  -6893.05518568101"
##  [6] "iteration:  6 log likelihood:  -6728.94831274647"
##  [7] "iteration:  7 log likelihood:  -6443.28038652938"
##  [8] "iteration:  8 log likelihood:  -6368.31837509574"
##  [9] "iteration:  9 log likelihood:  -6363.73371302523"
## [10] "iteration:  10 log likelihood:  -6363.10912783944"
## [11] "iteration:  11 log likelihood:  -6362.94687236738"
## [12] "iteration:  12 log likelihood:  -6362.89725846549"
```

```r
pi
```

```
## [1] 0.497125 0.502875
```

```r
mu
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4775488 0.4113939 0.5892308 0.3472420 0.6583712 0.2686589 0.7089490
## [2,] 0.5062860 0.5597531 0.4177551 0.6728856 0.3354854 0.7247188 0.2616231
##            [,8]      [,9]      [,10]
## [1,] 0.2118629 0.7957549 0.08905747
## [2,] 0.8007511 0.1678555 0.90027808
```

When the M is 2, we can see that the iteration runs 12 times, and the final log likelihood is -6362.8972. And the weight of these two distribution are around 0.497 and 0.502, which are almost 0.5. And with the

$$\mu$$

plot, we can notice that although there are only 2 distributions in this situation, the shapes are almost the same as the two roughest true

$$\mu$$

plot shows. And as the loglikelihood value plot shows, the value increases monotonically, and it change much slower after the 8th iteration. And the difference is smaller than 0.1 after 12 iteration.

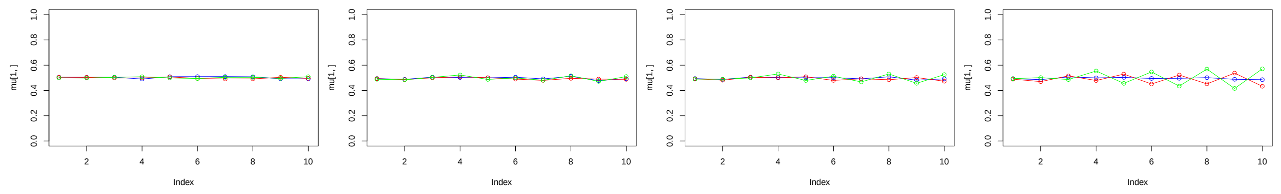
```r
set.seed(1234567890)

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
```

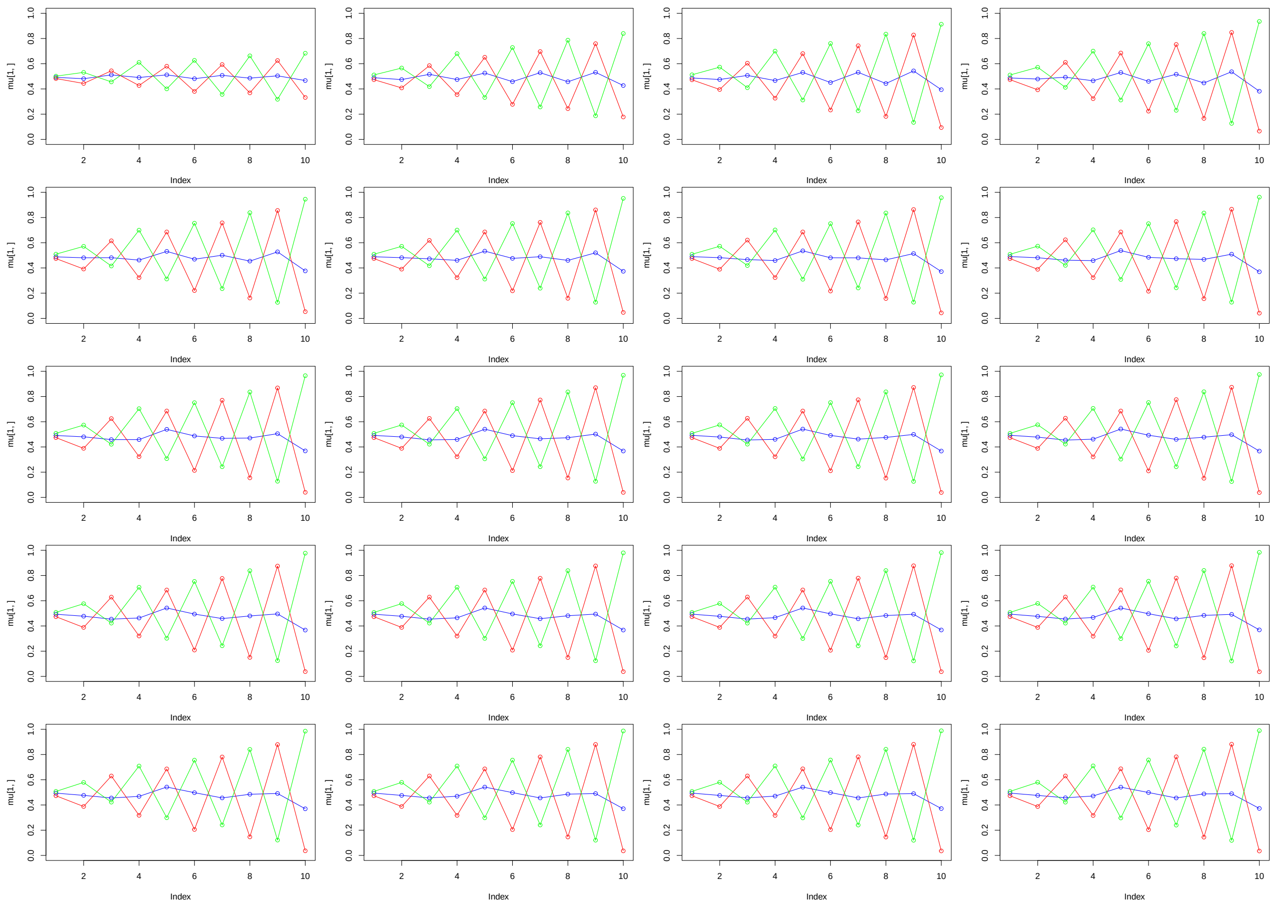
```
## [1] 0.3359578 0.3290183 0.3350239
```

```r
mu
```

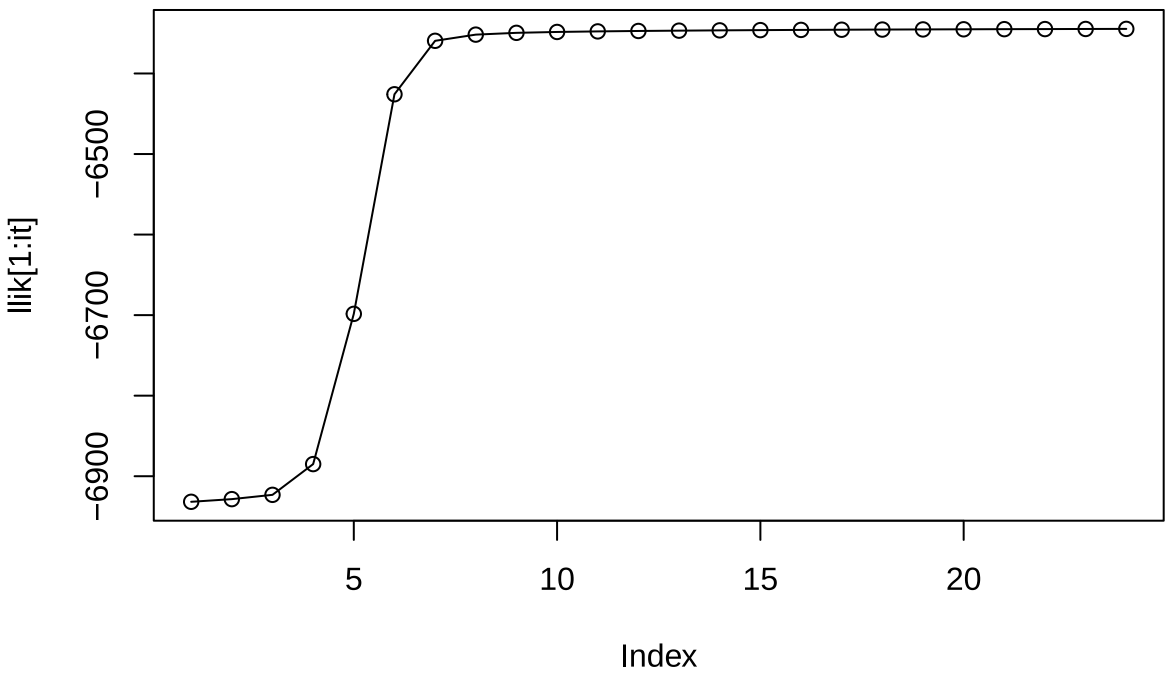
```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.5049455 0.5041509 0.5051382 0.4905578 0.5089228 0.5094530 0.5097480
## [2,] 0.5048207 0.5036886 0.4985467 0.4991731 0.5071384 0.4953800 0.4908757
## [3,] 0.4996279 0.4982070 0.5043346 0.5085042 0.4994862 0.4945702 0.5041462
##            [,8]      [,9]     [,10]
## [1,] 0.5082991 0.4926313 0.4921113
## [2,] 0.4917657 0.5040657 0.4956302
## [3,] 0.5040348 0.4955050 0.5088683
```

```r
iterLog <- vector(length = max_it)
for(it in 1:max_it) {
  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 weights

  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) *(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
# print(paste(i,m,d,pxi))
      }
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
 #print(paste(i,m,d,bernXMum))
        }
      w[i,m] <- (bernXMum * pi[m]) / pxi
      }
    w[i, ] <- w[i,] /sum(w[i,])
    }

  # Log likelihood computation.
  llik[it] <- 0
  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <-bernXMum*(mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
      }
    llik[it] <- llik[it] + log(pxi)
  }

  iterLog[it] <- paste("iteration: ", it, "log likelihood: ", llik[it])

  flush.console()

  # Stop if the look likelihood has not changed significantly

if(it > 1 && (llik[it] - llik[it - 1]) < min_change) break

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

pi<- apply(w, 2, mean)
mu<- t(w) %*% x / colSums(w)
}
```





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



```r
print(iterLog[1:it])
```

```
##  [1] "iteration:  1 log likelihood:  -6931.75122861544"
##  [2] "iteration:  2 log likelihood:  -6928.46695128332"
##  [3] "iteration:  3 log likelihood:  -6923.13091589768"
##  [4] "iteration:  4 log likelihood:  -6884.94468114899"
##  [5] "iteration:  5 log likelihood:  -6698.4237807576"
##  [6] "iteration:  6 log likelihood:  -6425.806406635"
##  [7] "iteration:  7 log likelihood:  -6359.4223002037"
##  [8] "iteration:  8 log likelihood:  -6351.74320723516"
##  [9] "iteration:  9 log likelihood:  -6349.54335699899"
## [10] "iteration:  10 log likelihood:  -6348.4434579416"
## [11] "iteration:  11 log likelihood:  -6347.73893891381"
## [12] "iteration:  12 log likelihood:  -6347.21907132148"
## [13] "iteration:  13 log likelihood:  -6346.8026280888"
## [14] "iteration:  14 log likelihood:  -6346.45328398564"
## [15] "iteration:  15 log likelihood:  -6346.15273441616"
## [16] "iteration:  16 log likelihood:  -6345.89062787433"
## [17] "iteration:  17 log likelihood:  -6345.66037682444"
## [18] "iteration:  18 log likelihood:  -6345.45729529842"
## [19] "iteration:  19 log likelihood:  -6345.2777404626"
## [20] "iteration:  20 log likelihood:  -6345.11870927259"
## [21] "iteration:  21 log likelihood:  -6344.97764443873"
## [22] "iteration:  22 log likelihood:  -6344.852335347"
## [23] "iteration:  23 log likelihood:  -6344.74086068395"
## [24] "iteration:  24 log likelihood:  -6344.64154880195"
```

```r
pi
```

```
## [1] 0.2663361 0.3438780 0.3897860
```

```r
mu
```

```
##           [,1]      [,2]     [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4938980 0.4758006 0.457040 0.4711376 0.5411178 0.4986316 0.4555200
## [2,] 0.4728756 0.3874569 0.630045 0.3163584 0.6869103 0.2039956 0.7823031
## [3,] 0.5075751 0.5799061 0.422322 0.7099547 0.2967462 0.7569457 0.2402904
##           [,8]     [,9]      [,10]
## [1,] 0.4878804 0.489488 0.37258194
## [2,] 0.1446699 0.881597 0.03501224
## [3,] 0.8422855 0.119219 0.98958938
```

When M is 3, we can see the iteration runs 24 times and the final log likelihood value is around -6362.897. And the weight of these three distributions are 0.3359, 0.3290 and 0.3350 which are all around 1/3. And the

$$\mu$$

plot is quite similar, especially for the two rough waving paths(blue and green paths), which have the weights 0.2663,0.3438 and 0.3897. And although the remaining path is not so similar to the original one, but it is still trends to be the same. So compared with the true thing, this estimate situation is good enough. And for the loglikelihood value, it also increases monotonically and changes much slower after the 7th iteration.

```r
set.seed(1234567890)

M=4 # 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
```

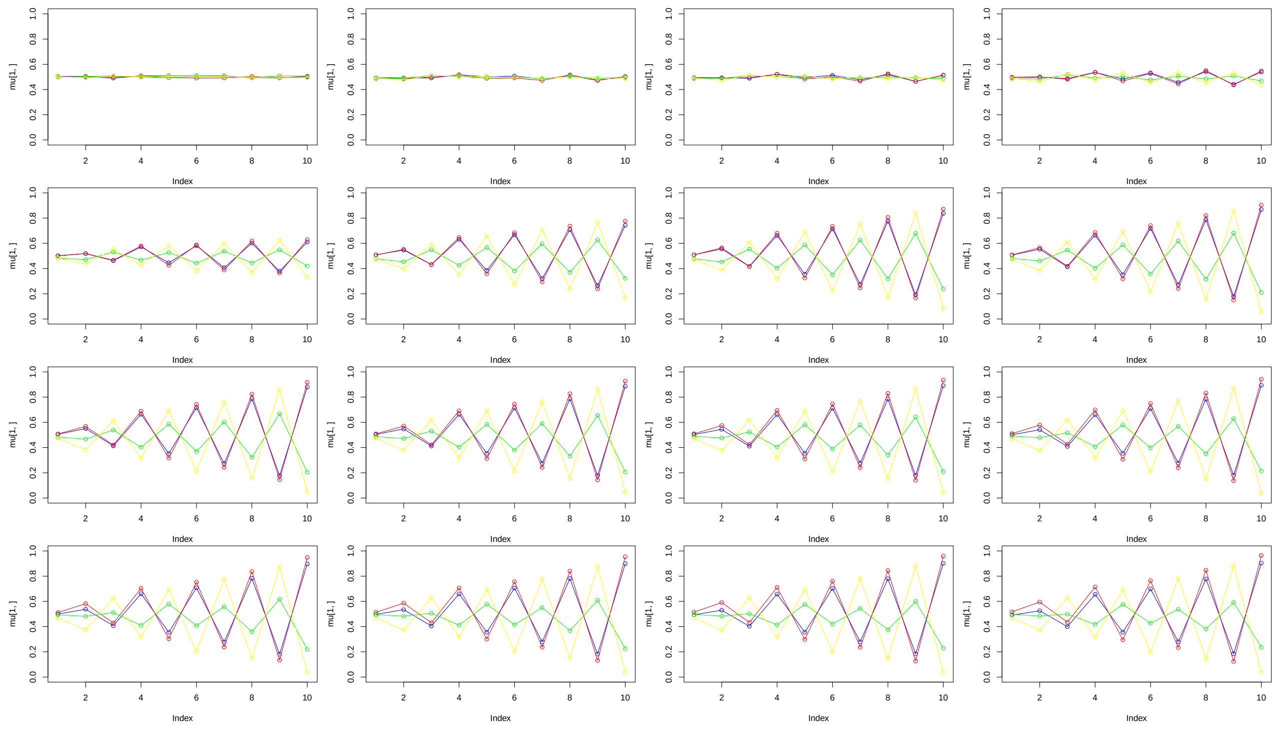
```
## [1] 0.2518811 0.2466783 0.2511809 0.2502598
```

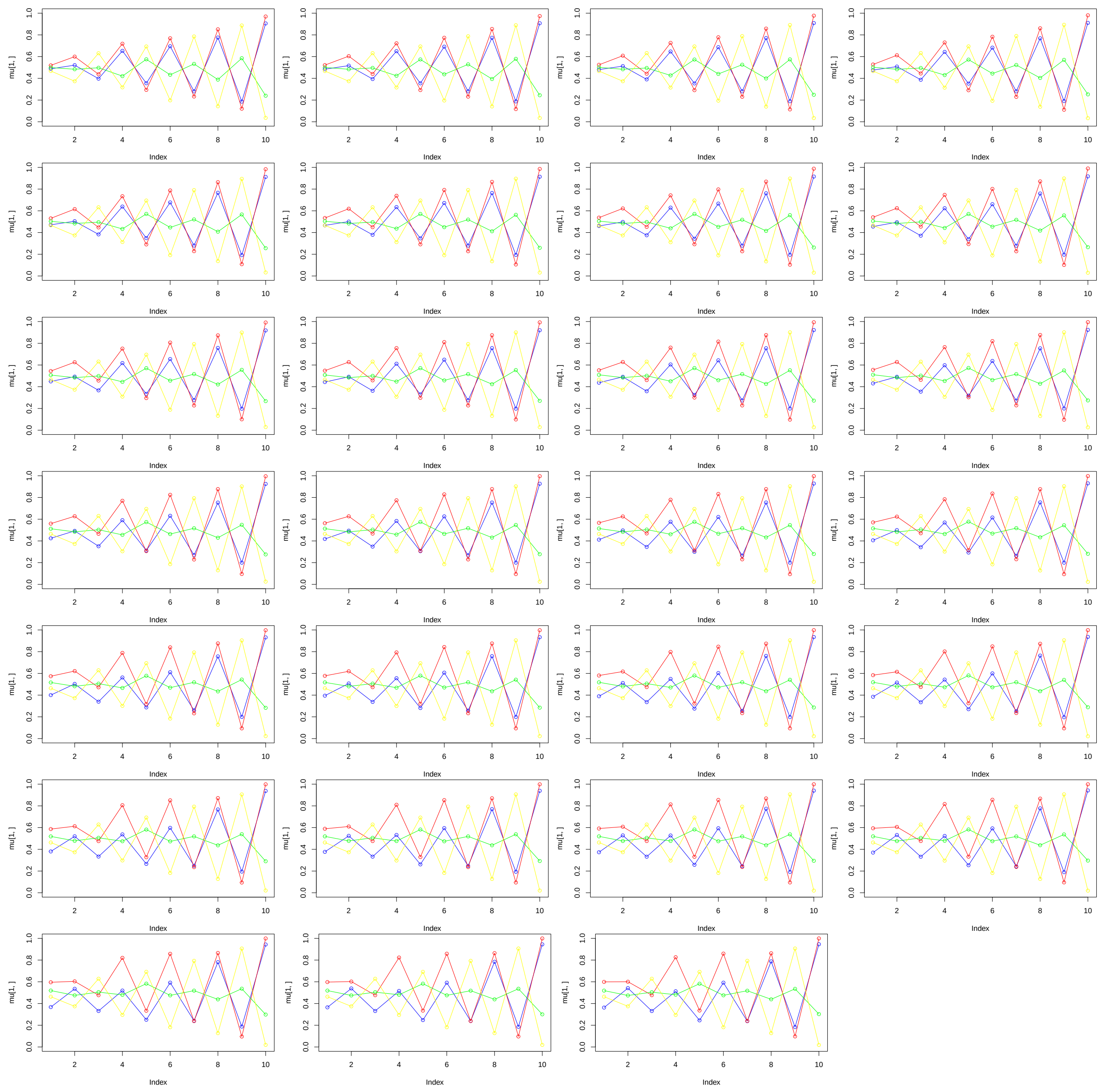
```
mu
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.5041509 0.5051382 0.4905578 0.5089228 0.5094530 0.5097480 0.5082991
## [2,] 0.5036886 0.4985467 0.4991731 0.5071384 0.4953800 0.4908757 0.4917657
## [3,] 0.4982070 0.5043346 0.5085042 0.4994862 0.4945702 0.5041462 0.5040348
## [4,] 0.5037389 0.4922173 0.5069624 0.5039756 0.5065369 0.5073122 0.5049473
##           [,8]      [,9]     [,10]
## [1,] 0.4926313 0.4921113 0.5048207
## [2,] 0.5040657 0.4956302 0.4996279
## [3,] 0.4955050 0.5088683 0.5072302
## [4,] 0.4943372 0.4951750 0.4940898
```

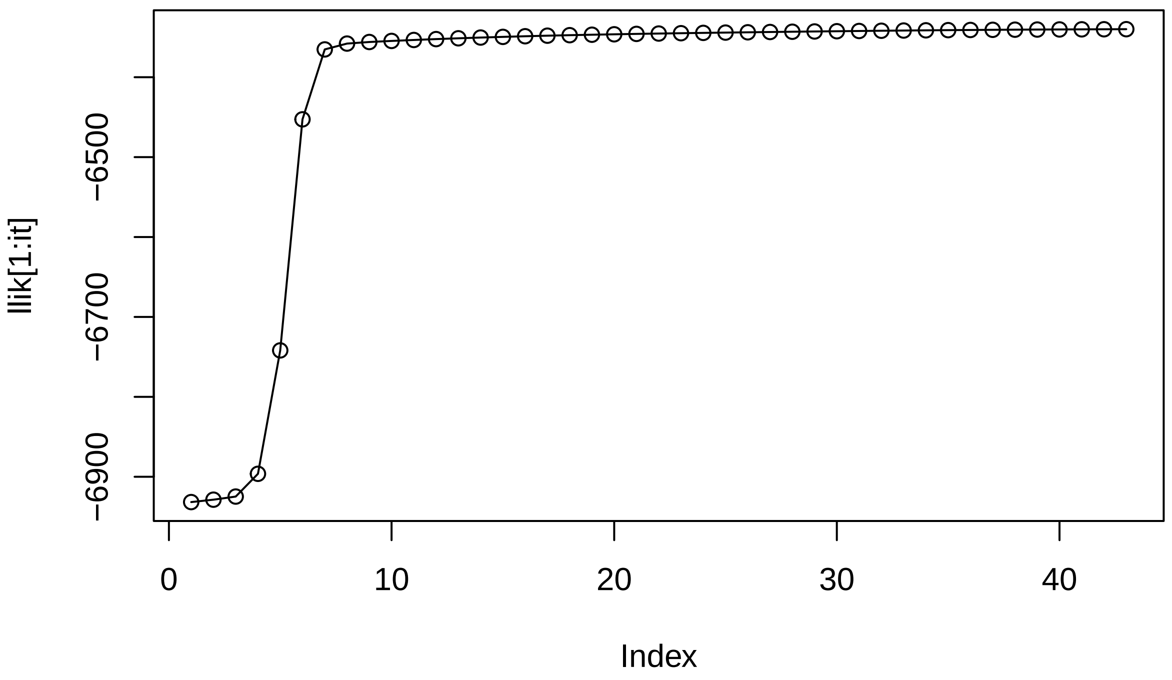
```
iterLog <- vector(length = max_it)
for(it in 1:max_it) {
  plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  points(mu[2,], type="o", col="red")
  points(mu[3,], type="o", col="green")
  points(mu[4,], type="o", col="yellow")
  Sys.sleep(0.5)
  # E-step: Computation of the weights

  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) *(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
# print(paste(i,m,d,pxi))
      }
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
 #print(paste(i,m,d,bernXMum))
        }
      w[i,m] <- (bernXMum * pi[m]) / pxi
      }
    w[i, ] <- w[i,] /sum(w[i,])
    }
```

```r
  # Log likelihood computation.
  llik[it] <- 0
  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <-bernXMum*(mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
      }
    llik[it] <- llik[it] + log(pxi)
  }

  iterLog[it] <- paste("iteration: ", it, "log likelihood: ", llik[it])

  flush.console()

  # Stop if the look likelihood has not changed significantly

if(it > 1 && (llik[it] - llik[it - 1]) < min_change) break

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

pi<- apply(w, 2, mean)
mu<- t(w) %*% x / colSums(w)
}
```





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



```r
print(iterLog[1:it])
```

```
##  [1] "iteration:  1 log likelihood:  -6931.66006994864"
##  [2] "iteration:  2 log likelihood:  -6928.66727678128"
##  [3] "iteration:  3 log likelihood:  -6924.77054344575"
##  [4] "iteration:  4 log likelihood:  -6896.29508885115"
##  [5] "iteration:  5 log likelihood:  -6741.85787805056"
##  [6] "iteration:  6 log likelihood:  -6452.71193962151"
##  [7] "iteration:  7 log likelihood:  -6365.22915492218"
##  [8] "iteration:  8 log likelihood:  -6357.90724015069"
##  [9] "iteration:  9 log likelihood:  -6355.94977356183"
## [10] "iteration:  10 log likelihood:  -6354.59564838264"
## [11] "iteration:  11 log likelihood:  -6353.40988750751"
## [12] "iteration:  12 log likelihood:  -6352.31517546734"
## [13] "iteration:  13 log likelihood:  -6351.30092218428"
## [14] "iteration:  14 log likelihood:  -6350.36818071176"
## [15] "iteration:  15 log likelihood:  -6349.51652090179"
## [16] "iteration:  16 log likelihood:  -6348.74223300225"
## [17] "iteration:  17 log likelihood:  -6348.03936943375"
## [18] "iteration:  18 log likelihood:  -6347.40094567819"
## [19] "iteration:  19 log likelihood:  -6346.81968070248"
## [20] "iteration:  20 log likelihood:  -6346.28834868353"
## [21] "iteration:  21 log likelihood:  -6345.79994106786"
## [22] "iteration:  22 log likelihood:  -6345.34778016045"
## [23] "iteration:  23 log likelihood:  -6344.92564619259"
## [24] "iteration:  24 log likelihood:  -6344.52792780162"
## [25] "iteration:  25 log likelihood:  -6344.14978243089"
## [26] "iteration:  26 log likelihood:  -6343.78728731843"
## [27] "iteration:  27 log likelihood:  -6343.43756285041"
## [28] "iteration:  28 log likelihood:  -6343.09885115178"
## [29] "iteration:  29 log likelihood:  -6342.77053143643"
## [30] "iteration:  30 log likelihood:  -6342.4530520589"
## [31] "iteration:  31 log likelihood:  -6342.14776283848"
## [32] "iteration:  32 log likelihood:  -6341.85664504623"
## [33] "iteration:  33 log likelihood:  -6341.58196036309"
## [34] "iteration:  34 log likelihood:  -6341.32586665114"
## [35] "iteration:  35 log likelihood:  -6341.09006544297"
## [36] "iteration:  36 log likelihood:  -6340.87554389922"
## [37] "iteration:  37 log likelihood:  -6340.68245157758"
## [38] "iteration:  38 log likelihood:  -6340.51011833165"
## [39] "iteration:  39 log likelihood:  -6340.35718728741"
## [40] "iteration:  40 log likelihood:  -6340.22181690354"
## [41] "iteration:  41 log likelihood:  -6340.10190243326"
## [42] "iteration:  42 log likelihood:  -6339.99527661232"
## [43] "iteration:  43 log likelihood:  -6339.89986543919"
```

```
pi
```

```
## [1] 0.1838186 0.2308466 0.2874187 0.2979162
```

```
mu
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.3627106 0.5416076 0.3311785 0.5125500 0.2451860 0.5918027 0.2376350
## [2,] 0.5991155 0.6005747 0.4768083 0.8260669 0.3357255 0.8593412 0.2392769
## [3,] 0.5188276 0.4740528 0.5042778 0.4812818 0.5823257 0.4755847 0.5177702
## [4,] 0.4628904 0.3744350 0.6280787 0.2945785 0.6916641 0.1817551 0.7930593
##           [,8]       [,9]      [,10]
## [1,] 0.7878580 0.18327423 0.94572455
## [2,] 0.8624625 0.09719753 0.99920005
## [3,] 0.4387190 0.53515949 0.30265960
## [4,] 0.1275007 0.90649071 0.01848283
```

When M is 4, we can see the iteration runs 43 times and the final log likelihood value is -6339.899. And the weight of these four distributions are 0.1838186, 0.2308466, 0.2874187 and 0.2979162. As the

$$\mu$$

plot shows, the yellow and green paths are almost the same as the green and blue paths in previous situation. And they have the weights 0.2518811, 0.2466783, 0.2511809 and 0.2502598 which is also close to 1/3. But the remaining path (green line) has the smaller weight(0.2466) and as the plot shows,it seems to be symmetric around the line

$$\mu$$

= 0.5. And the log likelihood value increases monotonically and changes much slower after the 7th iteration, just as the first two situations.

Appendix

Assignment 1

```
#Question 1a
#Generate train data
set.seed(12345)
x1<-runif(100)
x2<-runif(100)
traindata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trainlabel<-as.factor(y)
trdata<- as.data.frame(traindata)
trdata<-dplyr::mutate(trdata,trainlabel)
```

```r
#Generate test data
x1<-runif(1000)
x2<-runif(1000)
testdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
testlabel<-as.factor(y)
tsdata<-as.data.frame(testdata)
tsdata<-dplyr::mutate(tsdata,testlabel)
plot(x1,x2,col=(y+1))

#Create misclassification error function:
misclassification <- function(a,b){
  err <- length(which(a!= b))/length(a)
  return(err)
}


#Create random forest when ntree =1
forest1 <- randomForest(trainlabel~., data = trdata, ntree= 1,
                        nodesize=25, keep.forest=TRUE)
pred1 <- predict(forest1, tsdata)
misclass_err1 <- misclassification(pred1, tsdata$testlabel)

#Create random forest when ntree =10
forest10 <- randomForest(trainlabel~., data = trdata, ntree= 10,
                         nodesize=25, keep.forest=TRUE)
pred10 <- predict(forest10,  tsdata)
misclass_err10 <- misclassification(pred10, tsdata$testlabel)

#Create random forest when ntree =100
forest100 <- randomForest(trainlabel~., data = trdata, ntree= 100,
                          nodesize=25, keep.forest=TRUE)
pred100 <- predict(forest100,  tsdata)
misclass_err100 <- misclassification(pred100, tsdata$testlabel)


#Question 1b
miserror <- function(n, test, train){
  mis_class1 <- c()
  mis_class10 <- c()
  mis_class100 <- c()

  for (i in 1:n) {

    #random forest with ntree =1
    forest1 <- randomForest(trainlabel~., data = trdata, ntree= 1,
                            nodesize=25, keep.forest=TRUE)
    pred1 <- predict(forest1, tsdata)
    mis_class1[i] <- misclassification(pred1, tsdata$testlabel)

    #random forest with ntree =10
    forest10 <- randomForest(trainlabel~., data = trdata, ntree= 10,
                             nodesize=25, keep.forest=TRUE)
    pred10 <- predict(forest10, tsdata)
    mis_class10[i] <- misclassification(pred10, tsdata$testlabel)

    #random forest with ntree =100
    forest100 <- randomForest(trainlabel~., data = trdata, ntree=100,
                            nodesize= 25, keep.forest = TRUE)
    pred100 <- predict(forest100, tsdata)
    mis_class100[i] <- misclassification(pred100, tsdata$testlabel)


  }
  result <- c(mean1 = mean(mis_class1),
                mean10 = mean(mis_class10),
                mean100 = mean(mis_class100),
               var1 = var(mis_class1),
               var10 = var(mis_class10),
               var100 = var(mis_class100))
  return(result)

}


mis_values1 <- miserror(1000, tsdata, trdata)


#Question 1c

#Generate train data2
set.seed(12345)
x1<-runif(100)
x2<-runif(100)
traindata2<-cbind(x1,x2)
y2<-as.numeric(x1<0.5)
trainlabel2<-as.factor(y2)
trdata2<- as.data.frame(traindata2)
trdata2<-dplyr::mutate(trdata2,trainlabel2)

#Generate test data2
x1<-runif(1000)
x2<-runif(1000)
testdata2<-cbind(x1,x2)
y2<-as.numeric(x1<0.5)
testlabel2<-as.factor(y2)
tsdata2<-as.data.frame(testdata2)
tsdata2<-dplyr::mutate(tsdata2,testlabel2)
plot(x1,x2,col=(y2+1))


mis_values2 <- miserror(1000, tsdata2, trdata2)


#Question1d
```

```r
#Generate train data3
set.seed(12345)
x1<-runif(100)
x2<-runif(100)
traindata3<-cbind(x1,x2)
y3<-as.numeric((x1<0.5& x2<0.5)|(x1>0.5& x2>0.5))
trainlabel3<-as.factor(y3)
trdata3<- as.data.frame(traindata3)
trdata3<-dplyr::mutate(trdata3,trainlabel3)

#Generate test data3
x1<-runif(1000)
x2<-runif(1000)
testdata3<-cbind(x1,x2)
y3<-as.numeric((x1<0.5& x2<0.5)|(x1>0.5& x2>0.5))
testlabel3<-as.factor(y3)
tsdata3<-as.data.frame(testdata3)
tsdata3<-dplyr::mutate(tsdata3,testlabel3)
plot(x1,x2,col=(y3+1))

miserror2 <- function(n, test, train){
  mis_class1 <- c()
  mis_class10 <- c()
  mis_class100 <- c()

  for (i in 1:n) {

    #random forest with ntree =1
    forest1 <- randomForest(trainlabel~., data = trdata, ntree= 1,
                            nodesize=12, keep.forest=TRUE)
    pred1 <- predict(forest1, tsdata)
    mis_class1[i] <- misclassification(pred1, tsdata$testlabel)

    #random forest with ntree =10
    forest10 <- randomForest(trainlabel~., data = trdata, ntree= 10,
                             nodesize=12, keep.forest=TRUE)
    pred10 <- predict(forest10, tsdata)
    mis_class10[i] <- misclassification(pred10, tsdata$testlabel)

    #random forest with ntree =100
    forest100 <- randomForest(trainlabel~., data = trdata, ntree=100,
                              nodesize= 12, keep.forest = TRUE)
    pred100 <- predict(forest100, tsdata)
    mis_class100[i] <- misclassification(pred100, tsdata$testlabel)


  }
  result <- c(mean1 = mean(mis_class1),
               mean10 = mean(mis_class10),
               mean100 = mean(mis_class100),
               var1 = var(mis_class1),
               var10 = var(mis_class10),
               var100 = var(mis_class100))
  return(result)

}

mis_values3 <- miserror2(1000, tsdata3, trdata3)

#Question 1e
df <- data.frame(mis_values1, mis_values2, mis_values3)
df<- t(df)
knitr::kable(df)
```

Assignment 2

```
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=2 # 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

set.seed(1234567890)
iterLog <- vector(length = max_it)
```

```
for(it in 1:max_it) {
  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 weights

  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) *(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
# print(paste(i,m,d,pxi))
      }
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
 #print(paste(i,m,d,bernXMum))
        }
      w[i,m] <- (bernXMum * pi[m]) / pxi
      }
    w[i, ] <- w[i,] /sum(w[i,])
    }

  # Log likelihood computation.
  llik[it] <- 0
  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <-bernXMum*(mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
      }
    llik[it] <- llik[it] + log(pxi)
  }

  iterLog[it] <- paste("iteration: ", it, "log likelihood: ", llik[it])

  flush.console()

  # Stop if the look likelihood has not changed significantly

if(it > 1 && (llik[it] - llik[it - 1]) < min_change) break

  #M-step: ML parameter estimation from the data and weights
  # Your code here
```

```
pi<- apply(w, 2, mean)
mu<- t(w) %*% x / colSums(w)

}

plot(llik[1:it], type="o")
print(iterLog[1:it])
pi
mu

set.seed(1234567890)

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


iterLog <- vector(length = max_it)
for(it in 1:max_it) {
  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 weights

  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) *(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
# print(paste(i,m,d,pxi))
      }
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
 #print(paste(i,m,d,bernXMum))
        }
      w[i,m] <- (bernXMum * pi[m]) / pxi
      }
    w[i, ] <- w[i,] /sum(w[i,])
    }

  # Log likelihood computation.
  llik[it] <- 0
  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <-bernXMum*(mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
      }
    llik[it] <- llik[it] + log(pxi)
  }

  iterLog[it] <- paste("iteration: ", it, "log likelihood: ", llik[it])

  flush.console()

  # Stop if the look likelihood has not changed significantly

if(it > 1 && (llik[it] - llik[it - 1]) < min_change) break

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

pi<- apply(w, 2, mean)
mu<- t(w) %*% x / colSums(w)
}

plot(llik[1:it], type="o")
print(iterLog[1:it])
pi
mu

set.seed(1234567890)

M=4 # 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
```

```r
iterLog <- vector(length = max_it)
for(it in 1:max_it) {
  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 weights

  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) *(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
# print(paste(i,m,d,pxi))
      }
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
 #print(paste(i,m,d,bernXMum))
        }
      w[i,m] <- (bernXMum * pi[m]) / pxi
      }
    w[i, ] <- w[i,] /sum(w[i,])
    }

  # Log likelihood computation.
  llik[it] <- 0
  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <-bernXMum*(mu[m,d]^x[i,d])*(1-mu[m,d])^(1-x[i,d])
        }
      pxi <- pxi + pi[m] * bernXMum
      }
    llik[it] <- llik[it] + log(pxi)
  }

  iterLog[it] <- paste("iteration: ", it, "log likelihood: ", llik[it])

  flush.console()

  # Stop if the look likelihood has not changed significantly

if(it > 1 && (llik[it] - llik[it - 1]) < min_change) break
```

```
  #M-step: ML parameter estimation from the data and weights
  # Your code here

pi<- apply(w, 2, mean)
mu<- t(w) %*% x / colSums(w)
}

plot(llik[1:it], type="o")
print(iterLog[1:it])
pi
mu
```