Lab1_Block2

Group A15

December 4, 2018

Ensemble methods

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```r
library(mboost)
```

```
## Loading required package: parallel
```

```
## Loading required package: stabs
```

```
## This is mboost 2.9-1. See 'package?mboost' and 'news(package  = "mboost")'
## for a complete list of changes.
```

```r
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

```r
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
```

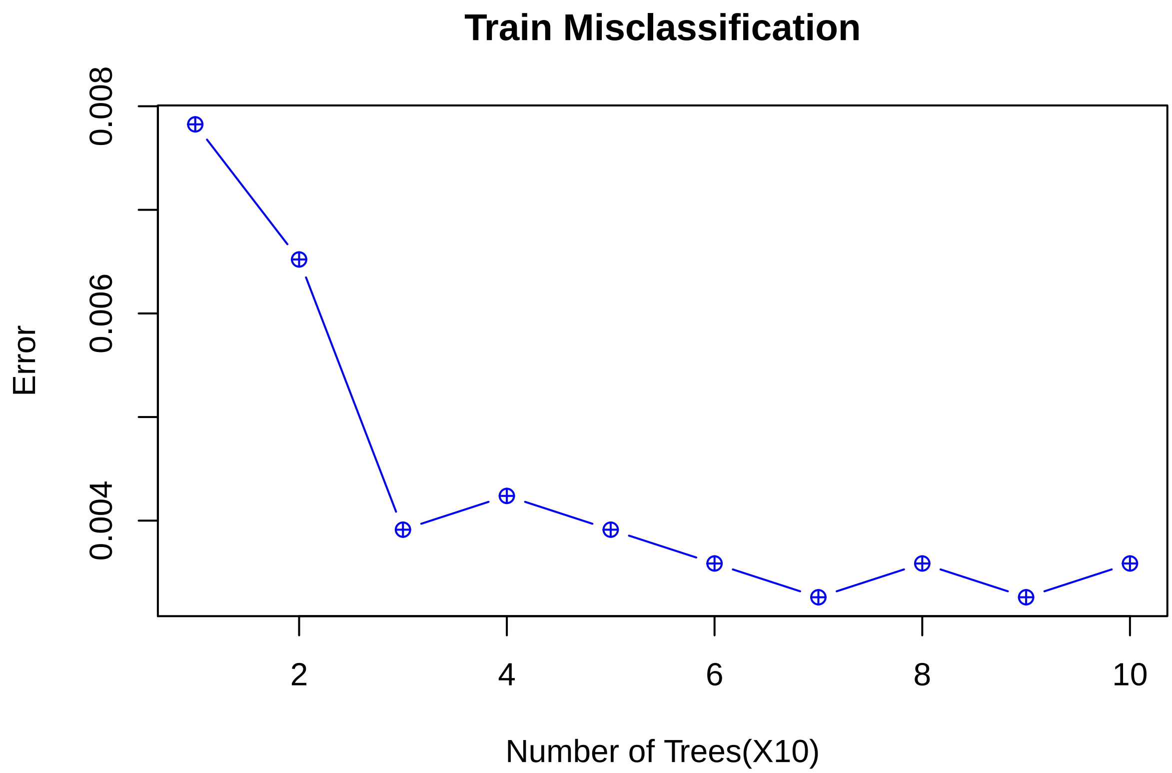
```
## The following object is masked from 'package:mboost':
##
##     %+%
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

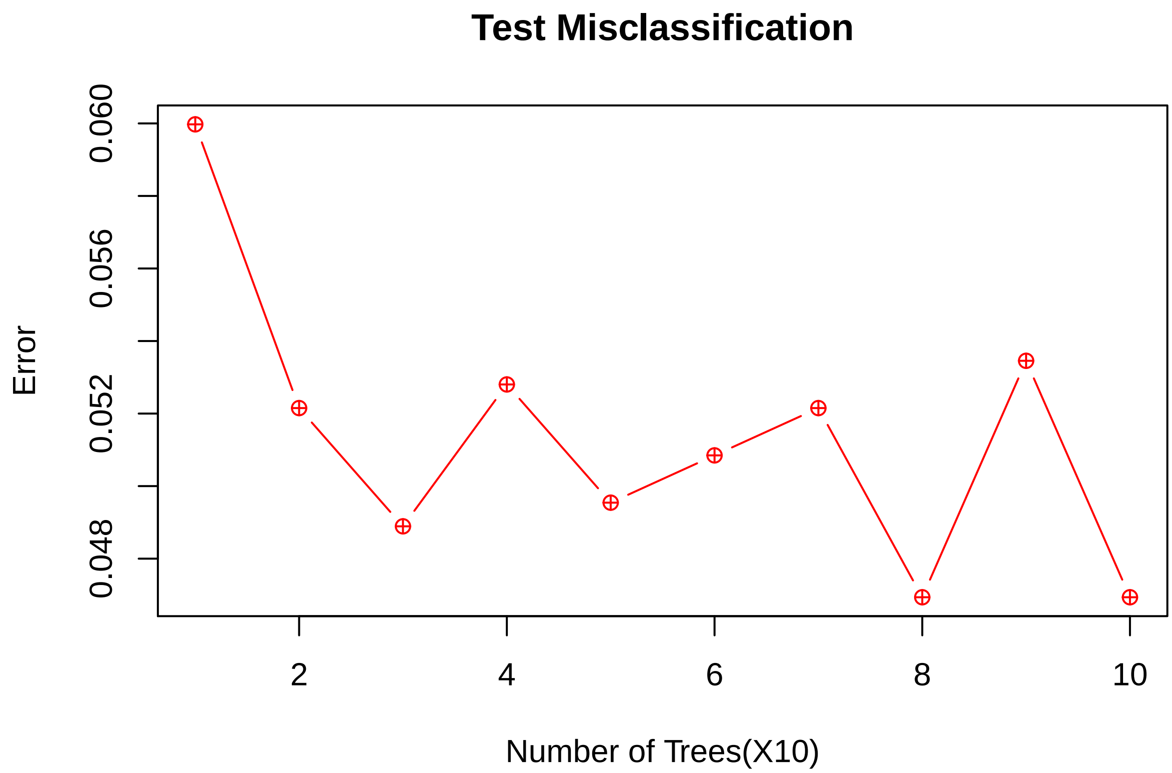
```r
sp <- read.csv2("spambase.csv")
sp$Spam <- as.factor(sp$Spam)

n=dim(sp)[1]
set.seed(12345)
id=sample(1:n, floor(n*(2/3)))
train=sp[id,]
test=sp[-id,]

number_of_trees <- seq(from = 10,to = 100, by = 10)
```

Random forest

```r
random_forest <- function(ntrees){

  model_1 <- randomForest(Spam ~ .,data = sp,subset = id,ntree = ntrees,importance = T)

  predict_1 <- predict(model_1,newdata = train, type = "class")
  con_mat_1 <- table(predict_1, train$Spam)
  train_error_1 <- 1-sum(diag(con_mat_1))/sum(con_mat_1)

  predict_2 <- predict(model_1,newdata = test, type = "class")
  con_mat_2 <- table(predict_2, test$Spam)
  test_error_1 <- 1-sum(diag(con_mat_2))/sum(con_mat_2)

  return(list(train_error = train_error_1,test_error = test_error_1))
}

error_rate_rf <- as.data.frame(t(sapply(number_of_trees, random_forest)))
train_error_rf <- as.vector(unlist(error_rate_rf$train_error))
test_error_rf <- as.vector(unlist(error_rate_rf$test_error))

plot(train_error_rf,type = "b",main="Train Misclassification", xlab= "Number of Trees(X10)",
     ylab= "Error", col="blue", pch=10, cex=1)
```




```
plot(test_error_rf,type = "b",main="Test Misclassification", xlab= "Number of Trees(X10)",
     ylab= "Error", col="red", pch=10, cex=1)
```

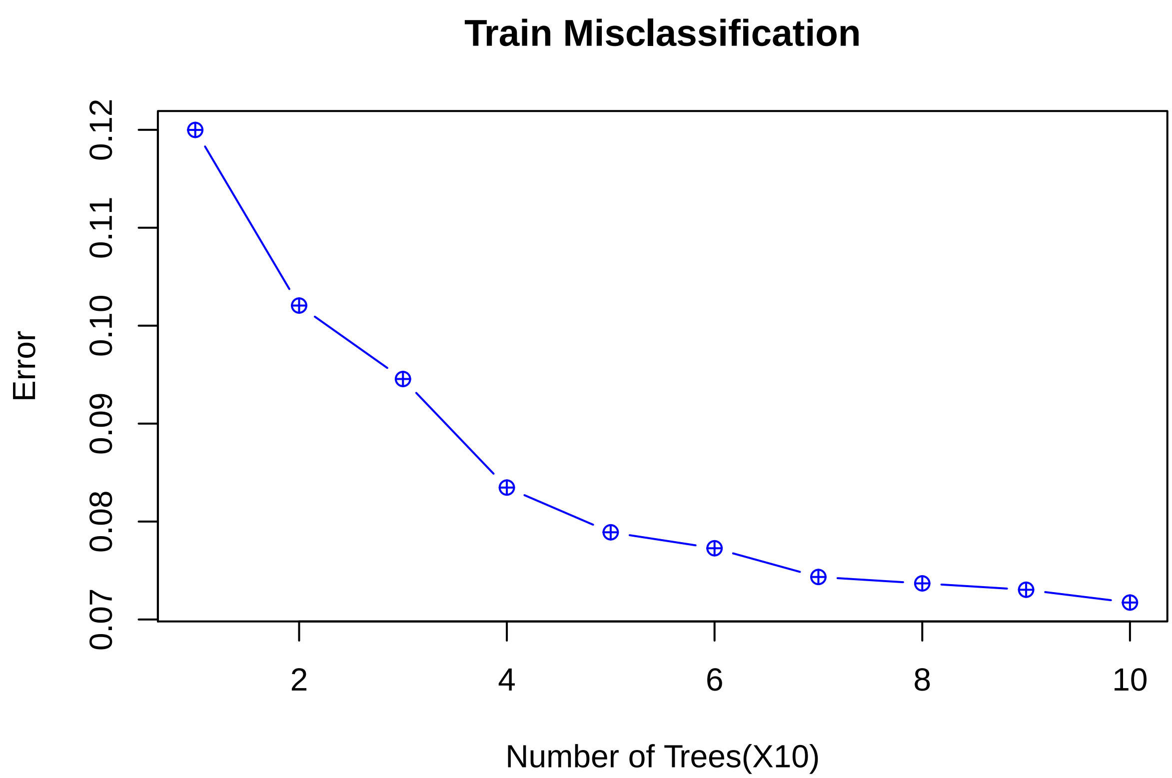


The above plots show the change in error for train and test data with respect to the number of trees considered. It can be seen that the error decreases till the number of trees considered increses upto 30 for Train data and 20 for test data. However after the number of tress increases after a particular number, in the case of train data 30 and for test data 20, the error rate increases and then again decreses. There is an almost alternate increase and decrease in the error with the increase in trees.

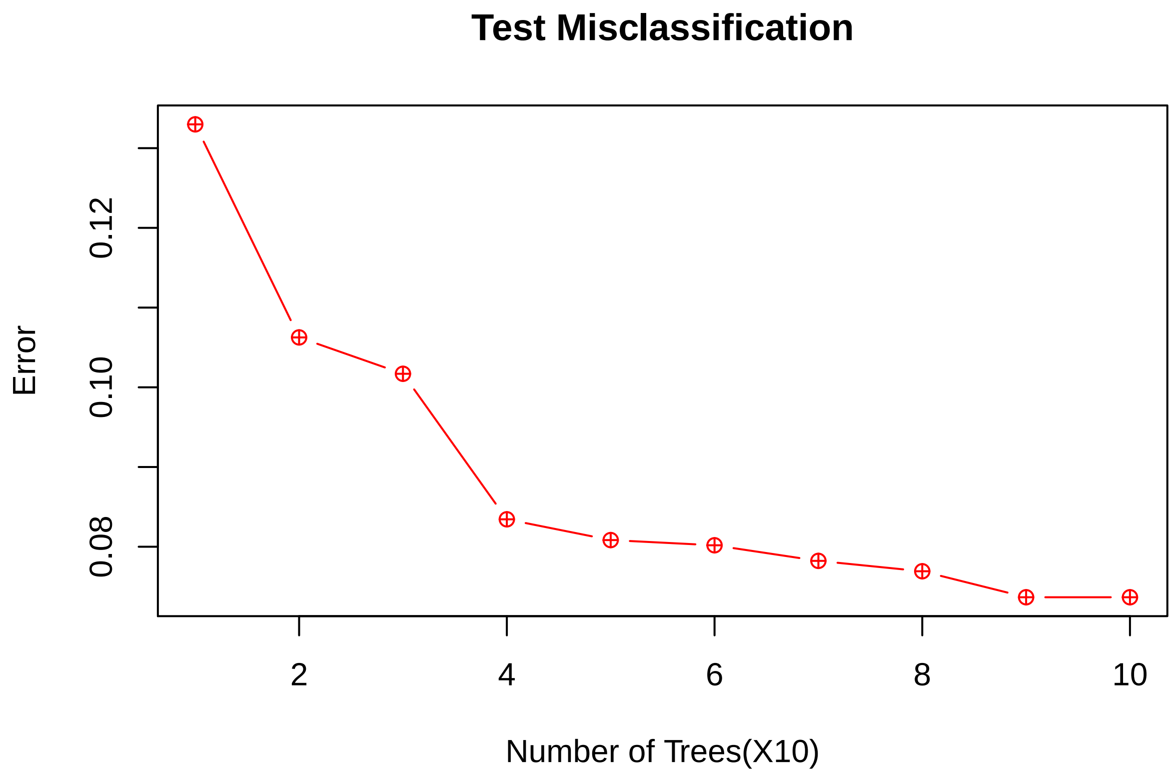
Adaboost classi???cation trees

```
adaboost <- function(ntrees){

  model_2 <- blackboost(Spam ~., data = train,
                       control = boost_control(mstop = ntrees, nu=0.1), family = AdaExp())

  predict_3 <- predict(model_2,newdata = train, type = "class")
  con_mat_3 <- table(predict_3, train$Spam)
  train_error_2 <- 1-sum(diag(con_mat_3))/sum(con_mat_3)

  predict_4 <- predict(model_2,newdata = test, type = "class")
  con_mat_4 <- table(predict_4, test$Spam)
  test_error_2 <- 1-sum(diag(con_mat_4))/sum(con_mat_4)

  return(list(train_error = train_error_2,test_error = test_error_2))
}
```

```r
error_rate_ada <- as.data.frame(t(sapply(number_of_trees, adaboost)))
train_error_ada <- as.vector(unlist(error_rate_ada$train_error))
test_error_ada <- as.vector(unlist(error_rate_ada$test_error))

plot(train_error_ada,type = "b",main="Train Misclassification", xlab= "Number of Trees(X10)",
     ylab= "Error", col="blue", pch=10, cex=1)
```




```r
plot(test_error_ada,type = "b",main="Test Misclassification", xlab= "Number of Trees(X10)",
     ylab= "Error", col="red", pch=10, cex=1)
```



The above plots show the change in error for train and test data with respect to the number of trees considered. Unlike Random forest there is no alternating between increase and decrease in error with the increase in trees, rather the error decreases continuously with the increase in number of trees.

Comparision of error for train data

```
ada_rf_train <- data.frame(adaboost = train_error_ada, random_forest = train_error_rf,
                 no_of_trees = number_of_trees)

plot_ada_train <- ggplot(ada_rf_train, aes(no_of_trees, y = adaboost)) +
  geom_line(colour="blue")+geom_point()+
  ggtitle("Adaboost Vs Random Forest","Train Misclassification")

plot_rf_train <- ggplot(ada_rf_train, aes(no_of_trees, y = random_forest)) +
  geom_line(colour="red")+geom_point()

grid.arrange(plot_ada_train,plot_rf_train,ncol=1,nrow=2)
```

The error for Adaboost seems to be much more than that for Random forest considering train data. The Random forest achieves its optimum i.e. gives least error with fairly less number of trees,approximately 30 tress in this case, compared to Adaboost.

Comparision of error for test data

```
ada_rf_test <- data.frame(adaboost = test_error_ada, random_forest = test_error_rf,
                  no_of_trees = number_of_trees)

plot_ada_test <- ggplot(ada_rf_test, aes(no_of_trees, y = adaboost)) +
  geom_line(colour="blue")+geom_point()+
  ggtitle("Adaboost Vs Random Forest","Test Misclassification")

plot_rf_test <- ggplot(ada_rf_test, aes(no_of_trees, y = random_forest)) +
  geom_line(colour="red")+geom_point()

grid.arrange(plot_ada_test,plot_rf_test,ncol=1,nrow=2)
```

The error for Adaboost seems to be much more than that for Random forest considering test data when less number of trees are considered. However it can be seen that as the number of trees increases the error for Adaboost decreses drastically and almost equals the error rate for Random forest, in this case when the number of trees is 100.

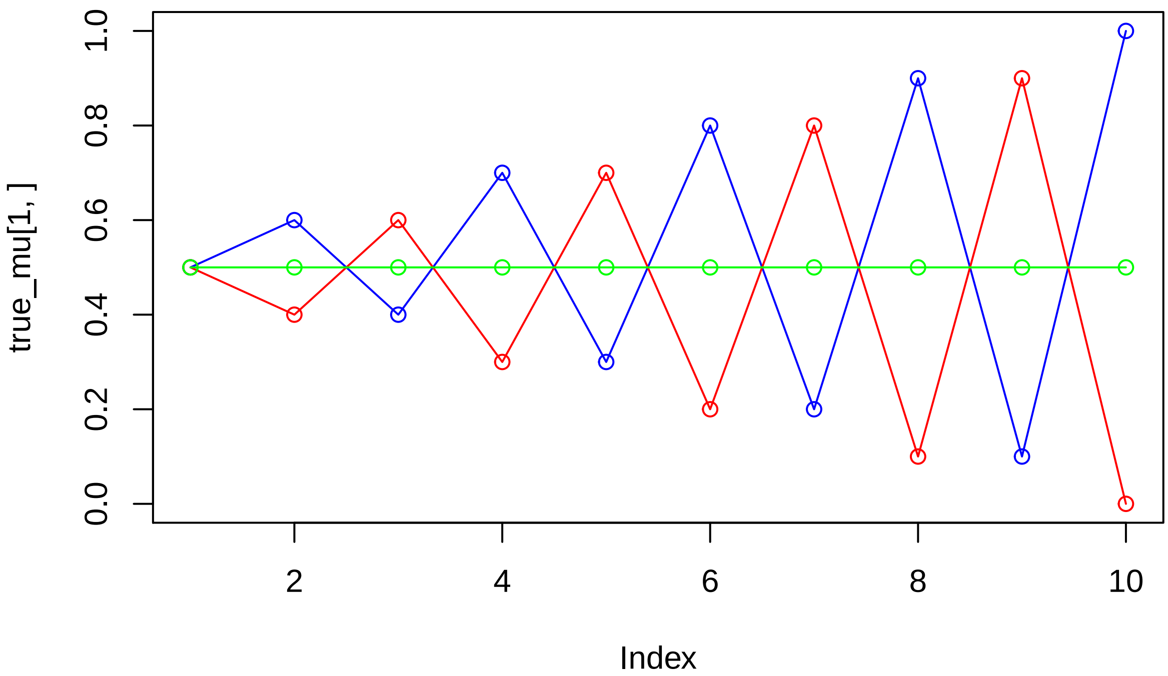
Mixture Models

```r
em_function <- function(given_k)
{
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")
```

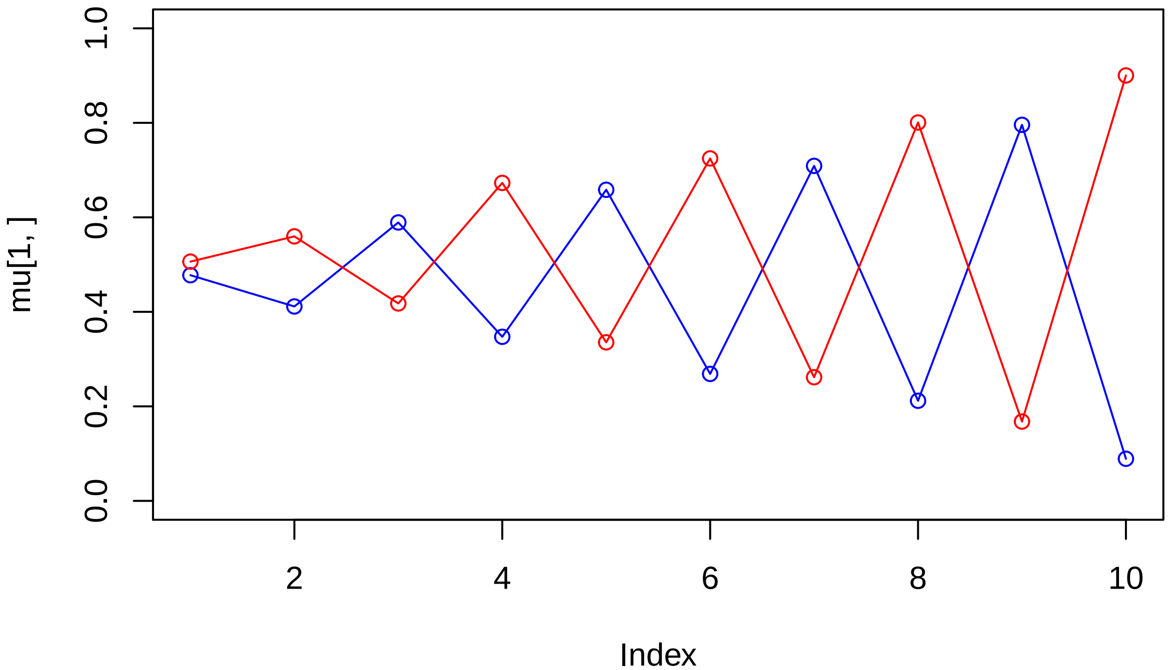
```r
# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}
K=given_k # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- 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
for(it in 1:max_it) {
  # if (k==2)
  # {
  #   plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  #   points(mu[2,], type="o", col="red")
  # }
  # else if (k==3)
  # {
  #   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")
  # }
  # else
  # {
  #   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
  for (n in 1:N)
  {
    prob=0
    for (k in 1:K)
    {
      prob=prob+prod(((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))*pi[k]
    }
    for (k in 1:K)
    {
      z[n,k]=pi[k]*prod(((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,])))) / prob
```

```
    }
  }
  #Log likelihood computation.
  # Your code here
  likelihood <-matrix(0,nrow =1000,ncol = K)
  llik[it] <-0
  for(n in 1:N)
  {
    for (k in 1:K)
    {
      likelihood[n,k] <- pi[k]*prod(((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))
    }
    llik[it]<- sum(log(rowSums(likelihood)))
  }
  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the log likelihood has not changed significantly
  # Your code here
  if (it > 1)
  {
    if (llik[it]-llik[it-1] < min_change)
    {
      if(K == 2)
      {
        plot(mu[1,], type="o", col="blue", ylim=c(0,1))
        points(mu[2,], type="o", col="red")
      }
      else if(K==3)
      {
        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")
      }
      else
      {
        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")
      }
      break
    }
  }
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
    mu<- (t(z) %*% x) /colSums(z)
    pi <- colSums(z)/N
  }
pi
mu
plot(llik[1:it], type="o")
}
```

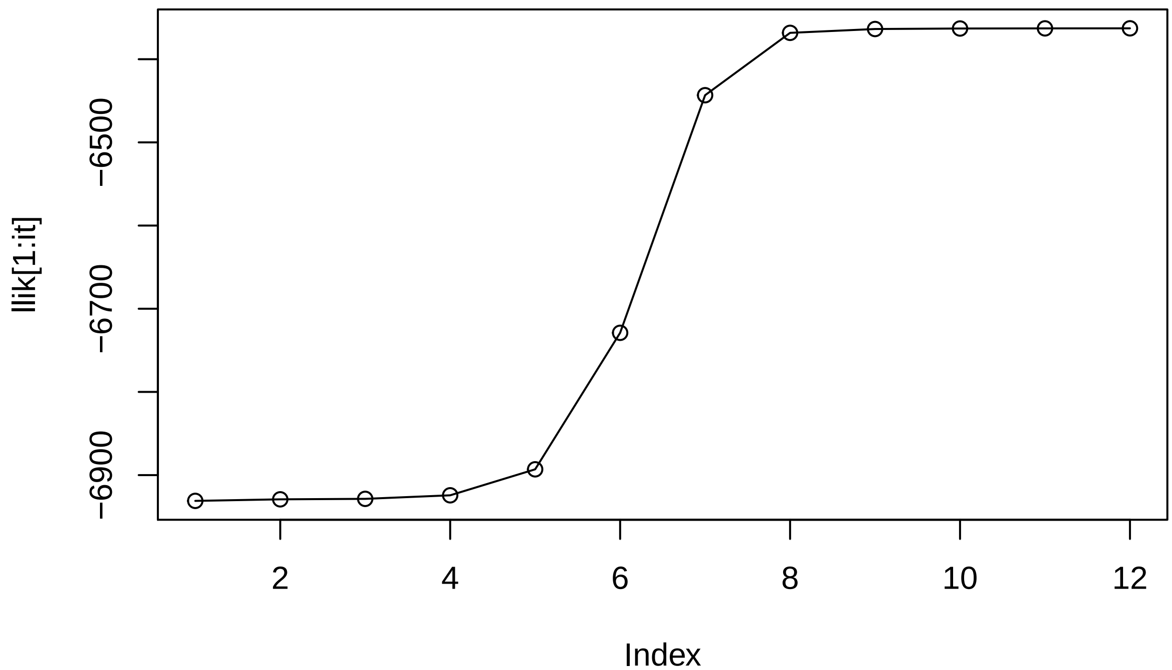
```r
em_function(2)
```




```
## 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
## iteration:  12 log likelihood:  -6362.897
```

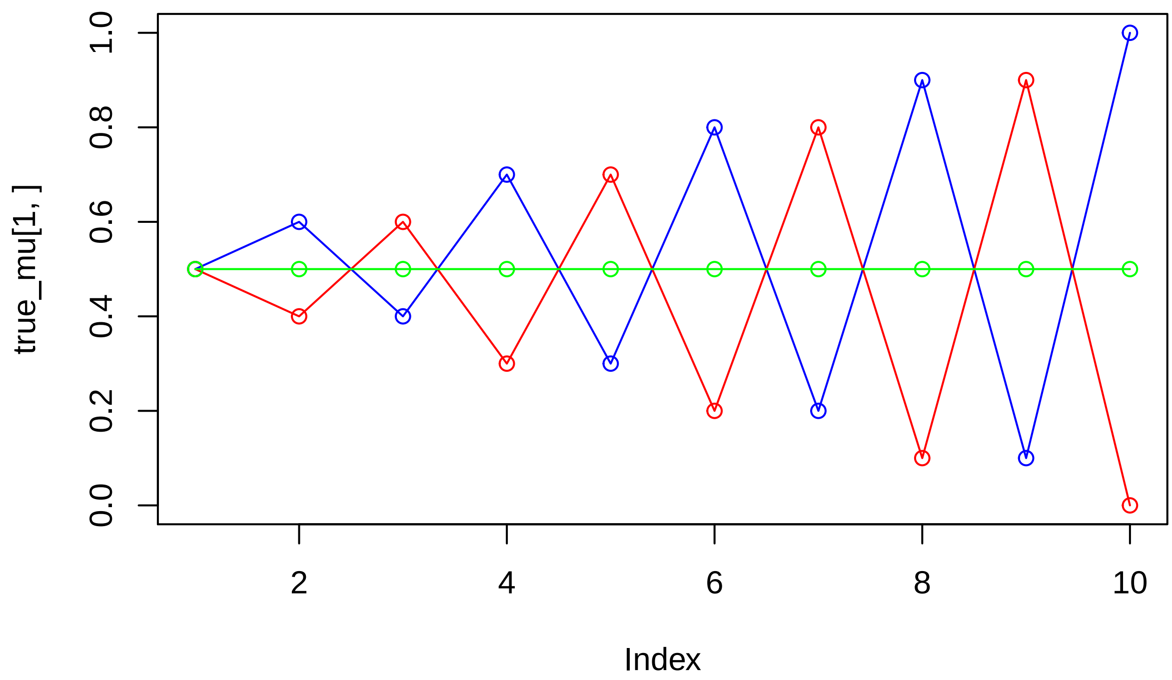


1.0
0.8
0.6
0.4
0.2
0.0
mu[1,]
2
4
6
8
10
Index

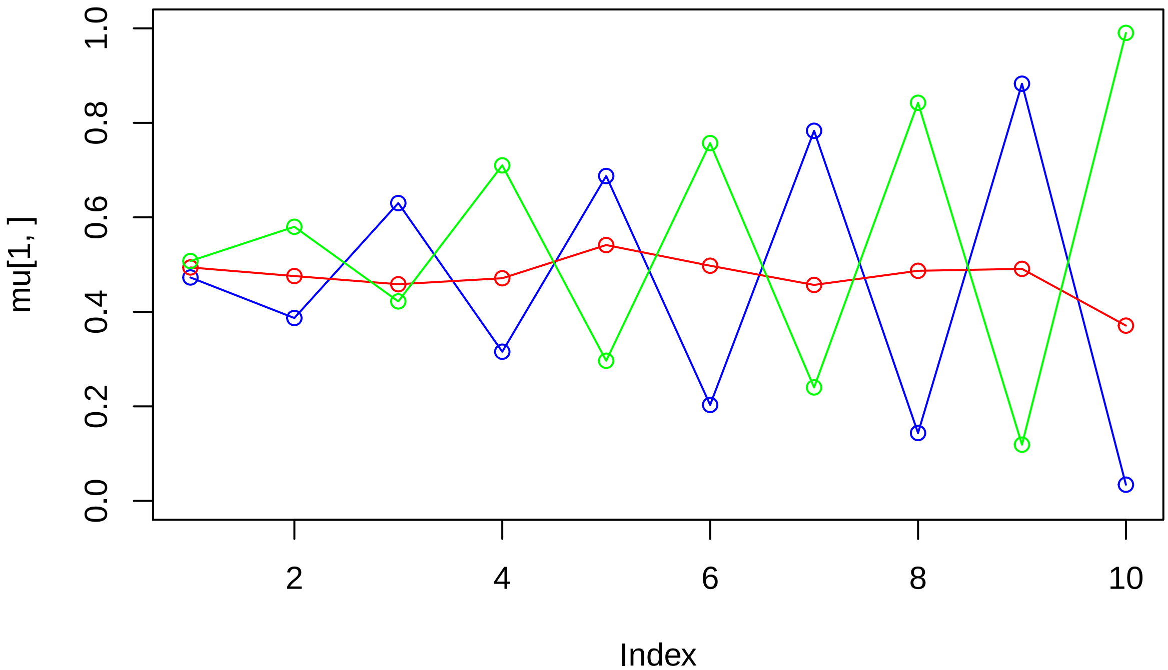


The above plots corresponds to k=2, the third component is not visible in the second plot when compared the first plot i.e. the original plot due to this fact. The data points are almost equally distributed since only to class exaists with each data point having equal proability getting classified into 1 class.

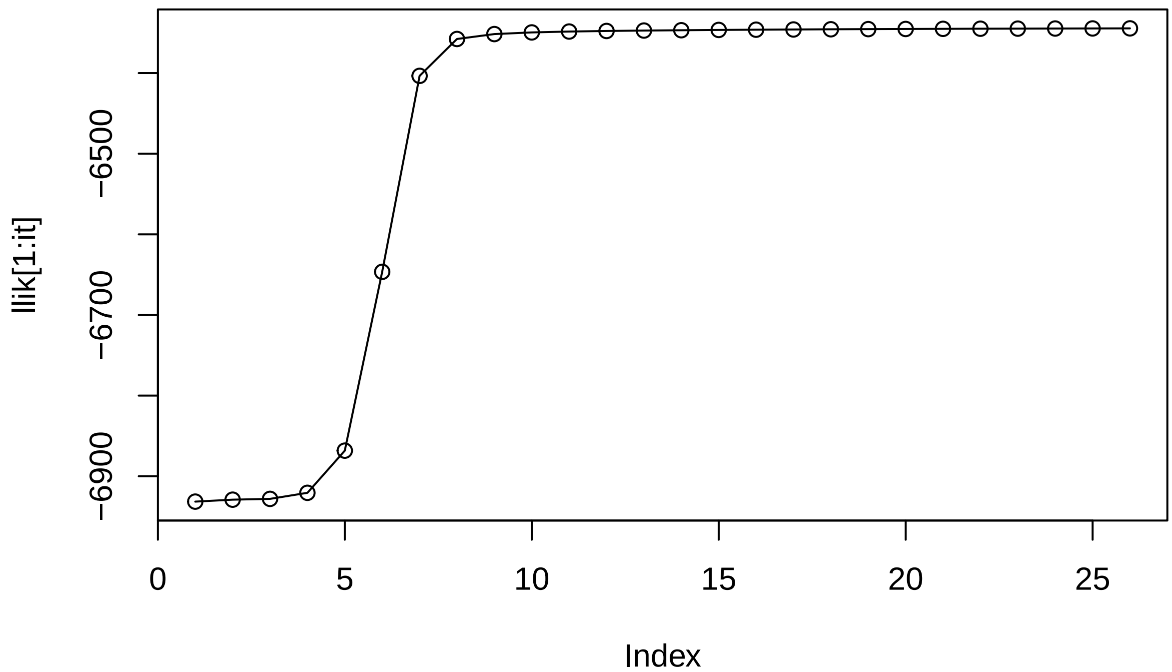
```
em_function(3)
```




```
## iteration:  1 log likelihood:  -6931.482
## iteration:  2 log likelihood:  -6929.074
## iteration:  3 log likelihood:  -6928.081
## iteration:  4 log likelihood:  -6920.57
## iteration:  5 log likelihood:  -6868.29
## iteration:  6 log likelihood:  -6646.505
## iteration:  7 log likelihood:  -6403.476
## iteration:  8 log likelihood:  -6357.743
## iteration:  9 log likelihood:  -6351.637
## iteration:  10 log likelihood:  -6349.59
## iteration:  11 log likelihood:  -6348.513
## iteration:  12 log likelihood:  -6347.809
## iteration:  13 log likelihood:  -6347.284
## iteration:  14 log likelihood:  -6346.861
## iteration:  15 log likelihood:  -6346.506
## iteration:  16 log likelihood:  -6346.2
## iteration:  17 log likelihood:  -6345.934
## iteration:  18 log likelihood:  -6345.699
## iteration:  19 log likelihood:  -6345.492
## iteration:  20 log likelihood:  -6345.309
## iteration:  21 log likelihood:  -6345.147
## iteration:  22 log likelihood:  -6345.003
## iteration:  23 log likelihood:  -6344.875
## iteration:  24 log likelihood:  -6344.762
## iteration:  25 log likelihood:  -6344.66
## iteration:  26 log likelihood:  -6344.57
```

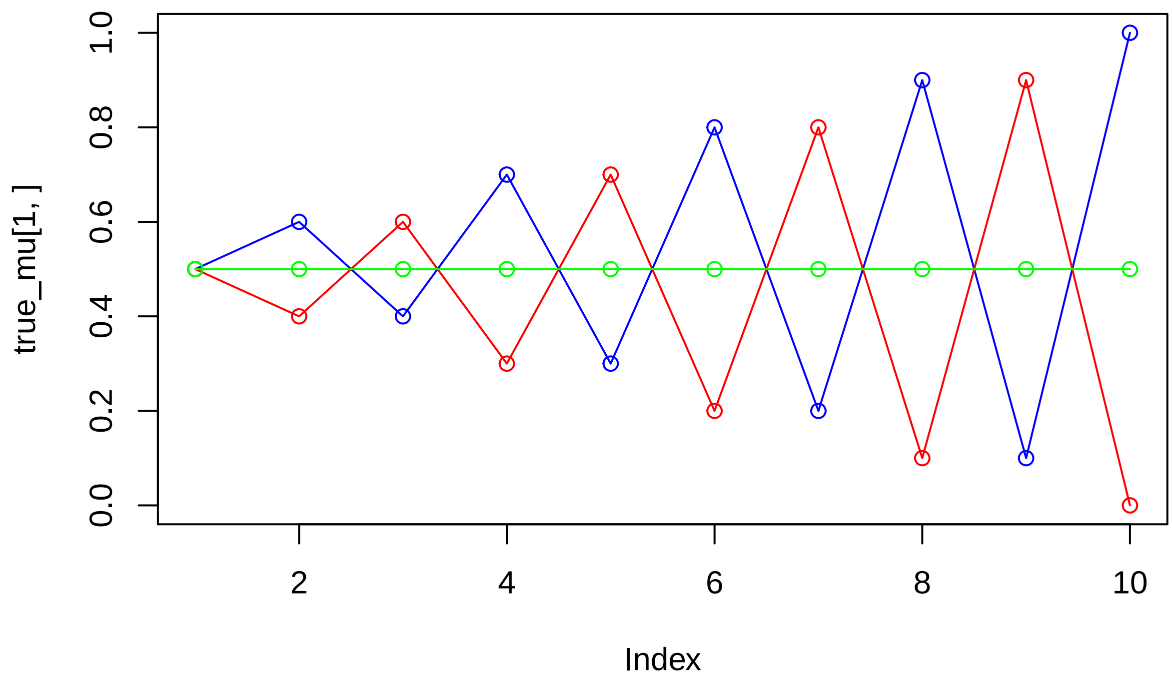


mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



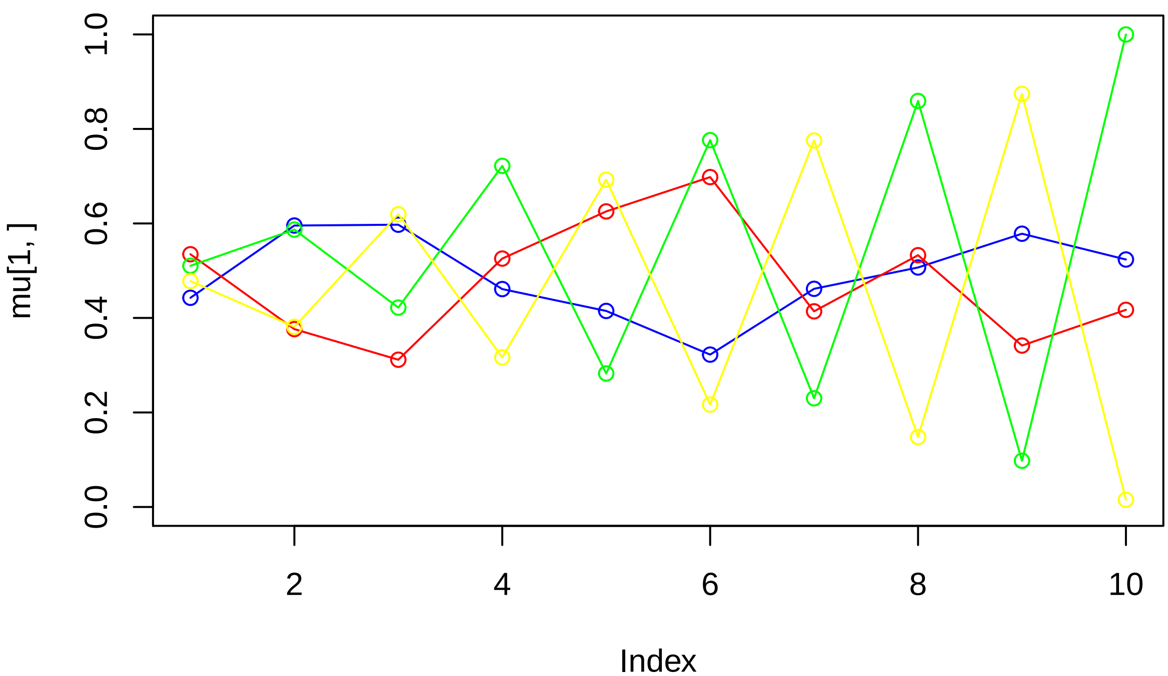
The above plots corresponds to k=3. The change in 3rd component is evident through the second plot where the estimated mu is plotted and looks different from the actual mu.

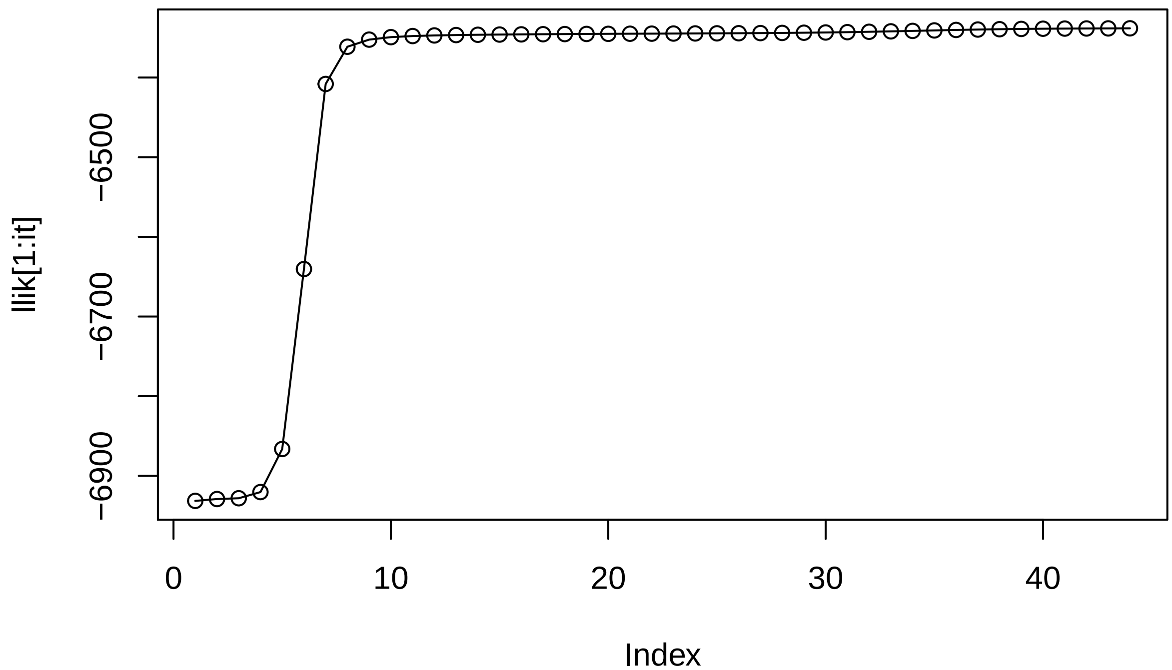
```
em_function(4)
```




```
## iteration:  1 log likelihood:  -6931.372
## iteration:  2 log likelihood:  -6929.087
## iteration:  3 log likelihood:  -6928.057
## iteration:  4 log likelihood:  -6920.335
## iteration:  5 log likelihood:  -6866.277
## iteration:  6 log likelihood:  -6640.396
## iteration:  7 log likelihood:  -6408.058
## iteration:  8 log likelihood:  -6361.322
## iteration:  9 log likelihood:  -6352.413
## iteration:  10 log likelihood:  -6349.293
## iteration:  11 log likelihood:  -6347.902
## iteration:  12 log likelihood:  -6347.148
## iteration:  13 log likelihood:  -6346.663
## iteration:  14 log likelihood:  -6346.308
## iteration:  15 log likelihood:  -6346.028
## iteration:  16 log likelihood:  -6345.797
## iteration:  17 log likelihood:  -6345.601
## iteration:  18 log likelihood:  -6345.43
## iteration:  19 log likelihood:  -6345.279
## iteration:  20 log likelihood:  -6345.142
## iteration:  21 log likelihood:  -6345.015
## iteration:  22 log likelihood:  -6344.894
## iteration:  23 log likelihood:  -6344.775
## iteration:  24 log likelihood:  -6344.652
## iteration:  25 log likelihood:  -6344.52
## iteration:  26 log likelihood:  -6344.373
```

```
## iteration:  27 log likelihood:  -6344.2
## iteration:  28 log likelihood:  -6343.992
## iteration:  29 log likelihood:  -6343.737
## iteration:  30 log likelihood:  -6343.421
## iteration:  31 log likelihood:  -6343.033
## iteration:  32 log likelihood:  -6342.57
## iteration:  33 log likelihood:  -6342.036
## iteration:  34 log likelihood:  -6341.451
## iteration:  35 log likelihood:  -6340.849
## iteration:  36 log likelihood:  -6340.272
## iteration:  37 log likelihood:  -6339.757
## iteration:  38 log likelihood:  -6339.327
## iteration:  39 log likelihood:  -6338.988
## iteration:  40 log likelihood:  -6338.732
## iteration:  41 log likelihood:  -6338.544
## iteration:  42 log likelihood:  -6338.406
## iteration:  43 log likelihood:  -6338.304
## iteration:  44 log likelihood:  -6338.228
```





The above plots corresponds to k=4. The addition of an extra component is evident in the 2nd plot which shows the plotted estimated mu.