Block 2 Lab 1 Report

Samia Noreen Butt

23/12/2019

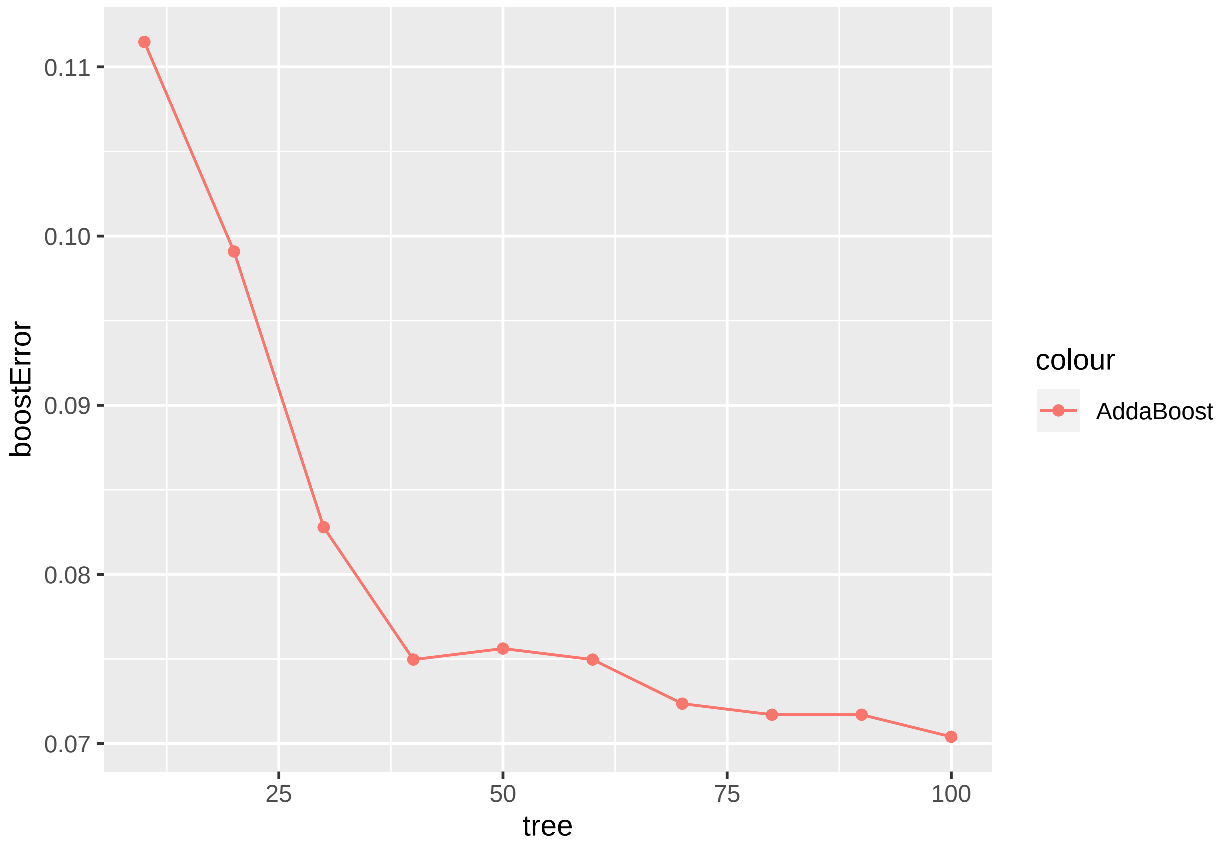
Summary

In this assignment, we are implementing ensemble methods and the EM algorithm. In ensemble methods, we are implementing Addaboost and RandomForest and then comparing their performance using error rate. In the EM algorithm task, we are evaluating the algorithm for different numbers of guessed components.

ENSEMBLE METHODS

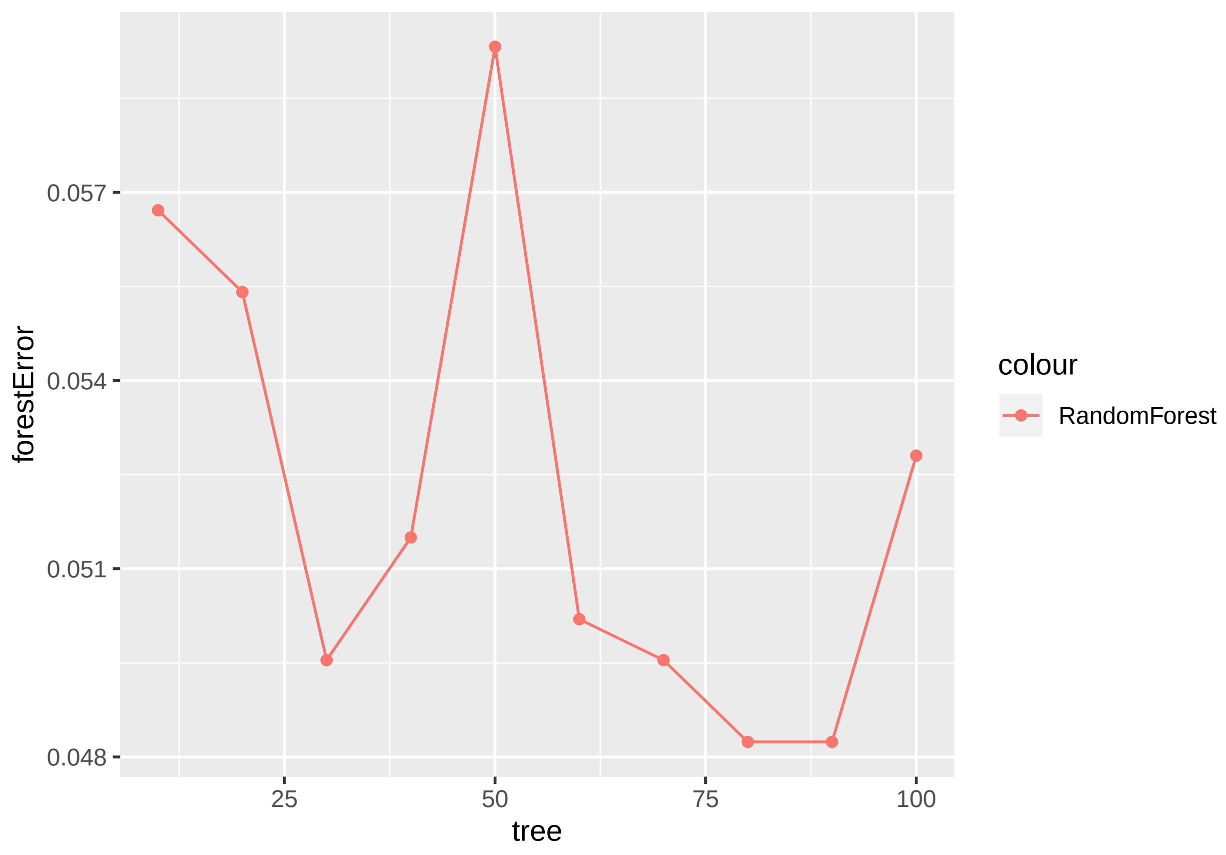
In this task, we are dividing data into 2/3 for the training dataset and the rest of the data we are considering for the test data. We are calculating errors for both Addabboost and RandomForest and then comparing their results.

AddaBoost



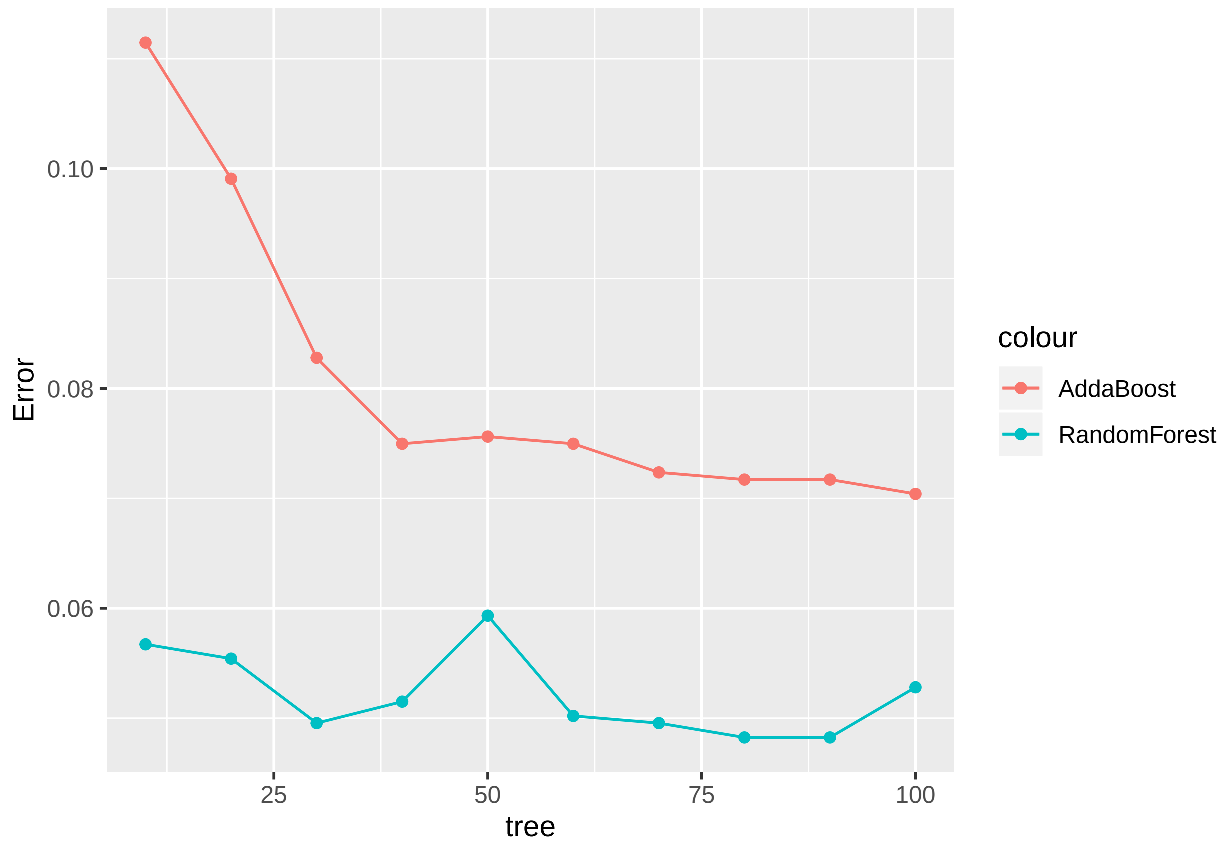
According to the above Addaboost graph, the error rate is very high e.g. around 0.11 when the number of trees is around 15. But as much as the number of trees are increasing error rate is decreasing. From 15 trees to around 40, error has dropped suddenly while after around 40 trees error decrease is monor.

RandomForest



As we can see in the above RandomForest graph, there is a continuous fluctuation in the errors. When the number of trees is 50, the error rate is on peak while the error rate is lowest when the number of trees is around 90.

Comparison



Random forest model deals with the individual trees in a parallel way while the AddaBoost deal with the trees sequentially. Random forest is a large number of trees and then combines at the end of the process by taking averages but AdaBoost combines the trees at the beginning. As we can see in the above graph, as much as the number of trees is increasing error rate is decreasing but the difference is that Addaboost has sudden fall in the error rate while the RandomForest has almost the same. According to the above comparison graph, We can conclude that the RandomFotrest is better than the AdaBoost.

MIXTURE MODELS

In this assignment, we will implement EM algorithm of Bernoulli distribution. As EM algorithm is the process of two steps e.g. E (expected), M (Maximum Likelihood) so we are implementing both steps for 3 different values of guessed components e.g. K=2, K=3, and K=4.

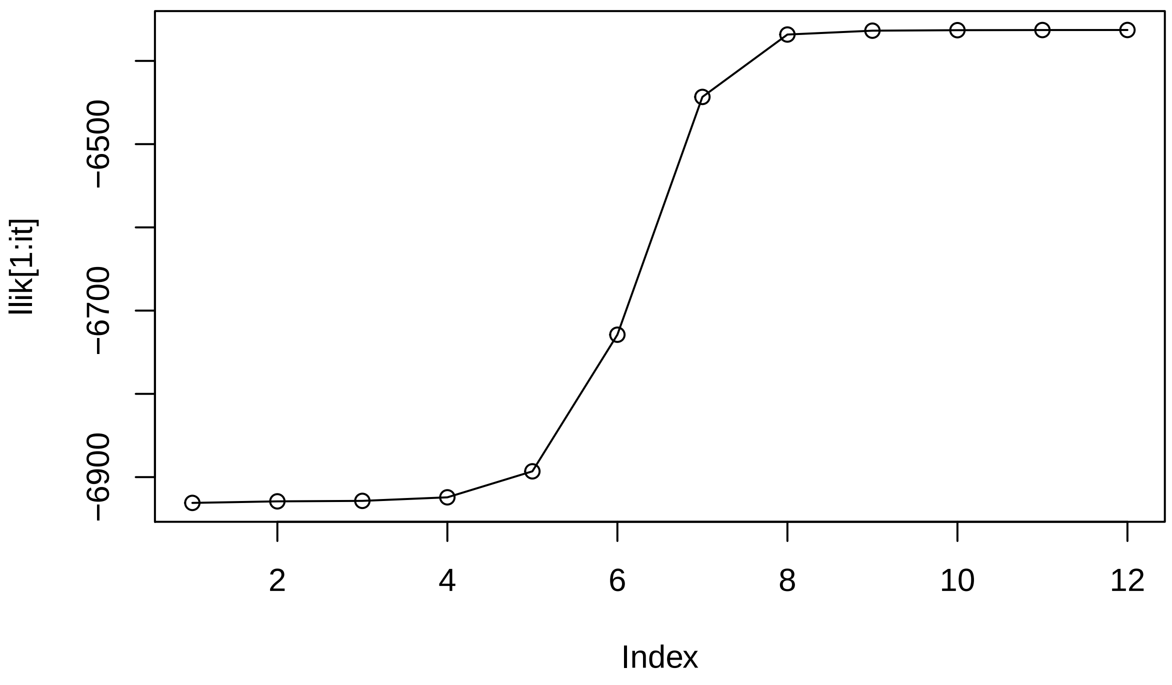
When K=2

```
## iteration:  1 log likelihood:  -6930.975
## iteration:  2 log likelihood:  -6929.125
## iteration:  3 log likelihood:  -6928.562
## iteration:  4 log likelihood:  -6924.281
## iteration:  5 log likelihood:  -6893.055
## iteration:  6 log likelihood:  -6728.948
## iteration:  7 log likelihood:  -6443.28
## iteration:  8 log likelihood:  -6368.318
## iteration:  9 log likelihood:  -6363.734
## iteration:  10 log likelihood:  -6363.109
## iteration:  11 log likelihood:  -6362.947
```

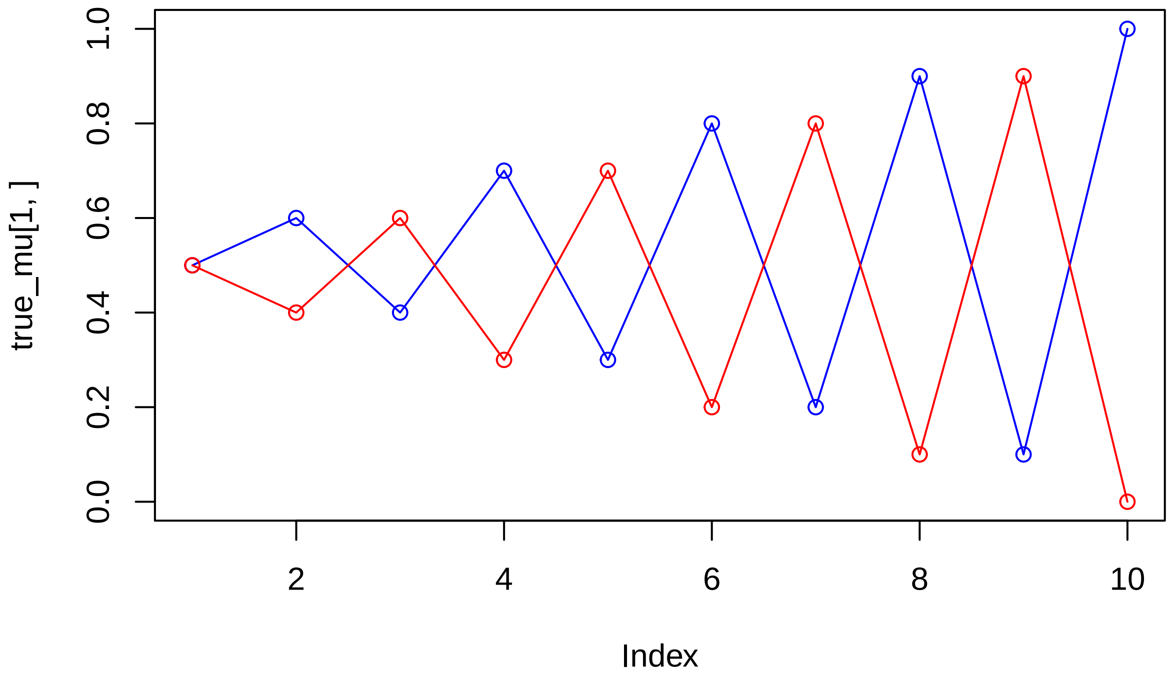
```
## [1] 0.497125 0.502875
```

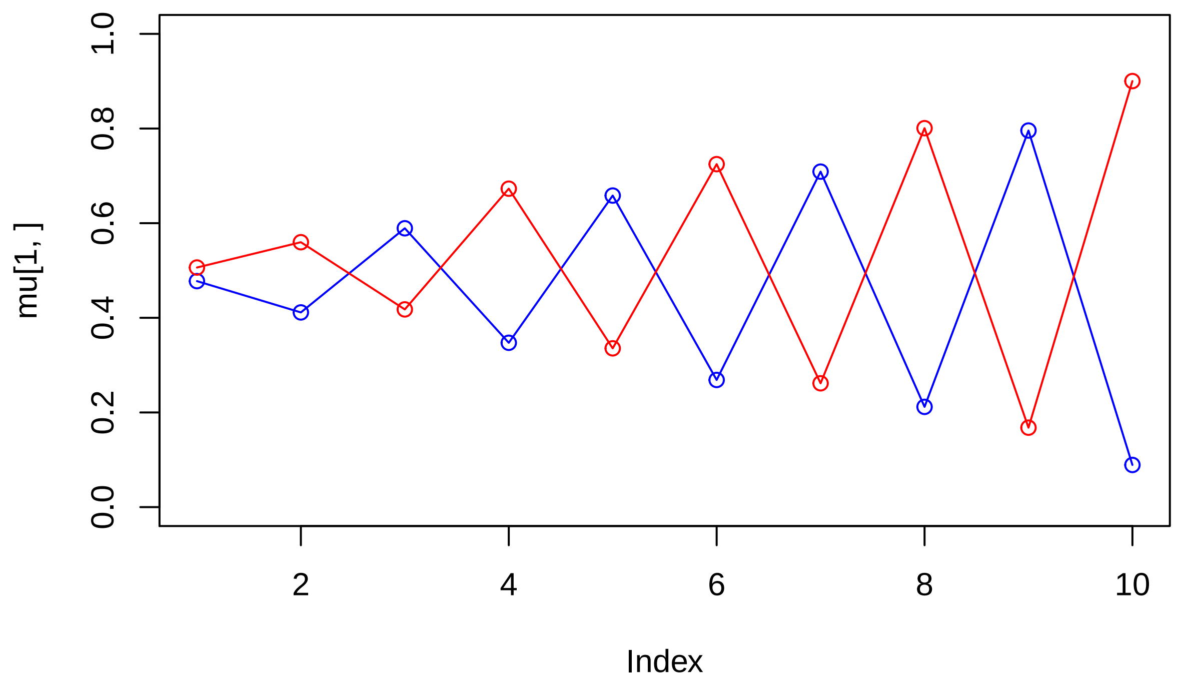
```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4775488 0.4113939 0.5892308 0.3472420 0.6583712 0.2686589 0.7089490
## [2,] 0.5062860 0.5597531 0.4177551 0.6728856 0.3354854 0.7247188 0.2616231
##           [,8]      [,9]      [,10]
## [1,] 0.2118629 0.7957549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

```
## [1] 12
```



When K=2 last iteration is 12 and the value of log likelihood is -6362.8972585.


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



In the above graph we are plotting the updated conditional distributions and the true conditional distributions when K=2.

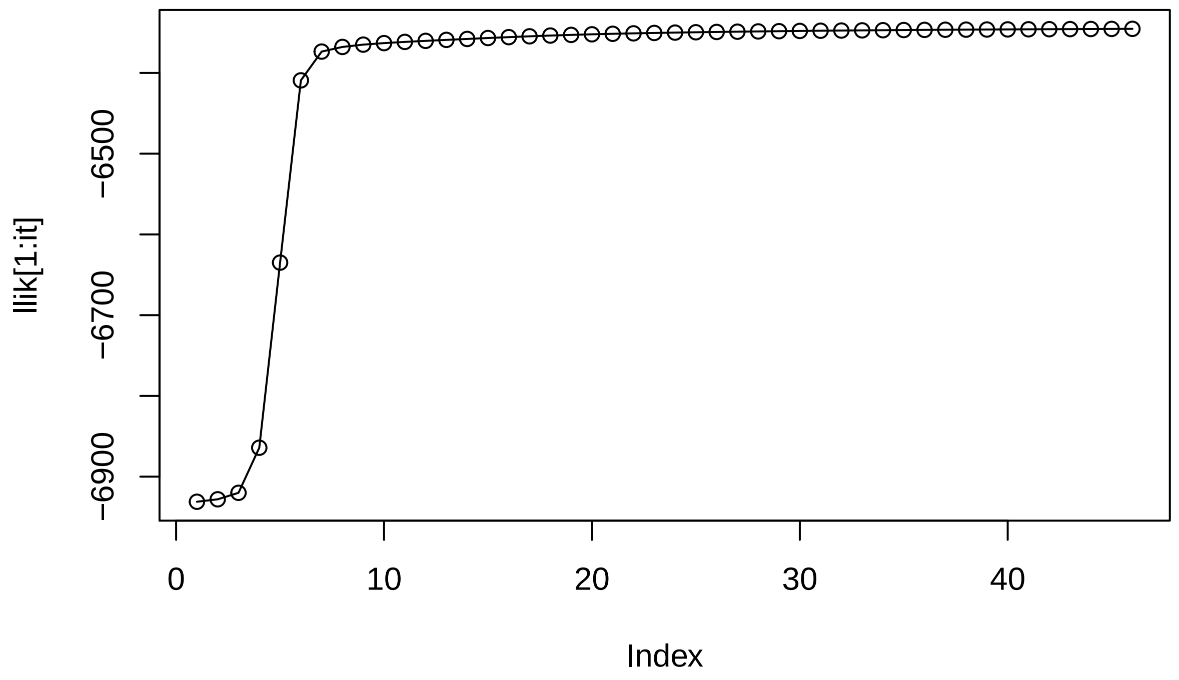
When K=3

```
## iteration:  1 log likelihood:  -6931.064
## iteration:  2 log likelihood:  -6928.051
## iteration:  3 log likelihood:  -6920.026
## iteration:  4 log likelihood:  -6864.176
## iteration:  5 log likelihood:  -6634.916
## iteration:  6 log likelihood:  -6409.234
## iteration:  7 log likelihood:  -6373.593
## iteration:  8 log likelihood:  -6367.833
## iteration:  9 log likelihood:  -6364.983
## iteration:  10 log likelihood:  -6363.074
## iteration:  11 log likelihood:  -6361.594
## iteration:  12 log likelihood:  -6360.309
## iteration:  13 log likelihood:  -6359.103
## iteration:  14 log likelihood:  -6357.93
## iteration:  15 log likelihood:  -6356.786
## iteration:  16 log likelihood:  -6355.689
## iteration:  17 log likelihood:  -6354.668
## iteration:  18 log likelihood:  -6353.742
## iteration:  19 log likelihood:  -6352.92
## iteration:  20 log likelihood:  -6352.199
## iteration:  21 log likelihood:  -6351.567
## iteration:  22 log likelihood:  -6351.011
## iteration:  23 log likelihood:  -6350.515
## iteration:  24 log likelihood:  -6350.069
## iteration:  25 log likelihood:  -6349.661
## iteration:  26 log likelihood:  -6349.286
## iteration:  27 log likelihood:  -6348.938
## iteration:  28 log likelihood:  -6348.616
## iteration:  29 log likelihood:  -6348.315
## iteration:  30 log likelihood:  -6348.036
## iteration:  31 log likelihood:  -6347.776
## iteration:  32 log likelihood:  -6347.534
## iteration:  33 log likelihood:  -6347.308
## iteration:  34 log likelihood:  -6347.099
## iteration:  35 log likelihood:  -6346.904
## iteration:  36 log likelihood:  -6346.722
## iteration:  37 log likelihood:  -6346.553
## iteration:  38 log likelihood:  -6346.394
## iteration:  39 log likelihood:  -6346.246
## iteration:  40 log likelihood:  -6346.107
## iteration:  41 log likelihood:  -6345.977
## iteration:  42 log likelihood:  -6345.854
## iteration:  43 log likelihood:  -6345.739
## iteration:  44 log likelihood:  -6345.63
## iteration:  45 log likelihood:  -6345.528
```

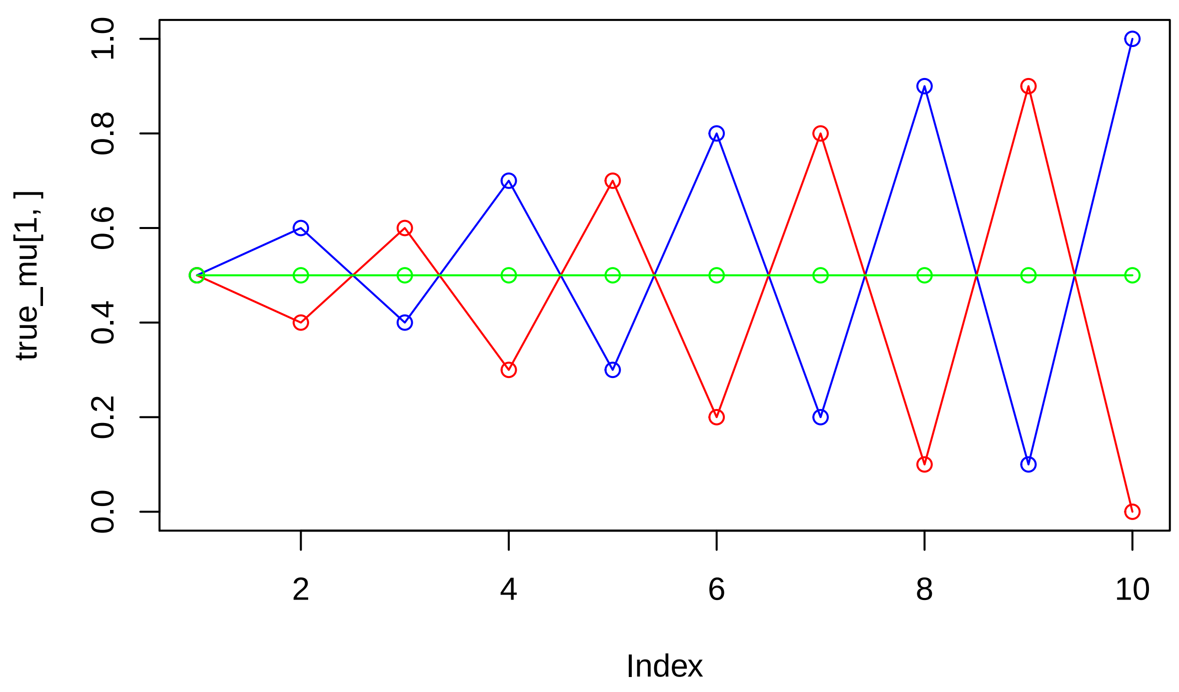
```
## [1] 0.3964172 0.2787080 0.3248749
```

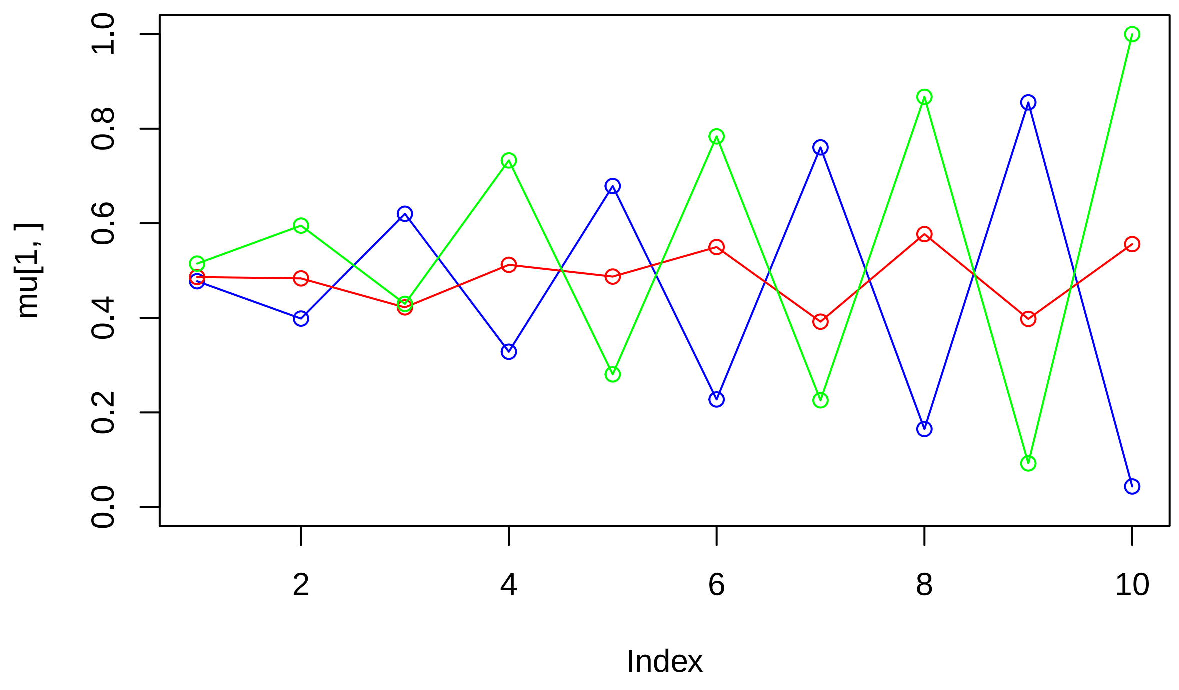
```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4774399 0.3984407 0.6200854 0.3283412 0.6787726 0.2274644 0.7605397
## [2,] 0.4863053 0.4834283 0.4221438 0.5121400 0.4872729 0.5497432 0.3918968
## [3,] 0.5146518 0.5950474 0.4294967 0.7329049 0.2804651 0.7837214 0.2255769
##           [,8]       [,9]      [,10]
## [1,] 0.1649014 0.85568380 0.04340808
## [2,] 0.5771323 0.39774130 0.55592505
## [3,] 0.8673460 0.09215422 0.99992821
```

```
## [1] 46
```



When K=3 last iteration is 46 and the value of log-likelihood is -6345.4309528.


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



In the above graph we are plotting the updated conditional distributions and the true conditional distributions when K=3.

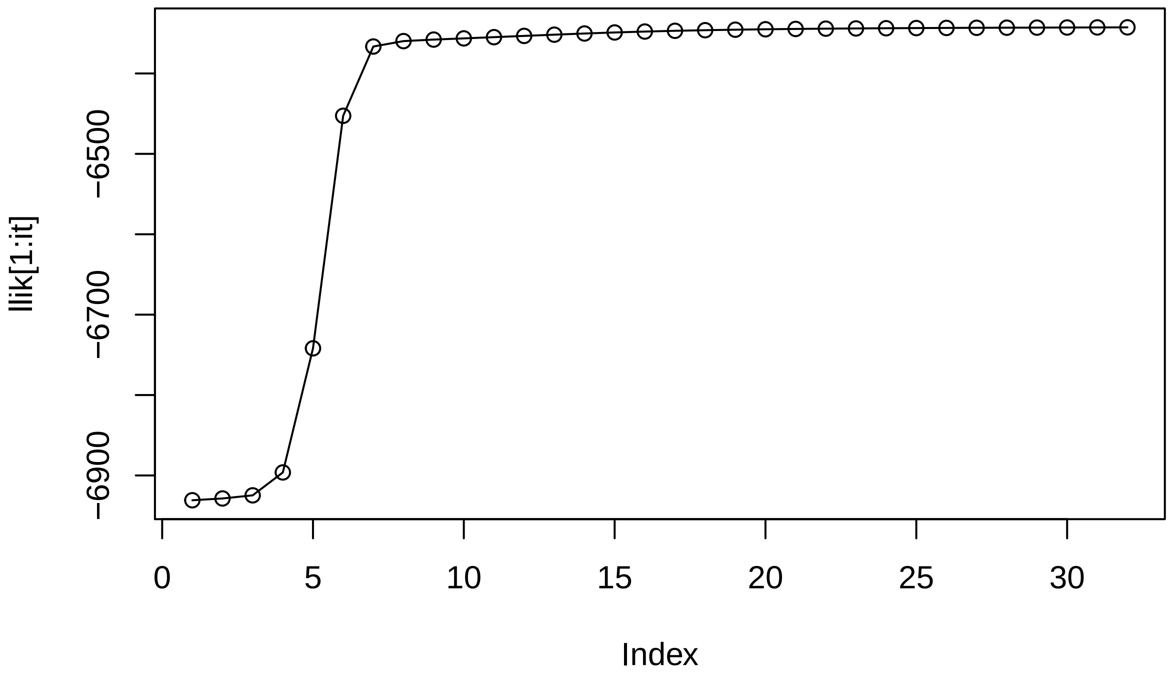
When K=4

```
## iteration:  1 log likelihood:  -6930.838
## iteration:  2 log likelihood:  -6928.641
## iteration:  3 log likelihood:  -6924.748
## iteration:  4 log likelihood:  -6896.25
## iteration:  5 log likelihood:  -6741.896
## iteration:  6 log likelihood:  -6452.658
## iteration:  7 log likelihood:  -6366.493
## iteration:  8 log likelihood:  -6359.764
## iteration:  9 log likelihood:  -6357.876
## iteration:  10 log likelihood:  -6356.372
## iteration:  11 log likelihood:  -6354.86
## iteration:  12 log likelihood:  -6353.31
## iteration:  13 log likelihood:  -6351.776
## iteration:  14 log likelihood:  -6350.33
## iteration:  15 log likelihood:  -6349.03
## iteration:  16 log likelihood:  -6347.908
## iteration:  17 log likelihood:  -6346.968
## iteration:  18 log likelihood:  -6346.196
## iteration:  19 log likelihood:  -6345.566
## iteration:  20 log likelihood:  -6345.055
## iteration:  21 log likelihood:  -6344.637
## iteration:  22 log likelihood:  -6344.293
## iteration:  23 log likelihood:  -6344.008
## iteration:  24 log likelihood:  -6343.768
## iteration:  25 log likelihood:  -6343.563
## iteration:  26 log likelihood:  -6343.387
## iteration:  27 log likelihood:  -6343.233
## iteration:  28 log likelihood:  -6343.097
## iteration:  29 log likelihood:  -6342.975
## iteration:  30 log likelihood:  -6342.864
## iteration:  31 log likelihood:  -6342.762
```

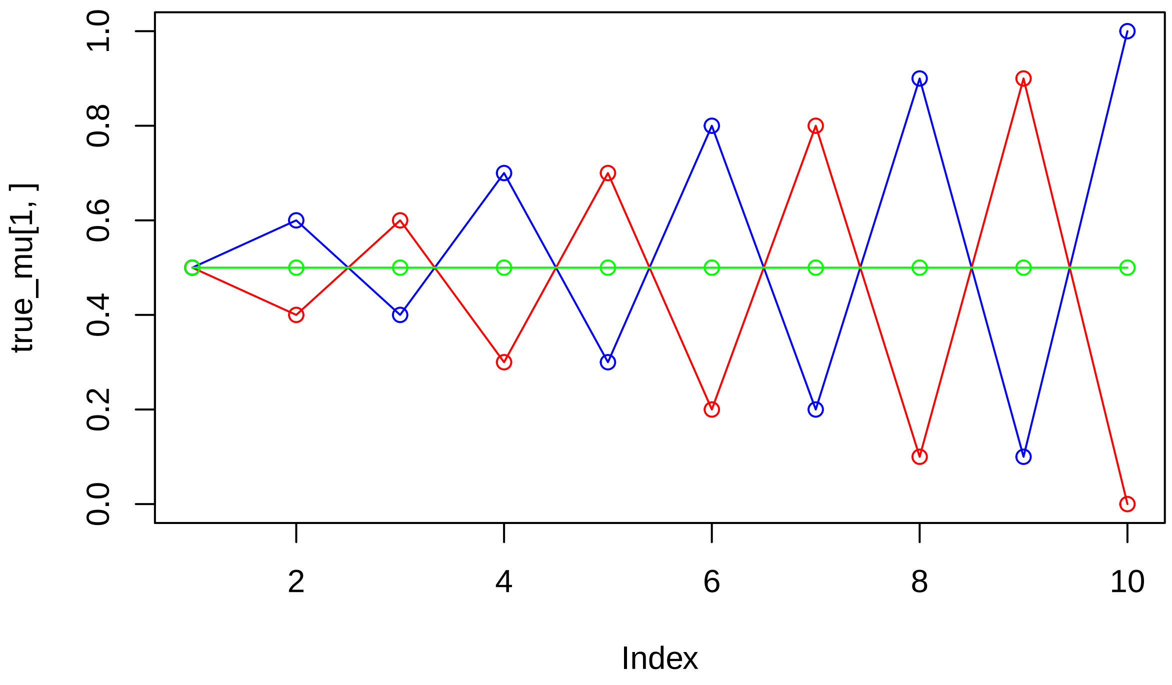
```
## [1] 0.2690190 0.2863050 0.2457246 0.1989515
```

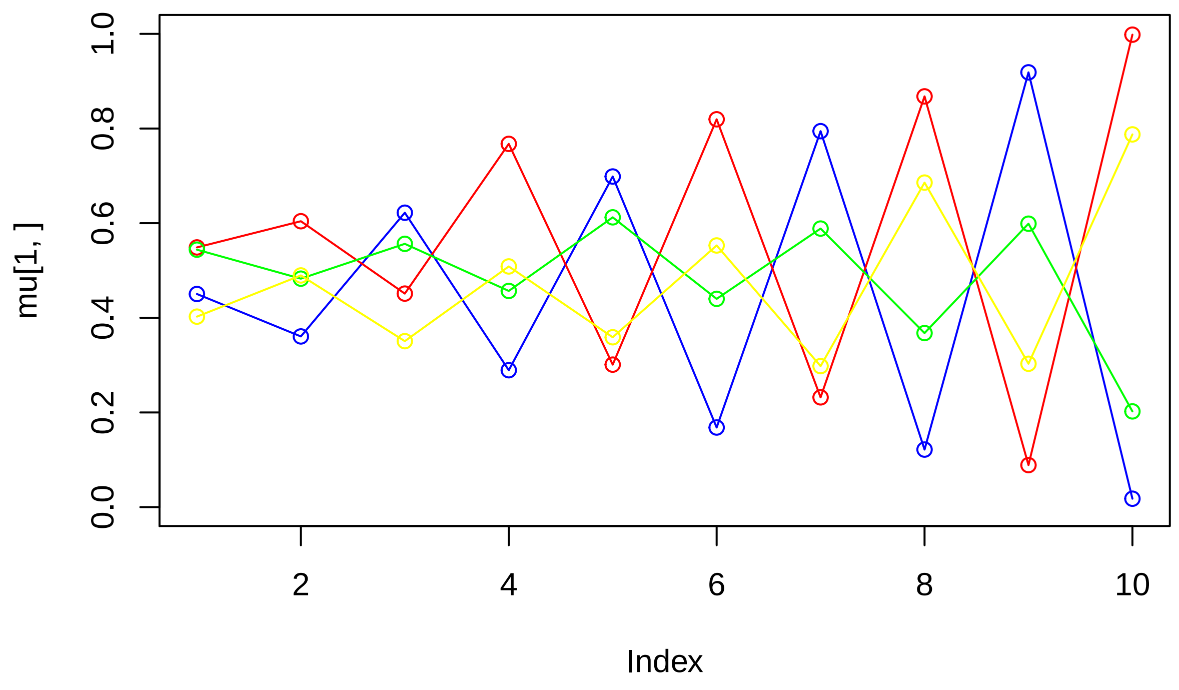
```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4502917 0.3606587 0.6220817 0.2892407 0.6986320 0.1681768 0.7943990
## [2,] 0.5487864 0.6040921 0.4511711 0.7675478 0.3010522 0.8195305 0.2318913
## [3,] 0.5439579 0.4827437 0.5563603 0.4568300 0.6123061 0.4400351 0.5885625
## [4,] 0.4025047 0.4895637 0.3506597 0.5085745 0.3588983 0.5528693 0.2979403
##           [,8]       [,9]      [,10]
## [1,] 0.1215732 0.91870837 0.01774129
## [2,] 0.8679302 0.08877946 0.99833984
## [3,] 0.3677877 0.59890299 0.20227253
## [4,] 0.6857315 0.30292227 0.78760041
```

```
## [1] 32
```



When K=4 last iteration is 32 and the value of log-likelihood is -6342.6675747.





In the above graph we are plotting the updated conditional distributions and the true conditional distributions when K=4.

Conclusion

According to the above different values of log-likelihood for each value of K, we can say that as much as we are increasing K the value of maximum log-likelihood is also increasing. At the end of the iterations, the difference is not significant for current and previous log-likelihood. We can conclude that our current selection of k is quite reasonable.

Code Appendix

```
library(mboost)
library(randomForest)
library(wSVM)
library(ggplot2)
knitr::opts_chunk$set(echo = TRUE)

spdata <- read.csv2("spambase.csv")
spdata$Spam <- as.factor(spdata$Spam)

RNGkind(sample.kind = "Rounding")
set.seed(12345)
```

```r
n = nrow(spdata)
id=sample(1:n, floor(n*2/3))
train=spdata[id,]
test=spdata[-id,]



dt = sapply(seq(from = 10, to = 100, by = 10), function(i){

  ########Boost Train
  boostmodel <- blackboost(Spam~., data=train,
                           family = Binomial(type=c("adaboost")),
                           control = boost_control(mstop = i))

  # boostpredictTrain <- predict(boostmodel,train,type="class")

  # boostErrortrain <- mean(boostpredictTrain != train$Spam) *100


  boostpredictTest <- predict(boostmodel,test,type="class")

  boostConf <- table(boostpredictTest, test$Spam)

  boostErrortest <- 1-sum(diag(boostConf))/sum(boostConf)

  ########RF Train
  randomforestmodeltrain<- randomForest(Spam~., data=train,ntree=i)

  # forestPredictTrain <- predict(randomforestmodeltrain,train ,type="class")

  # forestErrorTrain <- mean(forestPredictTrain != train$Spam)*100


  forestPredictTest <- predict(randomforestmodeltrain,test ,type="class")

  forestError <- table(forestPredictTest,test$Spam)

  forestErrortest <- 1-sum(diag(forestError))/sum(forestError)

  data.frame(boostError = boostErrortest,
             forestError = forestErrortest,tree = i)

})



dt = as.data.frame(t(dt))

df <- as.data.frame(lapply(dt, unlist))


ggplot(df)+ geom_line(aes(x=tree,y=boostError, col="AddaBoost"))+
  geom_point(aes(x=tree,y=boostError, col="AddaBoost"))
```

```r
ggplot(df) + geom_line(aes(x=tree,y=forestError, col="RandomForest"))+
  geom_point(aes(x=tree,y=forestError, col="RandomForest"))

ggplot(df) + geom_line(aes(x=tree,y=boostError, col="AddaBoost"))+
  geom_point(aes(x=tree,y=boostError, col="AddaBoost"))+
  geom_line(aes(x=tree,y=forestError, col="RandomForest"))+
  geom_point(aes(x=tree,y=forestError, col="RandomForest"))+ylab("Error")
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)
# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,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
# 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)
}
#pi
#mu
oldllik = 0
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 fractional component assignments
  # Your code here
  likelihood = vector(length = N)


  list_ln_p_xn_muk_pik<-sapply(1:N, function(nn){
    p_xn_muk_pikv<-c()
    list_p_xd_muk_pik<-sapply(1:K, function(comp,ni=nn){

        p_xn_muk<-prod(mu[comp,]^x[ni,]*(1-mu[comp,])^(1-x[ni,]))
        p_xn_muk_pik<-pi[comp]*p_xn_muk
        p_xn_muk_pikv <<-c(p_xn_muk_pikv,p_xn_muk_pik)
    })

      z[nn,] <<- p_xn_muk_pikv/sum(p_xn_muk_pikv)

    #from the book (9.51) - ln p(X|?,??)
    log(sum(p_xn_muk_pikv)) ## return log likelihood
  })


  llik[it] = sum(list_ln_p_xn_muk_pik)

  if(abs(oldllik - llik[it])<min_change){
    #print("break")
    break

  }else{
    oldllik = llik[it]
  }

  #Log likelihood computation.
  # Your code here
 cat("iteration: ", it, "log likelihood: ", llik[it], "\n")

  # Stop if the lok likelihood has not changed significantly
  # Your code here
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  #print("sad")
  latentSum = colSums(z)

  pi = latentSum/N

  for (i in 1:K){
    for (j in 1:D){
      mu[i,j] = sum(z[,i]*x[,j])/sum(z[,i])
    }
  }
}
 pi
 mu
 it
 plot(llik[1:it], type="o")

plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")

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

```
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
# 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)
}
#pi
#mu
oldllik = 0
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 fractional component assignments
  # Your code here
  likelihood = vector(length = N)


  list_ln_p_xn_muk_pik<-sapply(1:N, function(nn){
    p_xn_muk_pikv<-c()

    list_p_xd_muk_pik<-sapply(1:K, function(comp,ni=nn){

        p_xn_muk<-prod(mu[comp,]^x[ni,]*(1-mu[comp,])^(1-x[ni,]))
        p_xn_muk_pik<-pi[comp]*p_xn_muk
        p_xn_muk_pikv <<-c(p_xn_muk_pikv,p_xn_muk_pik)
    })

      z[nn,] <<- p_xn_muk_pikv/sum(p_xn_muk_pikv)

    #from the book (9.51) - ln p(X|?,??)
    log(sum(p_xn_muk_pikv)) ## return log likelihood
  })


  llik[it] = sum(list_ln_p_xn_muk_pik)

  if(abs(oldllik - llik[it])<min_change){
    #print("break")
    break

  }else{
    oldllik = llik[it]
  }

  #Log likelihood computation.
```

```r
  # Your code here
 cat("iteration: ", it, "log likelihood: ", llik[it], "\n")

  # Stop if the lok likelihood has not changed significantly
  # Your code here
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  #print("sad")
  latentSum = colSums(z)

  pi = latentSum/N

  for (i in 1:K){
    for (j in 1:D){
      mu[i,j] = sum(z[,i]*x[,j])/sum(z[,i])
    }
  }
}
 pi
 mu
 it

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

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")

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")

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
# 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)
}
#pi
#mu
oldllik = 0
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 fractional component assignments
  # Your code here
  likelihood = vector(length = N)


  list_ln_p_xn_muk_pik<-sapply(1:N, function(nn){
    p_xn_muk_pikv<-c()

    list_p_xd_muk_pik<-sapply(1:K, function(comp,ni=nn){

        p_xn_muk<-prod(mu[comp,]^x[ni,]*(1-mu[comp,])^(1-x[ni,]))
        p_xn_muk_pik<-pi[comp]*p_xn_muk
        p_xn_muk_pikv <<-c(p_xn_muk_pikv,p_xn_muk_pik)
    })

      z[nn,] <<- p_xn_muk_pikv/sum(p_xn_muk_pikv)

    #from the book (9.51) - ln p(X|?,??)
    log(sum(p_xn_muk_pikv)) ## return log likelihood
  })


  llik[it] = sum(list_ln_p_xn_muk_pik)

  if(abs(oldllik - llik[it])<min_change){
    #print("break")
    break

  }else{
    oldllik = llik[it]
  }

  #Log likelihood computation.
  # Your code here
 cat("iteration: ", it, "log likelihood: ", llik[it], "\n")

  # Stop if the lok likelihood has not changed significantly
  # Your code here
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  #print("sad")
  latentSum = colSums(z)

  pi = latentSum/N

  for (i in 1:K){
    for (j in 1:D){
      mu[i,j] = sum(z[,i]*x[,j])/sum(z[,i])
    }
  }
}
 pi
 mu
 it
 plot(llik[1:it], type="o")

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")

 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")
```