LAB 1 BLOCK 2: ENSEMBLE METHODS AND MIXTURE MODELS

Biswas Kumar(Liu Id : bisku859)

05/01/2020

Assignment 1. ENSEMBLE METHODS

Your task is to evaluate the performance of Adaboost classification trees and random forests on the spam data. Specifically, provide a plot showing the error rates when the number of trees considered are 10; 20; : : : ; 100. To estimate the error rates, use 2/3 of the data for training and 1/3 as hold-out test data.

Step 1:

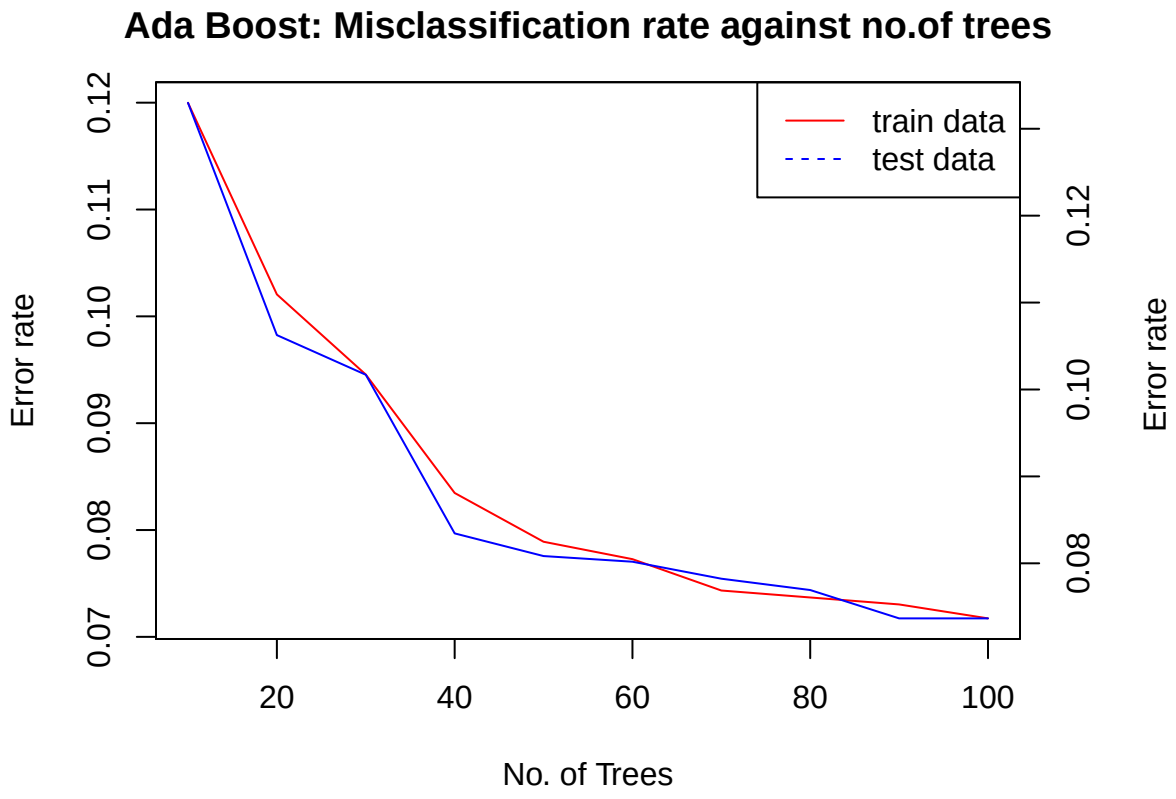
a. Reading the "spambase.csv" file
b. filtering the spam column and storing back as factor
c. Dividing the data set into two parts i.e. 1. Training data (train_set) - 2/3 (66.6%) & 2. Test data(test_set) - balance 1/3 (33.3%)

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-uniform
## 'Rounding' sampler used
```

Step- 2

a. Initializing trees in vector (Initial_trees_vector) for which the error rates shall be calculated and plotted
b. Saving the misclassification rate everytime for loop runs
c. Implementing blackboost function (Adaboost Classification) within for loop
d. Predicting blackboost function output
e. Calculating error rate through standrad classification formula and printing the same
f. Plotting the outcome for visual interpretation

```
##  Misclasification rate  0.1329857
##  Misclasification rate  0.1062581
##  Misclasification rate  0.1016949
##  Misclasification rate  0.08344198
##  Misclasification rate  0.08083442
##  Misclasification rate  0.08018253
##  Misclasification rate  0.07822686
##  Misclasification rate  0.07692308
##  Misclasification rate  0.07366362
##  Misclasification rate  0.07366362
```



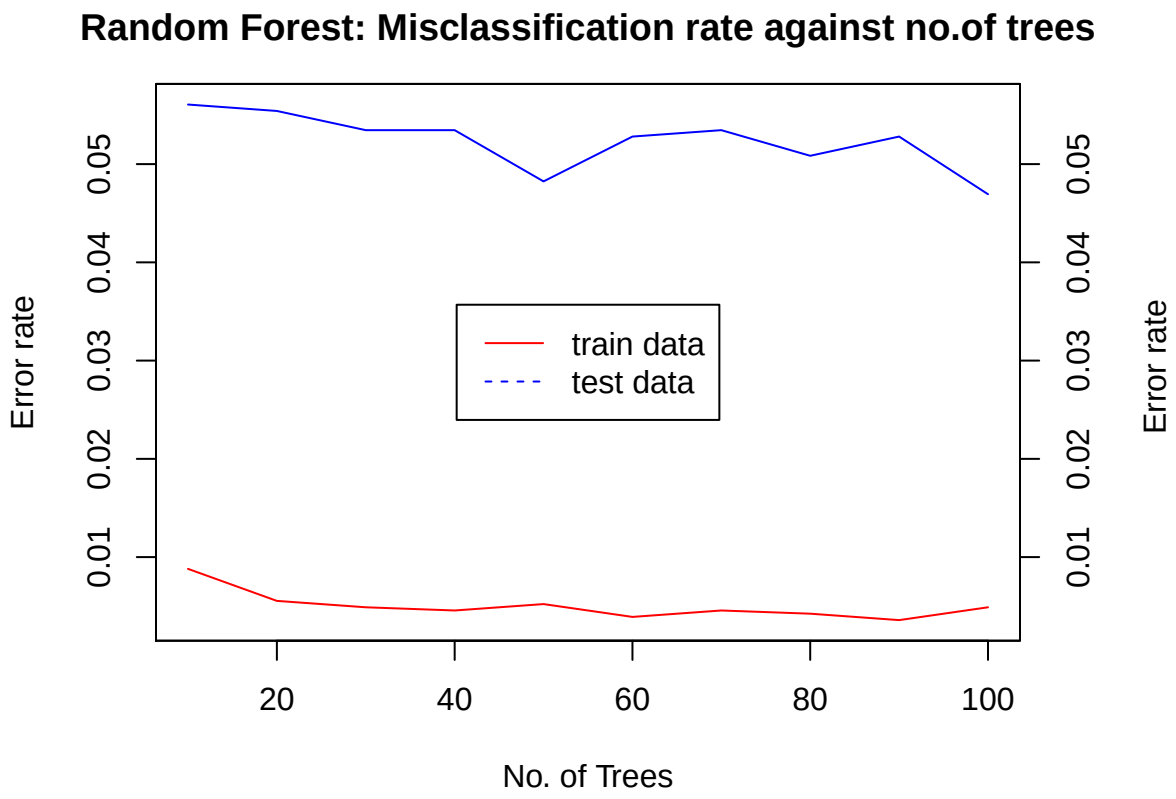
Comment updated (in context with submitted Group report which was right) : While analysing and comparing the chart above, we see that the Misclassification (test data) falls rapidly with increase in Number of trees till 40. It may be noted that the error does not change significantly once it crosses 40. The graph and misclassification table further suggest that it goes down very slowly. Though 100 trees can produce best result but that will be computationally very expensive. It is therefore wise to settle with the range around 50 trees.

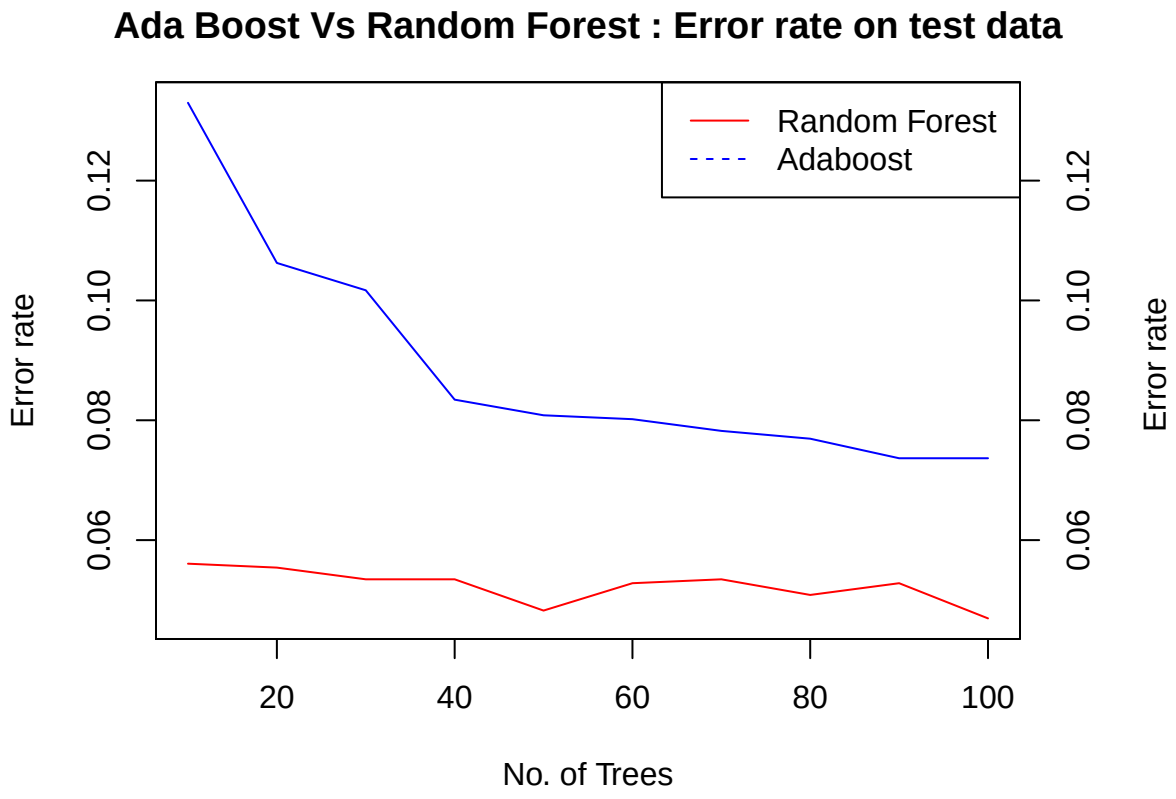
Step- 3

a. Initializing trees in vector (Initial_trees_vector) for which the error rates shall be calculated and plotted
b. Saving the misclassification rate everytime for loop runs
c. Implementing randomForest function within for loop
d. Predicting blackboost function output
e. Calculating error rate through standrad classification formula and printing the same
f. Plotting the outcome for visual interpretation

```
##   Misclasification rate  0.05606258
##   Misclasification rate  0.05541069
##   Misclasification rate  0.05345502
##   Misclasification rate  0.05345502
##   Misclasification rate  0.0482399
##   Misclasification rate  0.05280313
##   Misclasification rate  0.05345502
##   Misclasification rate  0.05084746
##   Misclasification rate  0.05280313
```

```
##   Misclasification rate  0.04693611
```





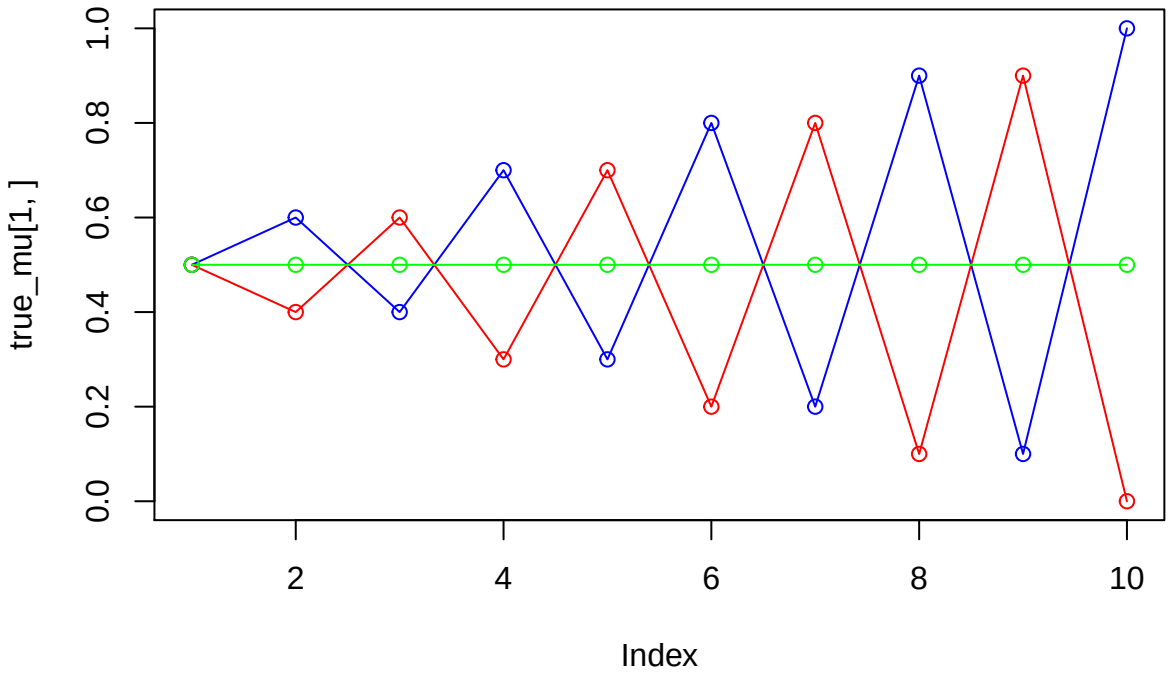
Simlarly,upon running the random forest model plot, we find that test misclassification does vary a bit and decline with increasing number of trees and tend to find low support around 50.The error rate increses thenafter while the lowest is around 90. Since, trees =90 will be computationally more expensive with negligible benefit in result,it is wise to go ahead with optimum number of trees, which is around 40-50.

Lastly, Looking at the comparative chart(Adaboost Vs Random Forest on test data), the Random forest misclassification gives lower error rate at all time when compared with Adaboost. It is therefore, we can consider Random Forest as the better model. The optimum number of trees looks to be around 40-50 for best result.

(Plesae note as advised, my observation has been updated as per group report as its conclusion was better. My earlier submission only explained the graph movement but did not reached to optimum conclusion. Also, I have included combined Adaboost & Random Forest graph which was not there in group report for easy comparision).

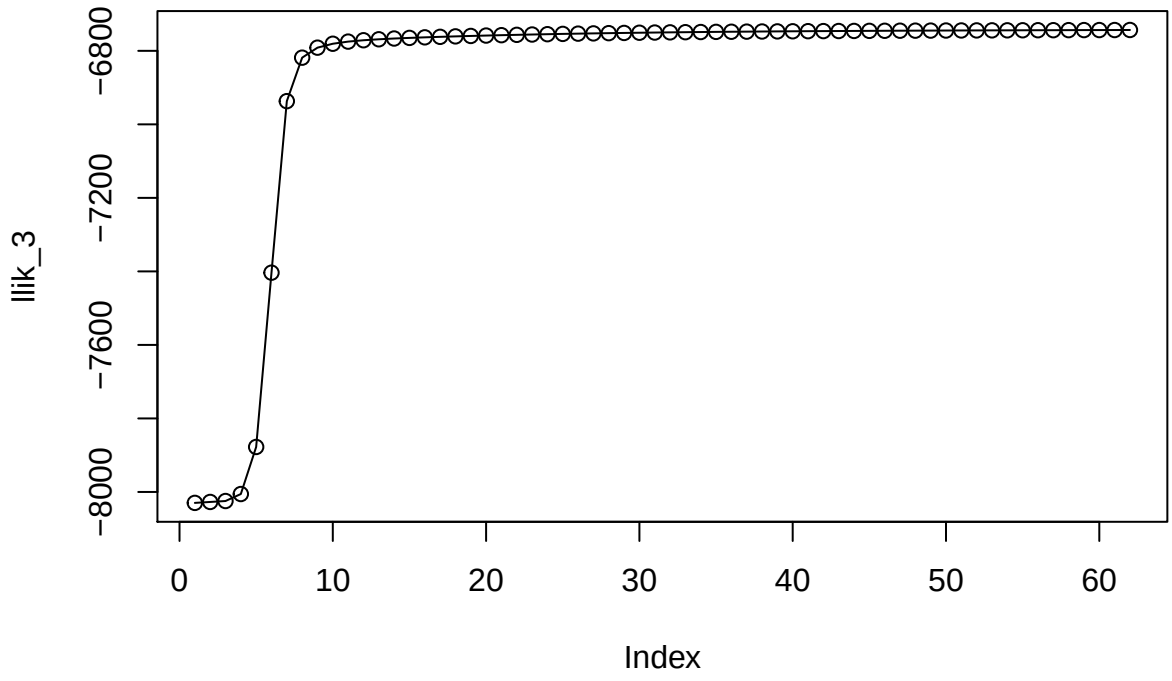
Assignment 2.MIXTURE MODELS

Your task is to implement the EM algorithm for mixtures of multivariate Benoulli distributions. Please use the template in the next page to solve the assignment. Then, use your implementation to show what happens when your mixture models has too few and too many components,i.e. set K = 2; 3; 4 and compare results. Please provide a short explanation as well.



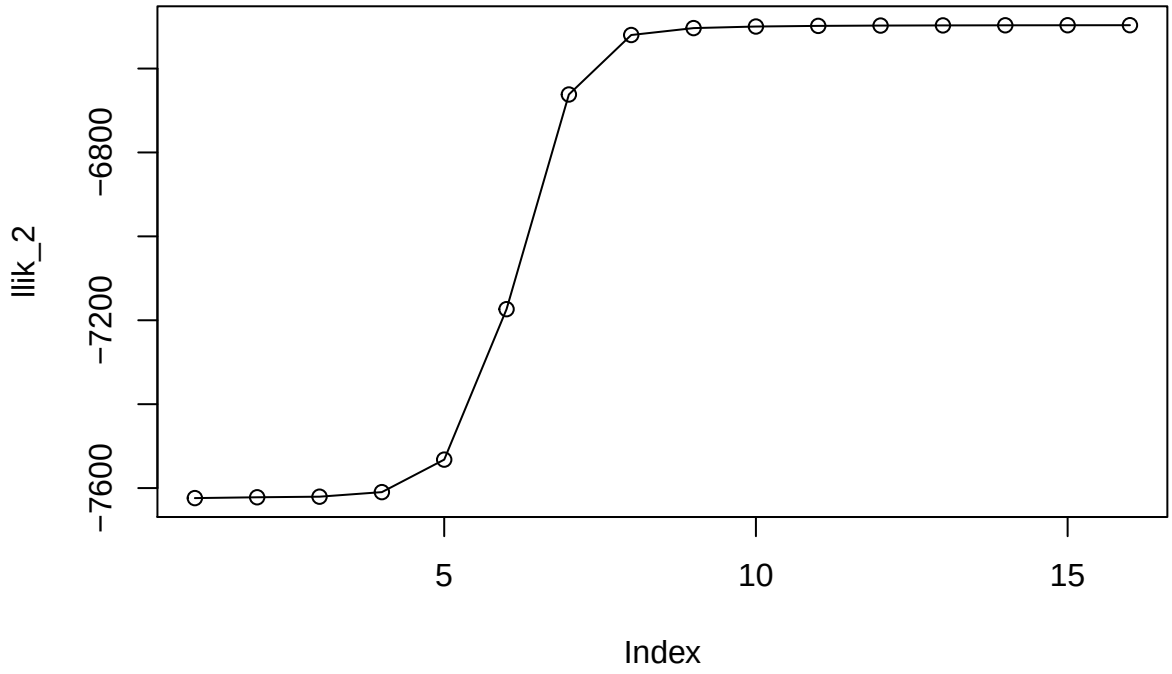

```
## Random initialized value of pi and mu :

## [1] 0.3326090 0.3336558 0.3337352

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4939877 0.4935375 0.5042511 0.5040286 0.4987810 0.5012754 0.4971036
## [2,] 0.4993719 0.5088453 0.5068730 0.5016720 0.4929275 0.5077146 0.5095075
## [3,] 0.4975302 0.5077926 0.4939841 0.5059821 0.5063490 0.5041462 0.4929400
##           [,8]      [,9]     [,10]
## [1,] 0.4982144 0.4987654 0.4929075
## [2,] 0.4924574 0.4992470 0.5008651
## [3,] 0.4992362 0.4943482 0.4903974

##
##  Estimated value of pi

## [1] 0.3259592 0.3044579 0.3695828

##
##  Estimated value of mu:

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4737193 0.3817120 0.6288021 0.3086143 0.6943731 0.1980896 0.7879447
## [2,] 0.4909874 0.4793213 0.4691560 0.4791793 0.5329895 0.4928830 0.4643990
```

```
## [3,] 0.5089571 0.5834802 0.4199272 0.7157107 0.2905703 0.7667258 0.2320784
##           [,8]      [,9]      [,10]
## [1,] 0.1349651 0.8912534 0.01937869
## [2,] 0.4902682 0.4922194 0.39798407
## [3,] 0.8516111 0.1072226 0.99981353

##   number of iterations for K=3  62
```



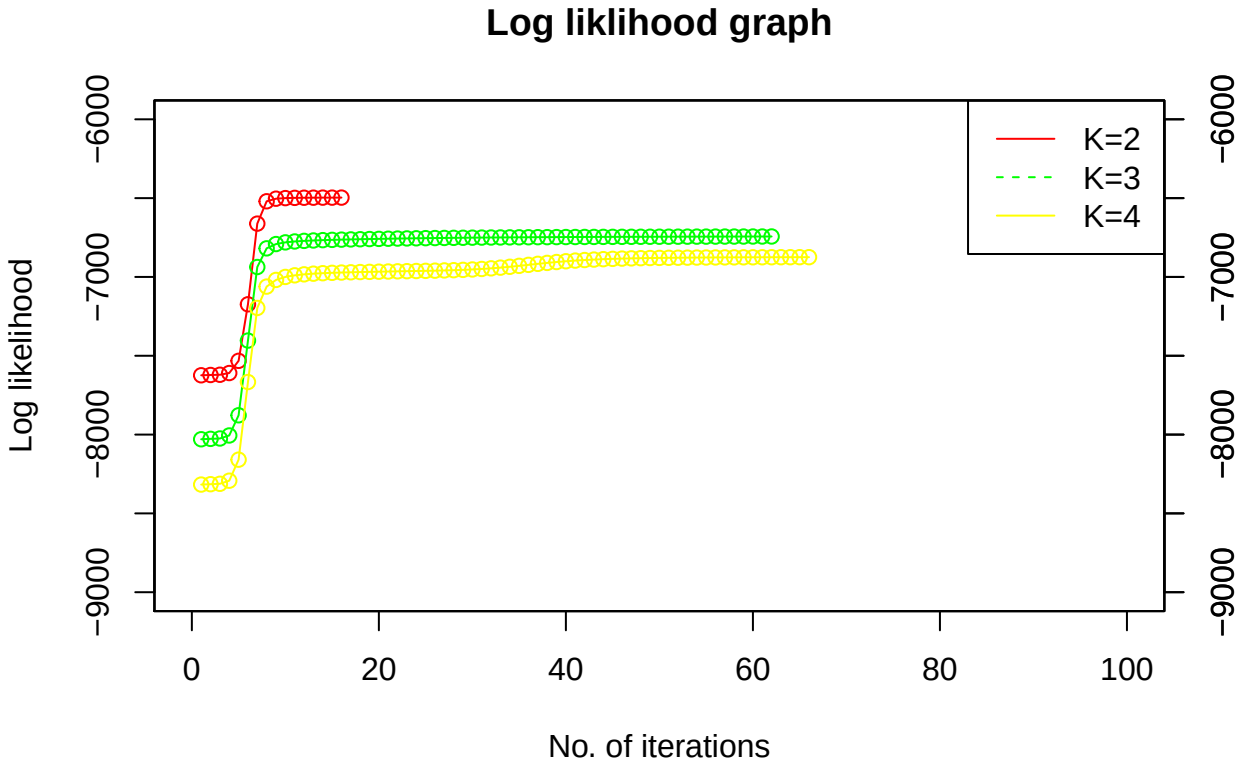

```
## Random initialized value of pi and mu

## [1] 0.4992145 0.5007855

##           [,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

##
##  Estimated value of pi

## [1] 0.4981919 0.5018081
```

```
##
##  Estimated value of mu:

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4777136 0.4113065 0.5888317 0.3477062 0.6580979 0.2698303 0.7078467
## [2,] 0.5061835 0.5601552 0.4177868 0.6731171 0.3350702 0.7245255 0.2617664
##           [,8]      [,9]     [,10]
## [1,] 0.2134061 0.7941168 0.0875152
## [2,] 0.8004711 0.1681467 0.9035340

##   number of iterations for K=2  16
```




```
## Random initialized value of pi and mu

## [1] 0.2491838 0.2499680 0.2500275 0.2508207

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4935375 0.5042511 0.5040286 0.4987810 0.5012754 0.4971036 0.4982144
## [2,] 0.5088453 0.5068730 0.5016720 0.4929275 0.5077146 0.5095075 0.4924574
## [3,] 0.5077926 0.4939841 0.5059821 0.5063490 0.5041462 0.4929400 0.4992362
## [4,] 0.4909320 0.4982342 0.4961948 0.4967226 0.4984220 0.4960055 0.4997165
##           [,8]      [,9]     [,10]
## [1,] 0.4987654 0.4929075 0.4993719
## [2,] 0.4992470 0.5008651 0.4975302
## [3,] 0.4943482 0.4903974 0.5089045
## [4,] 0.5090205 0.5057927 0.4947660
```

```
##
##  Estimated value of pi

## [1] 0.1614155 0.1383613 0.3609912 0.3392319

##
##  Estimated value of mu:

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4372908 0.5716691 0.6230114 0.4717152 0.4251232 0.2940734 0.4797605
## [2,] 0.5381955 0.3913346 0.2971686 0.5062848 0.6375272 0.7107583 0.4202372
## [3,] 0.5102441 0.5846281 0.4200464 0.7178717 0.2850900 0.7735833 0.2327656
## [4,] 0.4797762 0.3788928 0.6181216 0.3114748 0.6964392 0.2149967 0.7793732
##           [,8]      [,9]       [,10]
## [1,] 0.4812185 0.5945364 0.500815206
## [2,] 0.5246082 0.3534161 0.384513179
## [3,] 0.8546627 0.1022323 0.999999734
## [4,] 0.1450708 0.8791286 0.005800712

##   number of iterations for K=4  66
```



Looking at the plot, we see that the highest likelihood is attained when the number of components is least (k=2) whereas the lowest liklihood is given when the number of component is high (k=4).Higher liklihood corresponds to better classification of data for model.

Moreover, the selection is further attributed by number of iterations which involves computing power. More the component, higher the complexity which leads to higher consumption of computing power.K2 has 16 while K4 has 66 iteration.

Unless it is highlighted as priority between higher likelihood or complexity, it is generally advisable to settle for optimum component which suffice the need for the solution giving optimum log liklihood.

In short, a balanced fit (optimum selection) is normally followed unless specified otherwise.

Code Appendix

```
knitr::opts_chunk$set(echo = TRUE)
tidy.opts = list(width.cutoff = 60)
library(readxl)
library(mboost)
library(randomForest)
sp <- read.csv2("spambase.csv")
sp$Spam <- as.factor(sp$Spam)
#setting dimension of spam
dimension=dim(sp)[1]
#version control as specified by faculty
RNGversion('3.5.1')
set.seed(12345)
#setting up 2/3rd of train data and balance 1/3 for test data as specified in course ppt
indexset=sample(1:dimension, floor(dimension*(2/3)))
train_set=sp[indexset,]
test_set=sp[-indexset,]
Initial_trees_vector<- c(10,20,30,40,50,60,70,80,90,100)
Misclassification_error_train_ada <- c(length=10)
for (i in 1:10){
m_boost <- blackboost(formula = Spam~.,data = train_set,
                      family = AdaExp(), control = boost_control(mstop = Initial_trees_vector[i]))
boost_projection <- predict(m_boost,newdata = train_set, type="class")
#boost_projection
plan=table(train_set$Spam, boost_projection)
#True Positive, True Negative, False Positive & False Negative calculation
TrueP=plan[2,2];TrueN=plan[1,1];FalseN=plan[2,1];FalseP=plan[1,2]
#cat("\t Misclasification rate ",(FalseP + FalseN)/ sum(plan), "\n")
Misclassification_error_train_ada[i] <- (FalseP + FalseN)/ sum(plan)
}
# Misclassification_error_train , comment : not printing it anymore

# updated comment :Deleted plot from here to combine in plot below for easy
# visualization and comparision

#performing for test data (test_set)

Misclassification_error_test_ada <- c(length=10)
for (i in 1:10){
  m_boost <- blackboost(formula = Spam~.,data = train_set,
                        family = AdaExp(), control = boost_control(mstop = Initial_trees_vector[i]))
  # wrongly done above  on test data earlier
```

```
boost_projection_test <- predict(m_boost,newdata = test_set, type="class")

plan=table(test_set$Spam, boost_projection_test);plan
#True Positive, True Negative, False Positive & False Negative calculation
TrueP=plan[2,2];TrueN=plan[1,1];FalseN=plan[2,1];FalseP=plan[1,2]
cat("\tMisclasification rate ",(FalseP + FalseN)/ sum(plan),"\n")
Misclassification_error_test_ada[i] <- (FalseP + FalseN)/ sum(plan)
}
#Misclassification_error_test comment : not printing it anymore

# Comment : Updating the graph below  while including
# both train and test misclassification

par(mar = c(5, 5, 3, 5)) #size of plot
#plot(Initial_trees_vector,Misclassification_error_train_ada,xlab)
plot(x=Initial_trees_vector,y=Misclassification_error_train_ada,
     xlab = "No. of Trees",ylab = "Error rate",type="l",col="red",
     main = "Ada Boost: Misclassification rate against no.of trees")

par(new = TRUE) # for combining two plots
plot(Misclassification_error_test_ada,col="blue",type = "l",xaxt = "n",
     yaxt = "n",ylab = "", xlab = "")
axis(side = 4)
mtext("Error rate ", side = 4, line = 3);
legend("topright", c("train data", "test data"),
       col = c("red", "blue"), lty = c(1, 2)) # Indexing the plot at top right position


Initial_trees_vector<- c(10,20,30,40,50,60,70,80,90,100)
Misclassification_error_train_randomforest <- c(length=10)
for (i in 1:10){
r_forest<- randomForest(formula = Spam~.,ntree=Initial_trees_vector[i],data = train_set)

Randomforest_projection <- predict(r_forest,newdata = train_set, type="class")

plan=table(train_set$Spam, Randomforest_projection)
#True Positive, True Negative, False Positive & False Negative calculation
TrueP=plan[2,2];TrueN=plan[1,1];FalseN=plan[2,1];FalseP=plan[1,2]
#cat("\t Misclasification rate ",(FalseP + FalseN)/ sum(plan),"\n")    ...not printing anymore
Misclassification_error_train_randomforest[i] <- (FalseP + FalseN)/ sum(plan)
}

#performing for test data (test_set)

Misclassification_error_test_randomforest <- c(length=10)
for (i in 1:10){
r_forest<- randomForest(formula = Spam~.,
                        ntree=Initial_trees_vector[i],data = train_set, family = binomial)

Randomforest_projection <- predict(r_forest,newdata = test_set, type="class")

plan=table(test_set$Spam, Randomforest_projection);plan
#True Positive, True Negative, False Positive & False Negative calculation
```

```
TrueP=plan[2,2];TrueN=plan[1,1];FalseN=plan[2,1];FalseP=plan[1,2]
cat("\t Misclasification rate ",(FalseP + FalseN)/ sum(plan),"\n")
Misclassification_error_test_randomforest[i] <- (FalseP + FalseN)/ sum(plan)
}

# Comment : Updating the graph below  while including both train and test misclassification

par(mar = c(5, 5, 3, 5)) #size of plot
plot(Initial_trees_vector, Misclassification_error_train_randomforest
     ,xlab = "No. of Trees",ylab = "Error rate",
     type="l",col="red",main = "Random Forest: Misclassification rate against no.of trees",
     ylim = range(c( Misclassification_error_train_randomforest,
                     Misclassification_error_test_randomforest)))
par(new = TRUE) # for combining two plots
plot(Initial_trees_vector,Misclassification_error_test_randomforest ,
     col="blue",type = "l",xaxt = "n", yaxt = "n",ylab = "", xlab = "",
     ylim = range(c( Misclassification_error_train_randomforest,
                     Misclassification_error_test_randomforest)))
axis(side = 4)
mtext("Error rate ", side = 4, line = 3);
legend("center", c("train data", "test data"),
       col = c("red", "blue"), lty = c(1, 2)) # Indexing the plot at top right position

# Comparision of Adaboost and Random forest on the same chart :

par(mar = c(5, 5, 3, 5)) #size of plot
plot(Initial_trees_vector, Misclassification_error_test_randomforest ,
     xlab = "No. of Trees",ylab = "Error rate",type="l",
     col="red",main = "Ada Boost Vs Random Forest : Error rate on test data",
     ylim = range(c(Misclassification_error_test_randomforest,
                    Misclassification_error_test_ada)))
par(new = TRUE) # for combining two plots
plot(Misclassification_error_test_ada ,col="blue",type = "l",xaxt = "n",
     yaxt = "n",ylab = "", xlab = "",
     ylim = range(c(Misclassification_error_test_randomforest,
                    Misclassification_error_test_ada)))
axis(side = 4)
mtext("Error rate ", side = 4, line = 3);
legend("topright", c("Random Forest ", "Adaboost"),
       col = c("red", "blue"), lty = c(1, 2)) # Indexing the plot at top right position


  # For K= 3
  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)
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(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<-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)
}
cat("Random initialized value of pi and mu :\n")
pi
mu

for(it in 1:max_it) {
  # not printing below plots on report
#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

  # Mixture distribution is given by :
  # mixture_matrix p(x|mu,pi) = (pi[1,k]* p(x|mu[1,k]) ...over K iterations
  # whereas p(x|mu[1,k])= (mu[1,k]^x) (1- mu(1,k)^(1-x)...summation from 1,D
  n=1
  while (n <= N) {  # N=1000
    # Empty matrix (probability of x) ,
    # parameters :(column 1:K = mixture_matrix_given_k; column K+1 = p(mixture_matrix|all k)
    mixture_matrix = matrix(data = c(rep(1,K), 0), nrow = 1, ncol = K+1)
    #  step 1: Numerator: probability of x when k is given, p(x|k) as per Mixture distribution
    #  step 2: Denominator: probability of x when k is given, p(x|all k)
    #  step 3: z is given by Numerator/ Denominator
    k=1
```

```
  while(k<=K) {   # K=3
    #  step 1
    d=1
    while(d<=D){  # D= 10
      # multiplication of matrices
      mixture_matrix[1,k] = prod(mixture_matrix[1,k],
                                 (mu[k,d]^x[n,d]),(1-mu[k,d])^(1-x[n,d]))
      d=d+1
    }
    mixture_matrix[1,k] = mixture_matrix[1,k] * pi[k] # multiplying with weights
    # step 2
    mixture_matrix[1,K+1] = mixture_matrix[1,K+1] + mixture_matrix[1,k]
    k=k+1
  }
  # step 3:
  for (k in 1:K) {
    # Nmerator/Denominator
    z[n,k] = mixture_matrix[1,k] / mixture_matrix[1,K+1]

  }
  n=n+1
}

# Log likelihood computation
# Your code here

# implemeting of Log likelihood of Bernoulli mixture

n=1 # initializing n for N loop
while (n<=N) {    # summation due to logarithm
  k=1
  while(k<=K) {   # summation due to mixture
    # initializing log_likelihood as 0 to update values from loop below
    log_likelihood = 0
    d=1
    while(d<=D) {

      log_likelihood = log_likelihood + x[n,d] *
        log(mu[k,d]) + (1-x[n,d]) * log(1-mu[k,d])
      d=d+1
    }
    llik[it] = llik[it] + z[n,k] * (log(pi[k]) +
                                      log_likelihood)  # iteration it in 1: 100(max_it)
    k=k+1     # incrementing k for while loop
  }
  n=n+1       # incrementing n for while loop
}
#not printing it anymore
#cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
#flush.console()
# Stop if the log likelihood has not changed significantly
# Your code here
```

```
   # if iteration is more than one  and has not changed significantly
   # i.e less than min_change of 0.1, then it should stop
    if(it != 1){
        if (abs(llik[it] - llik[it-1]) < min_change)
      {break}
    }

   # M-step: ML parameter estimation from the data and fractional component assignments
   # Your code here
   # M step : We maximize the expected complete data loglikelihood wrt parameters mu[k] and pi
   # parameter estimation: pi
   pi=colMeans(z)    # mean of columns
   # parameter estimation :mu
   k=1
   while(k<=K) {
     mu[k,] = 0  # initialized mu to get value from loop
     n=1
     while(n<=N) {
       mu[k,] = mu[k,] + x[n,] * z[n,k]
       n=n+1
     }
     mu[k,] = mu[k,] / sum(z[,k])
     k=k+1
   }
}
   pi_3<-pi
   cat("\n Estimated value of pi \n")
   pi_3
   mu_3<-mu
   cat("\n Estimated value of mu: \n")
   mu_3
   llik_3<-llik[1:it]
   cat("\t number of iterations for K=3 ",length(llik_3),"\n")
   plot(llik_3,type = "o")


  # For K= 2
 set.seed(1234567890)



 # 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)
}
cat("Random initialized value of pi and mu \n")
pi
mu

for(it in 1:max_it) {
  # not printing below plots on report
#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

  # Mixture distribution is given by :
  # mixture_matrix p(x|mu,pi) = (pi[1,k]* p(x|mu[1,k]) ...over K iterations
  # whereas p(x|mu[1,k])= (mu[1,k]^x) (1- mu(1,k)^(1-x)...summation from 1,D
  n=1
  while (n <= N) {  # N=1000
    # Empty matrix (probability of x) ,
    #parameters :(column 1:K = mixture_matrix_given_k; column K+1 = p(mixture_matrix|all k)
    mixture_matrix = matrix(data = c(rep(1,K), 0), nrow = 1, ncol = K+1)
    #  step 1 : Numerator: probability of x when k is given, p(x|k) as per Mixture distribution
    #  step 2: Denominator: probability of x when k is given, p(x|all k)
    #  step 3: z is given by Numerator/ Denominator
    k=1
    while(k<=K) {
      #  step 1
      d=1
      while(d<=D){  # D= 10
        #multiplication of matrices
      mixture_matrix[1,k] = prod(mixture_matrix[1,k],
                                 (mu[k,d]^x[n,d]),(1-mu[k,d])^(1-x[n,d]))
        d=d+1
      }
      mixture_matrix[1,k] = mixture_matrix[1,k] * pi[k] # multiplying with weights
      # step 2
      mixture_matrix[1,K+1] = mixture_matrix[1,K+1] + mixture_matrix[1,k]
      k=k+1
    }
    # step 3:
    for (k in 1:K) {
      # Nmerator/Denominator
      z[n,k] = mixture_matrix[1,k] / mixture_matrix[1,K+1]

    }
    n=n+1
```

```
}

# Log likelihood computation
# Your code here

# implemeting of Log likelihood of Bernoulli mixture

n=1 # initializing n for N loop
while (n<=N) {    # summation due to logarithm
  k=1
  while(k<=K) {   # summation due to mixture
    # initializing log_likelihood as 0 to update values from loop below
    log_likelihood = 0
    d=1
    while(d<=D) {

      log_likelihood = log_likelihood +
        x[n,d] * log(mu[k,d]) + (1-x[n,d]) * log(1-mu[k,d])
      d=d+1
    }
    llik[it] = llik[it] + z[n,k] * (log(pi[k]) + log_likelihood) #iteration it in 1: 100(max_it)
    k=k+1     # incrementing k for while loop
  }
  n=n+1       # incrementing n for while loop
}
#not printing it anymore
#cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the log likelihood has not changed significantly
# Your code here

# if iteration is more than one  and has not changed significantly
# i.e less than min_change of 0.1, then it should stop
if(it != 1){
     if (abs(llik[it] - llik[it-1]) < min_change)
   {break}
 }

# M-step: ML parameter estimation from the data and fractional component assignments
# Your code here
# parameter estimation: pi
pi=colMeans(z)   # mean of columns
# parameter estimation :mu
k=1
while(k<=K) {
  mu[k,] = 0  # initialized mu to get value from loop
  n=1
  while(n<=N) {
    mu[k,] = mu[k,] + x[n,] * z[n,k]
    n=n+1
  }
  mu[k,] = mu[k,] / sum(z[,k])
  k=k+1
```

```
   }
}
   pi_2<-pi
   cat("\n Estimated value of pi \n")
   pi_2
   mu_2<-mu
   cat("\n Estimated value of mu: \n")
   mu_2

   llik_2<-llik[1:it]
   cat("\t number of iterations for K=2 ",length(llik_2),"\n")
   plot(llik_2,type = "o")

#_____________________________________________________________________________________________
    # For K= 4
set.seed(1234567890)


# 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=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)
}
cat("Random initialized value of pi and mu ")
pi
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")
#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

  # Mixture distribution is given by :
  # mixture_matrix p(x|mu,pi) = (pi[1,k]* p(x|mu[1,k]) ...over K iterations
  # whereas p(x|mu[1,k])= (mu[1,k]^x) (1- mu(1,k)^(1-x)...summation from 1,D
```

```
n=1
while (n <= N) {  # N=1000
  # Empty matrix (probability of x) ,
  # parameters :(column 1:K = mixture_matrix_given_k; column K+1 = p(mixture_matrix|all k)
  mixture_matrix = matrix(data = c(rep(1,K), 0), nrow = 1, ncol = K+1)
  #  step 1 : Numerator: probability of x when k is given,p(x|k) as per Mixture distribution
  #  step 2: Denominator: probability of x when k is given, p(x|all k)
  #  step 3: z is given by Numerator/ Denominator
  k=1
  while(k<=K) {
    #  step 1
    d=1
    while(d<=D){  # D= 10
      #multiplication of matrices
      mixture_matrix[1,k] = prod(mixture_matrix[1,k],
                                 (mu[k,d]^x[n,d]),(1-mu[k,d])^(1-x[n,d]))
      d=d+1
    }
    mixture_matrix[1,k] = mixture_matrix[1,k] * pi[k] # multiplying with weights
    # step 2
    mixture_matrix[1,K+1] = mixture_matrix[1,K+1] + mixture_matrix[1,k]
    k=k+1
  }
  # step 3:
  for (k in 1:K) {
    # Nmerator/Denominator
    z[n,k] = mixture_matrix[1,k] / mixture_matrix[1,K+1]

  }
  n=n+1
}

# Log likelihood computation
# Your code here

# implemeting of Log likelihood of Bernoulli mixture

n=1 # initializing n for N loop
while (n<=N) {    # summation due to logarithm
  k=1
  while(k<=K) {   # summation due to mixture
    # initializing log_likelihood as 0 to update values from loop below
    log_likelihood = 0
    d=1
    while(d<=D) {

      log_likelihood = log_likelihood + x[n,d] *
        log(mu[k,d]) + (1-x[n,d]) * log(1-mu[k,d])
      d=d+1
    }
    llik[it] = llik[it] + z[n,k] * (log(pi[k])+ log_likelihood)#iteration it in 1: 100(max_it)
    k=k+1     # incrementing k for while loop
  }
```

```
    n=n+1        # incrementing n for while loop
  }
  #not printing it anymore
  #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  #flush.console()
  # Stop if the log likelihood has not changed significantly
  # Your code here

  # if iteration is more than one  and has not changed significantly
  #i.e less than min_change of 0.1, then it should stop
  if(it != 1){
       if (abs(llik[it] - llik[it-1]) < min_change)
     {break}
   }

  # M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  # parameter estimation: pi
  pi=colMeans(z)    # mean of columns
  # parameter estimation :mu
  k=1
  while(k<=K) {
    mu[k,] = 0  # initialized mu to get value from loop
    n=1
    while(n<=N) {
      mu[k,] = mu[k,] + x[n,] * z[n,k]
      n=n+1
    }
    mu[k,] = mu[k,] / sum(z[,k])
    k=k+1
  }
}
  pi_4<-pi
  cat("\n Estimated value of pi \n")
  pi_4
  mu_4<-mu
  cat("\n Estimated value of mu: \n")
  mu_4
  llik_4<-llik[1:it]
  cat("\t number of iterations for K=4 ",length(llik_4),"\n")
  #plot(llik[1:it],type = "o")

  plot(llik_2, type="o",col="red",main = "Log liklihood graph",
       xlab="No. of iterations", ylab="Log likelihood",ylim =c(-9000,-6000),xlim=c(0,100) )
  par(new = TRUE) # for combining two plots
  plot(llik_3,type="o",col="green",xaxt = "n", yaxt = "n",
       ylab = "", xlab = "",ylim =c(-9000,-6000),xlim=c(0,100) )
  axis(side = 4)
  par(new = TRUE) # for combining two plots
  plot(llik_4,type="o",col="yellow",xaxt = "n", yaxt = "n",
       ylab = "", xlab = "",ylim =c(-9000,-6000),xlim=c(0,100))
  axis(side = 4)
  mtext("EM : Mixtures of Multivariate Benoulli Distributions", side = 4, line = 3);
```

```
    legend("topright", c("K=2 ", "K=3","K=4"),col = c("red","green","yellow"),
           lty = c(1, 2)) # Indexing the plot at top right position



#-------------------------------------------------------------------------------
# Additional Concept references:
    #https://cedar.buffalo.edu/~srihari/CSE574/Chap9/Ch9.4-MixturesofBernoulli.pdf
```