Introduction to Machine Learning - Lab 1 Block 2

Ugurcan Lacin

11/16/2017

Ensemble Methods

Data is imported in R and divided as train and test data. The data set ,which is spambase.csv, contains information about the frequency of various words, characters, etc. for a total of 4601 e-mails. Furthermore, these e-mails have been classified as spams (spam = 1) or regular e-mails (spam = 0).

```
library(mboost)
library(randomForest)
library(ggplot2)

data <- read.csv("/home/ugur/git/ML_Lab1_Block2_Group/spambase.csv", sep = ";",dec = ",")
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.66))
train=data[id,]
test=data[-id,]
```

In this assignment, Random Forest and Adaboost algorithms will be applied. They will be used with different tree numbers from 10 to 100 by increasing 10. Then, the result will be analyzed with respect to misclassification rates each other.

For that purpose, Random Forest is used to fit model with mentioned sequence of trees.

```
numberOfTrees <- seq(10,100,10)
randomForestErrorRates <- c()
for(i in 1:10){
  numberOfTree <- numberOfTrees[i]
  resRandomForest <- randomForest(as.factor(Spam) ~ .,data = train,ntree=numberOfTree)
  pred <- predict(resRandomForest,newdata=test)
  tab <- table(pred,test$Spam)
  misclassificationRate <- 1-sum(diag(tab))/sum(tab)
  randomForestErrorRates <- cbind(randomForestErrorRates,misclassificationRate)
}
```

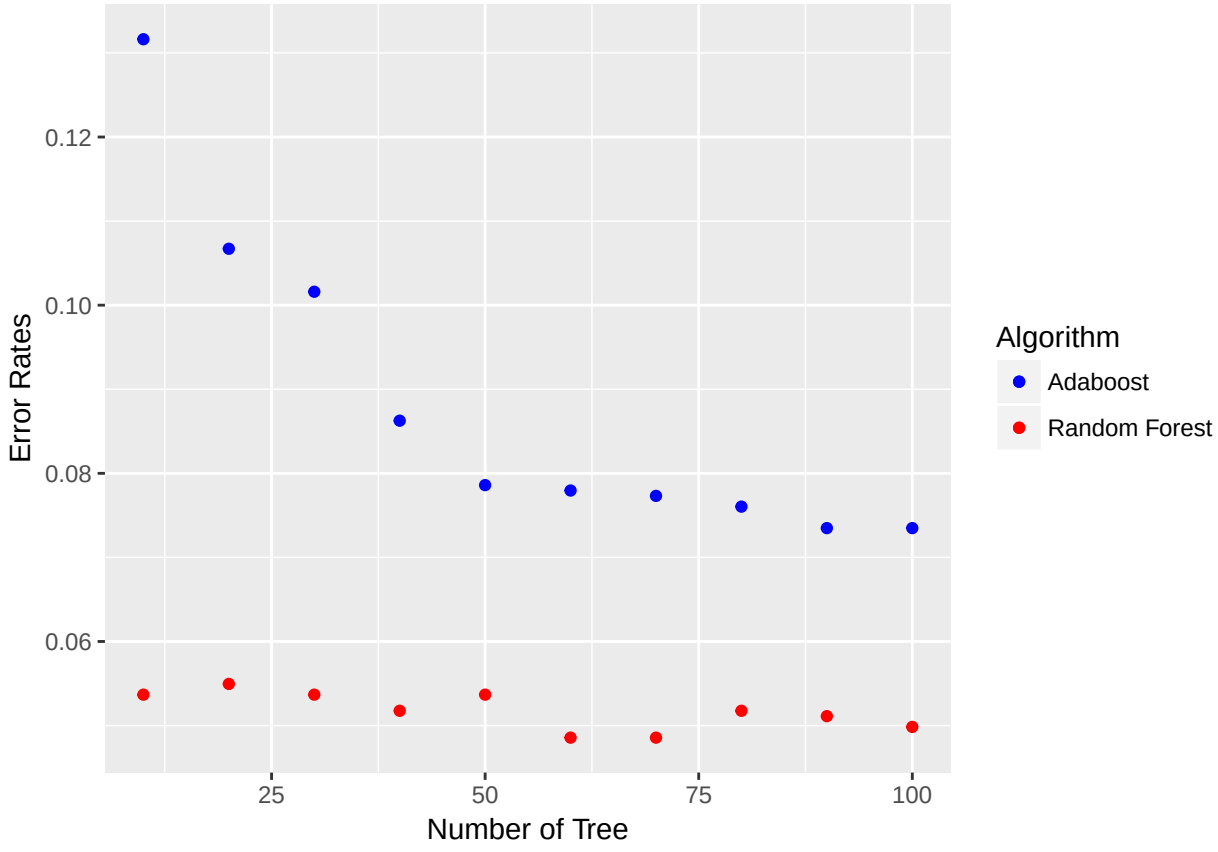
Then, same procedure is applied for Adaboost algorithm as well, and stored misclassification rates as stored previously.

```
AdaboostErrorRates <- c()
for(i in 1:10){
  numberOfTree <- numberOfTrees[i]
  resAdaboost <- blackboost(as.factor(Spam) ~ ., data = train,
```

```
                              control = boost_control(mstop= numberOfTree),family = AdaExp())
  pred <- predict(resAdaboost,newdata=test,type="class")
  tab <- table(pred,test$Spam)
  misclassificationRate <- 1-sum(diag(tab))/sum(tab)
  AdaboostErrorRates <- cbind(AdaboostErrorRates,misclassificationRate)
}
```

Once we calculate the misclassification rates for both algorithms, we put the results into a form that the ggplot library can interpret to compare the results. Then, We plot comperative graph with both misclassification rates.

```
mx <- as.matrix(randomForestErrorRates)
rfdata <- as.data.frame(mx[1,])
mx2 <- as.matrix(AdaboostErrorRates)
adadata <- as.data.frame(mx2[1,])

ggplot() + geom_point(mapping = aes( x=numberOfTrees ,y = mx[1, ], colour = "red"),data = rfdata) +
 geom_point(mapping = aes( x=numberOfTrees ,y = mx2[1, ], colour = "blue"),data = adadata) +
  labs(x="Number of Tree",y="Error Rates", ttile="Adaboost vs Random Forest",color = "Algorithm") +
  scale_color_manual(labels = c("Adaboost", "Random Forest"), values = c("blue", "red"))
```

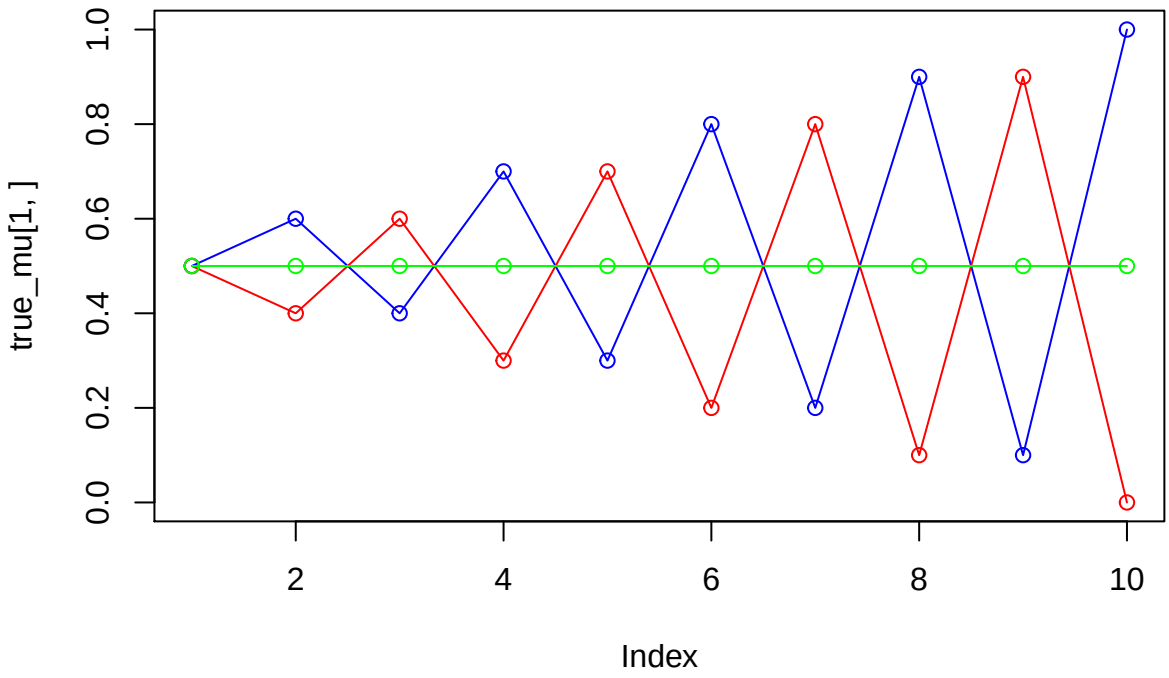


When the graph is analyzed, the error rate decreases as the number of trees increases in the Random Forest algorithm. But this decreasing is taking place with a very low slope and irregularity. On the contrary, the

Adaboost algorithm reduces the error rate with a large slope in parallel with the increase of the number of trees. After 50 as a number of tree continues to reduce the error rate in a linear manner with a lower slope rate. The result that can be extracted from here is as follows. For the Adaboost algorithm, the optimum error rate can be found by further increasing the number of trees. However, increasing the number of trees does not make a huge impact on the random forest algorithm unlike Adaboost.

Ensemble Methods

```
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")
```



The graph above shows the true data we are trying to predict with the EM Algorithm.

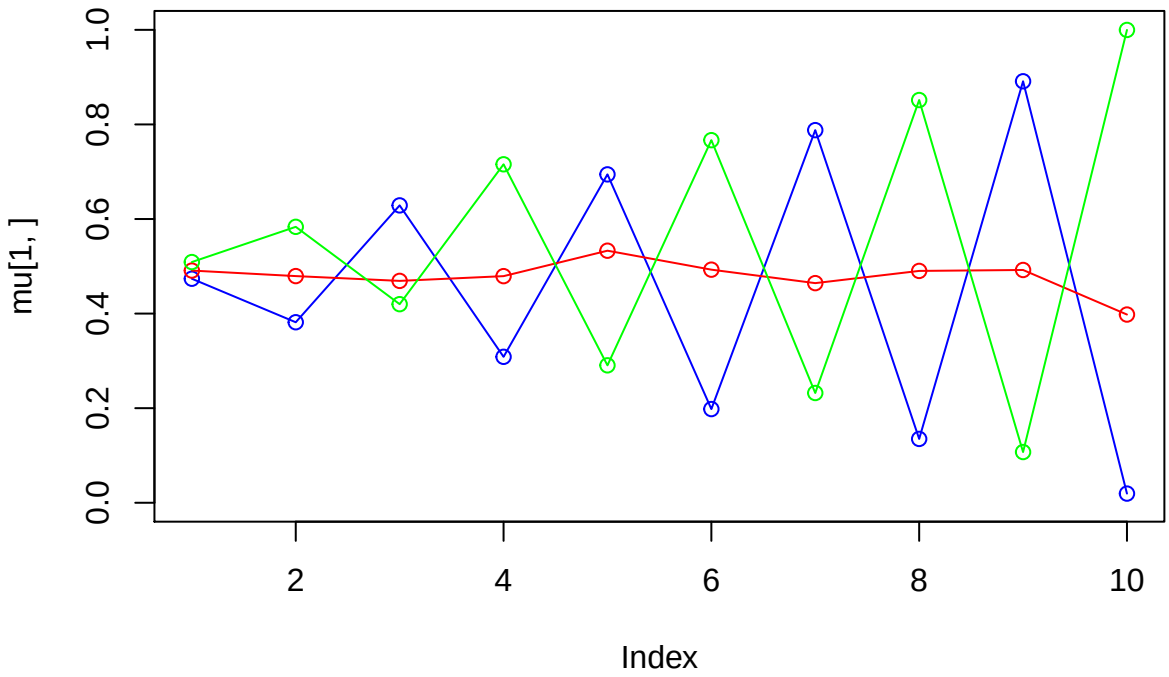
Our input data is matrix x, which is Bernoulli distributed with 10 *dimensions*, D. The true number of *components* in the data is 3, the probability of belonging to each component, π is 1/3 and μ is the mean for each component.

```
# 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)
}
```

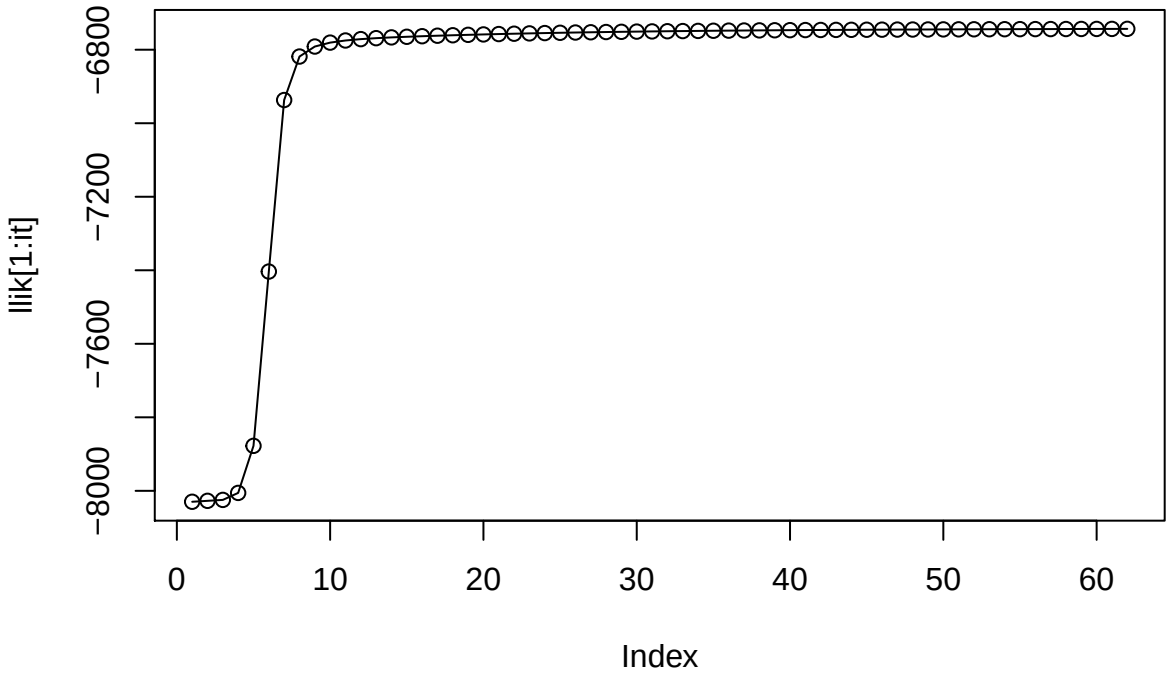
The code below first generates 10 random numbers of the density bernoulli distribution, one number for each *dimension*. Thereafter the product of the vector is multiplied with the probability of belonging to each *component* and that product is standardized so the sum of probabilities equals to 1.

```
for(it in 1:max_it) {
    # E-step: Computation of the fractional component assignments
    # pi prob of being in a certain component
    # mu is the mean for the k component
    xrow <- c()
    for(k in 1:K){
        for(n in 1:N){
            xrow <- dbinom(x = x[n, ], size = 1, prob = mu[k,])
            z[n,k] <- prod(xrow) * pi[k]
        }
    }

    z <- apply(z, 2, function(x){x/rowSums(z)})

    # Log likelihood computation.
    # Equation 9.54 Bishop
    loglik <- 0
    for(n in 1:N){
        for(k in 1:K){
            tmp=0
            for(d in 1:D){
                tmp = tmp +  x[n,d] * log(mu[k,d]) + (1 - x[n,d]) * log(1 - mu[k,d])
```

```
            }
            loglik <- loglik + z[n,k] * (log(pi[k]) + tmp)
        }
    }

    llik[it] <- loglik

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

    # Stop if the lok likelihood has not changed significantly
    if(it > 1){
    if((llik[it] - llik[it-1] < min_change)){break}
        }

    #M-step: ML parameter estimation from the data and fractional component assignments
    pi <- colSums(z)/N
    denom <- (t(x) %*% z)
    mu_new <- matrix(NA, ncol = K, nrow = 10)
    for(k in 1:K){
        output <- c()
        mu_new[,k] <- as.matrix(denom[,k]/sum(z[,k]))
    }
    mu <- t(mu_new)
}

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")
```



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

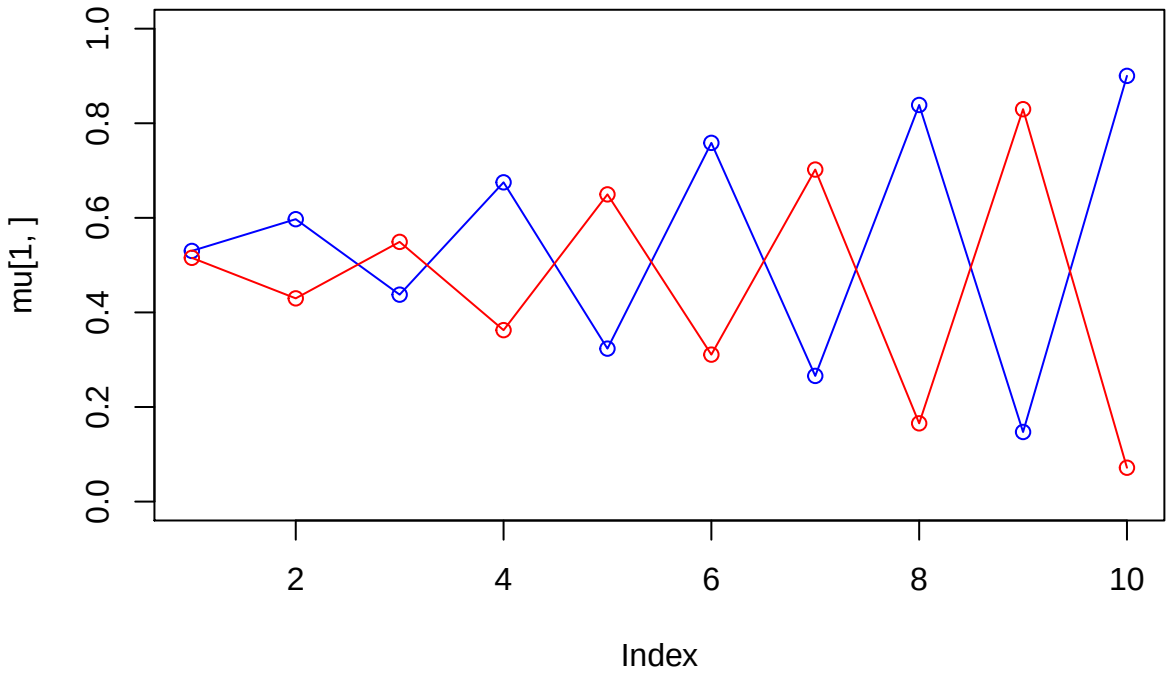


When the K is equal to 3, 62 iterations after EM algorithm stops with a log likelihood -6743.326.

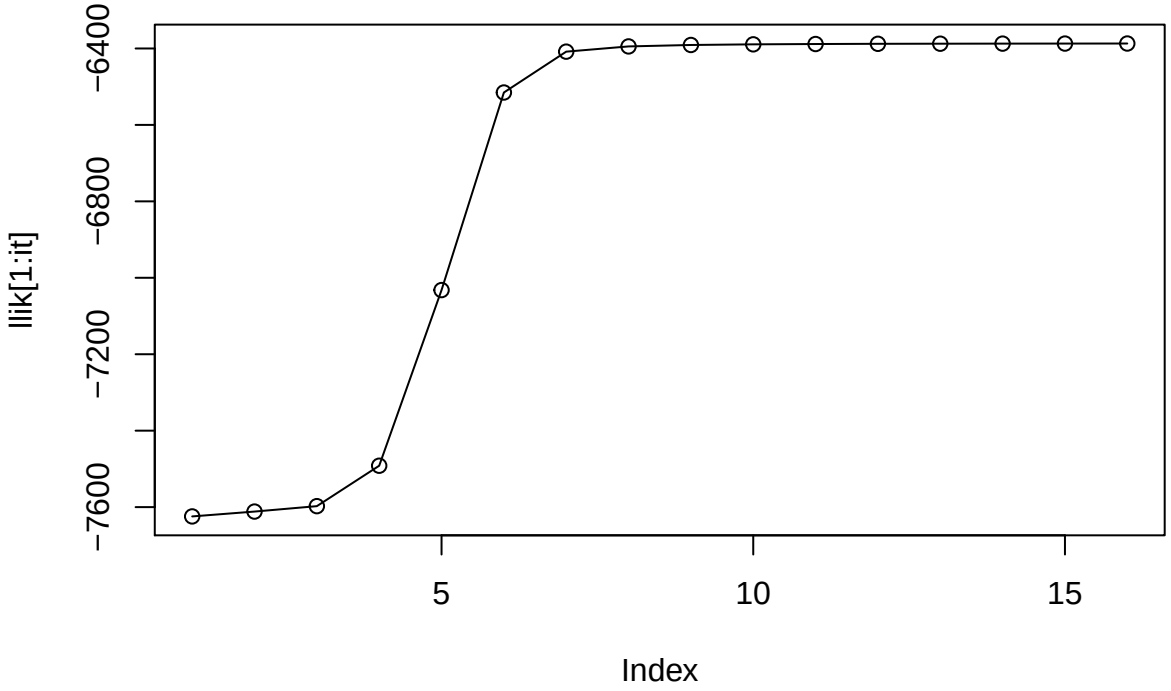
What happens if we try different K?

Model can be faster in case we suspect there is only two components, but the true number of components is 3. It takes 16 iterations at most.However, two to components captured seem to be similar as the two first components in the case with K = 3.

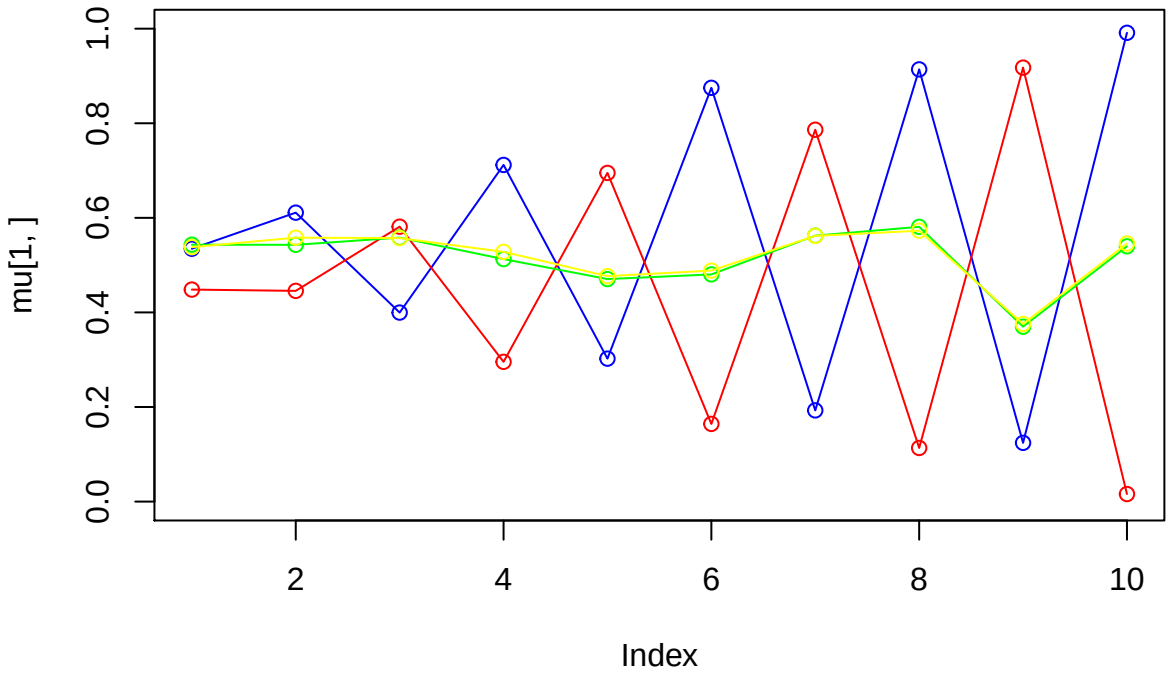
```
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
```



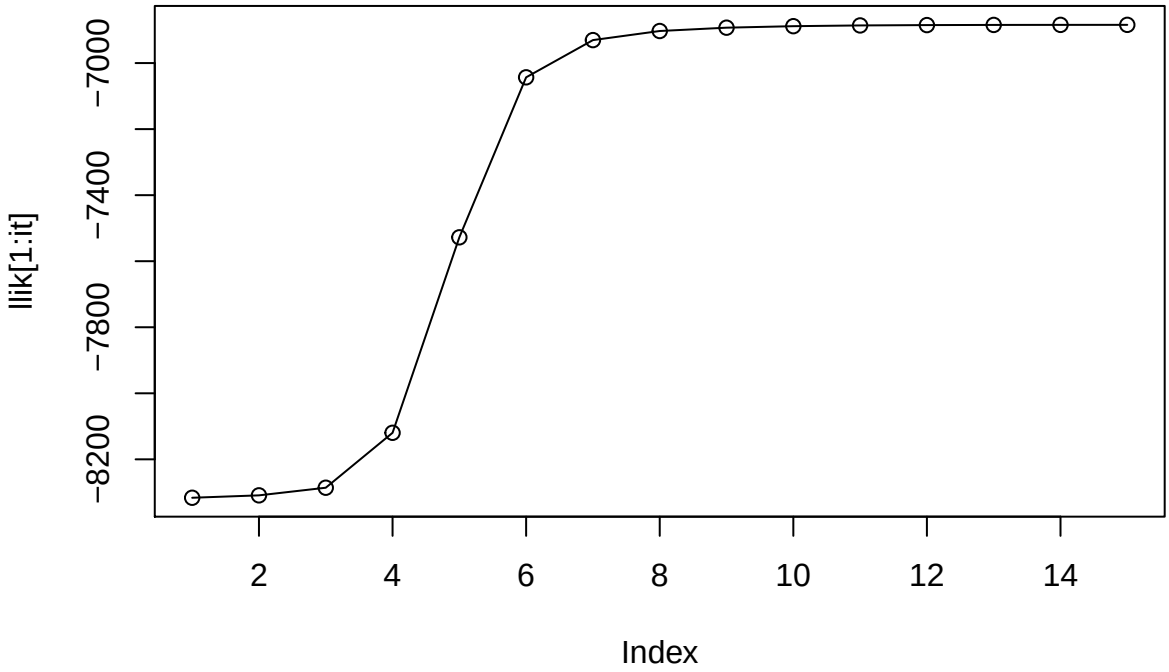

```
plot(llik[1:it], type="o")
```




```
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")
```




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



Model can be faster in case we suspect there is only 4 components, but the true number of components is 3. it can be seen that the first two components is still very similar as in the case with K = 2 and K = 3. It seems like it's the third component that now have been split up in two different components. The iteration converged after 66 iterations with an log likelihood at -6874. In order to estimate how many components there are within the data, one can use Principal Components Analysis and analyze each possible component.