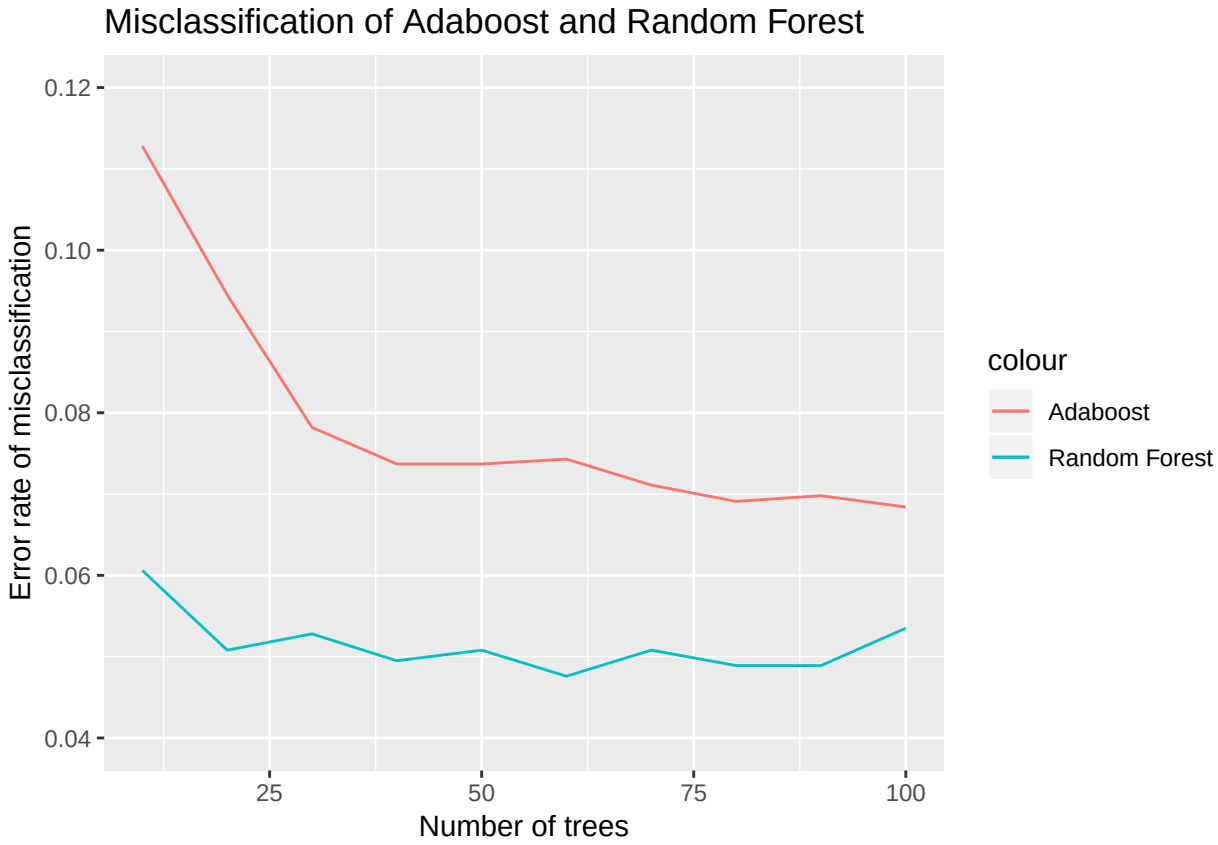
LAB 1 BLOCK 2: ENSEMBLE METHODS AND MIXTURE MODELS

Phillip Hoelscher (phiho267)

4 12 2018

1. ENSEMBLE METHODS

In this picture can we see the error rates of the misclassification of the Adaboost and Random Forest. On the y-axes is the error rate of misclassifications visible and the x-axes represents the number of trees. We considered tree numbers from 10 to 100 with a step of 10 (10,20,. . . ,100). We are able to see the error rate for "Adaboost" decreased by increasing amount of trees. The minimal value of the error is 6,84% with the tree number of 100. We also are able to see the performance of "Adaboost" increases a lot from the number 10 to 30 trees. The performance of the "Random Forest" is always better than the "Adaboost". With 100 trees does the "Random Forest" have an error rate is misclassification of 5,48% , which is lower than the best error rate of "Adaboost." The best performance of "Random Forest" is at the 6th iteration (60 trees), the error rate is 4,82%.



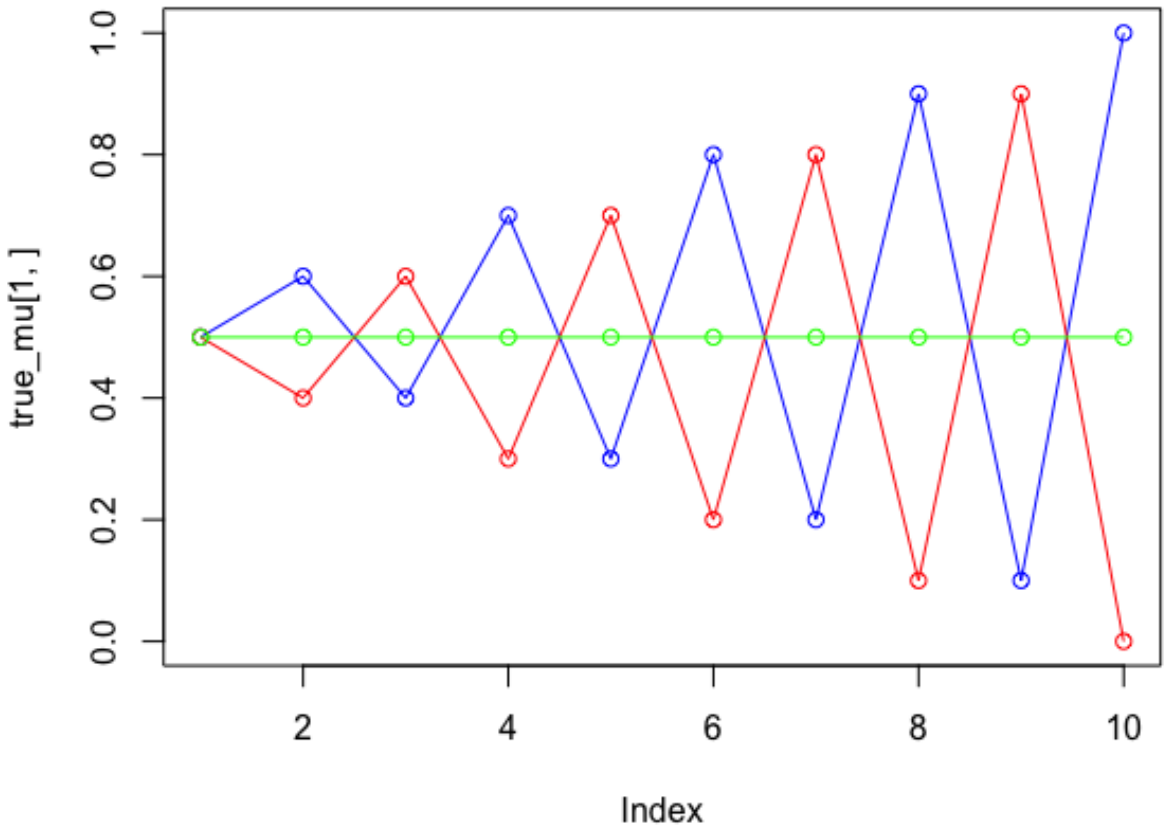


Figure 1: Plot with real values

2. MIXTURE MODELS

The task was to extend the given code and to implement the algorithm "Expectation Maximization (EM)" manually. It was specified that the K should have the values 2,3 and 4. The results are briefly explained below.

In this part we can see that after the twelve the value of log likelihood has not decreased by 0.1. From the fourth iteration to the seventh iteration the value of the log likelihood decreases the most. It can also be seen that in the first of the K=2 images it can be seen that one line is less.

Here we see three lines that are very similar to the original course. The line in the middle is not quite as straight as the original gradient. It can be seen that 26 iterations have taken place. Also from five to 7 the largest change of the log liklihood can be seen. After the eighth iteration there are only very small changes.

In the upper picture of K = 4 can be seen that only 4 lines, this picture hardly resembles that of the original. In this case the most iterations were performed, more than 40. From four to 7 the log likelihood decreases the most, afterwards only very small changes can be seen.

Altogether it has to be said that the number of components (K=3) best corresponds to the original one, which is no surprise, since the original one also contains 3 components.

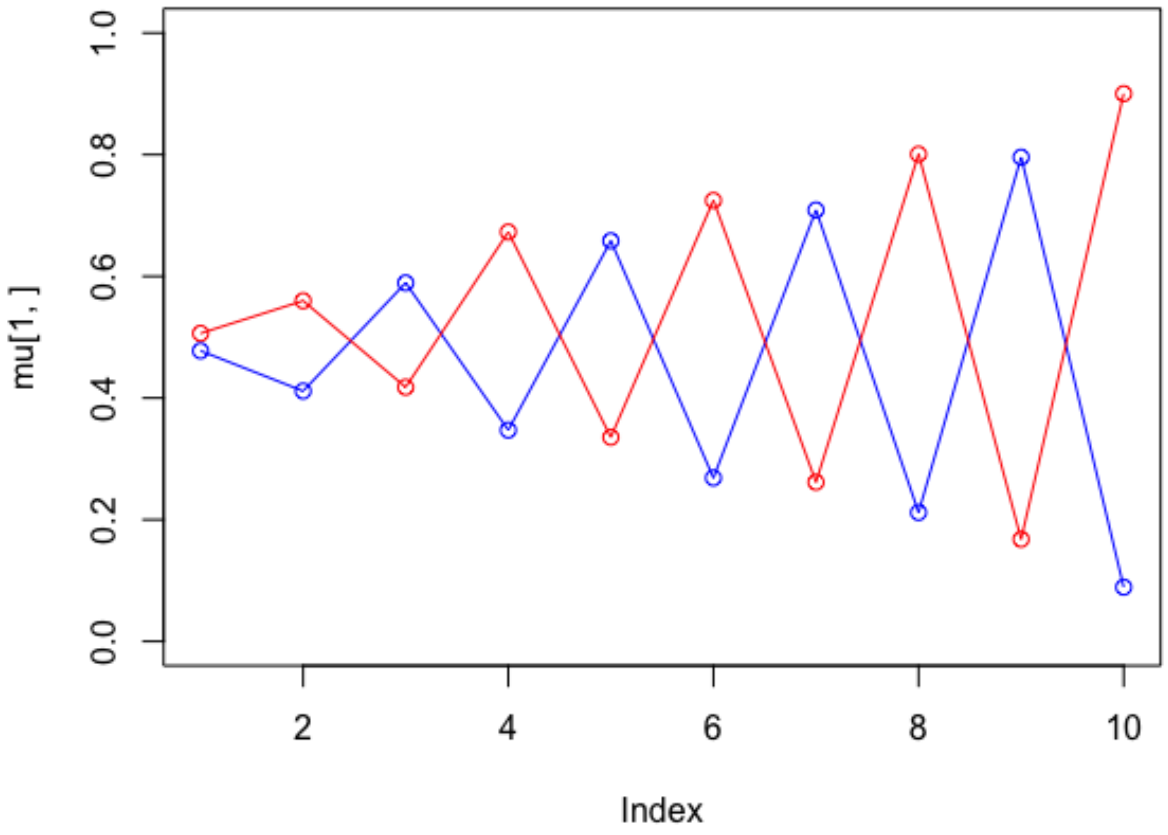


Figure 2: K = 2

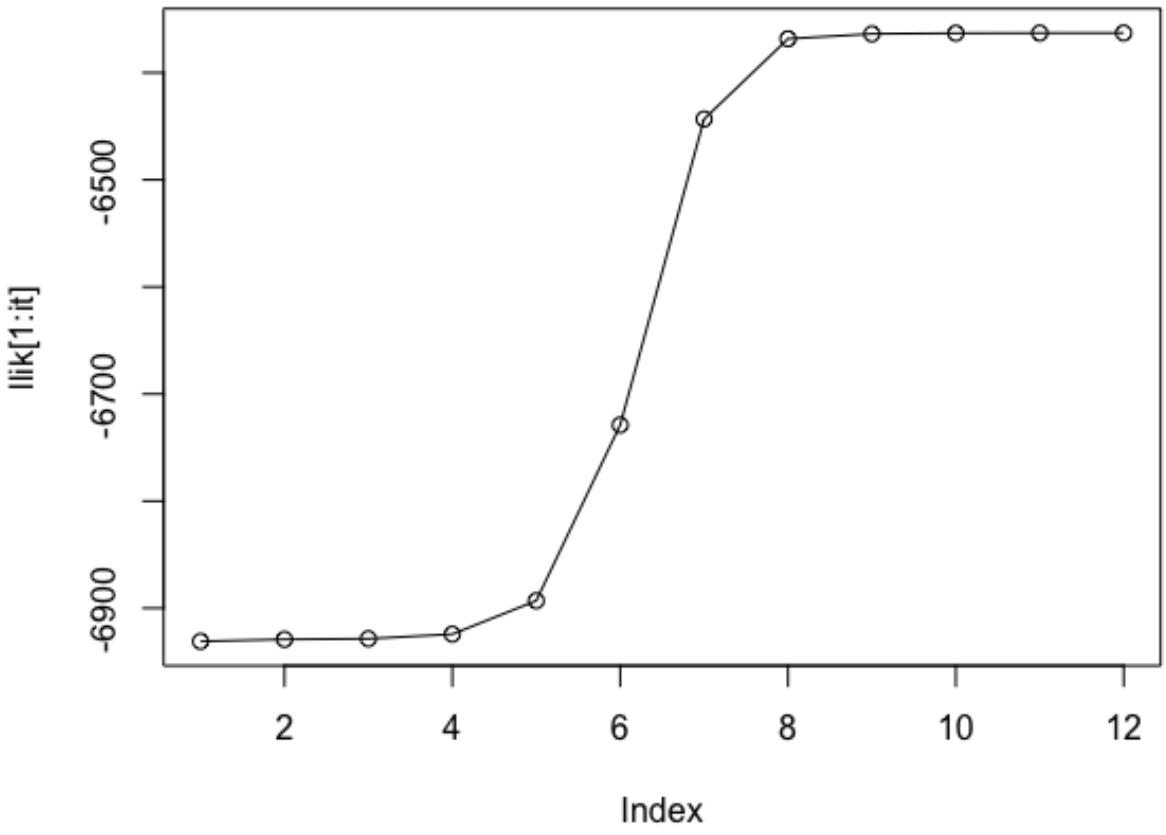


Figure 3: K = 2

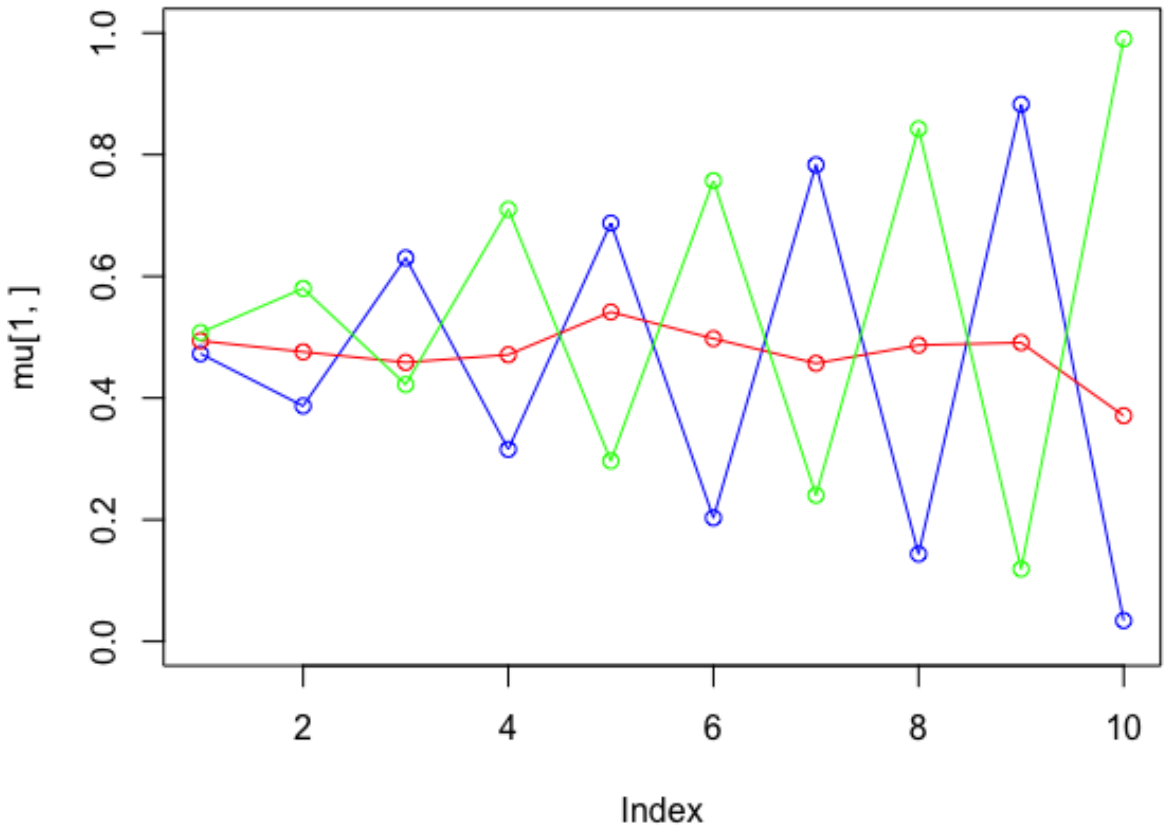


Figure 4: K = 3

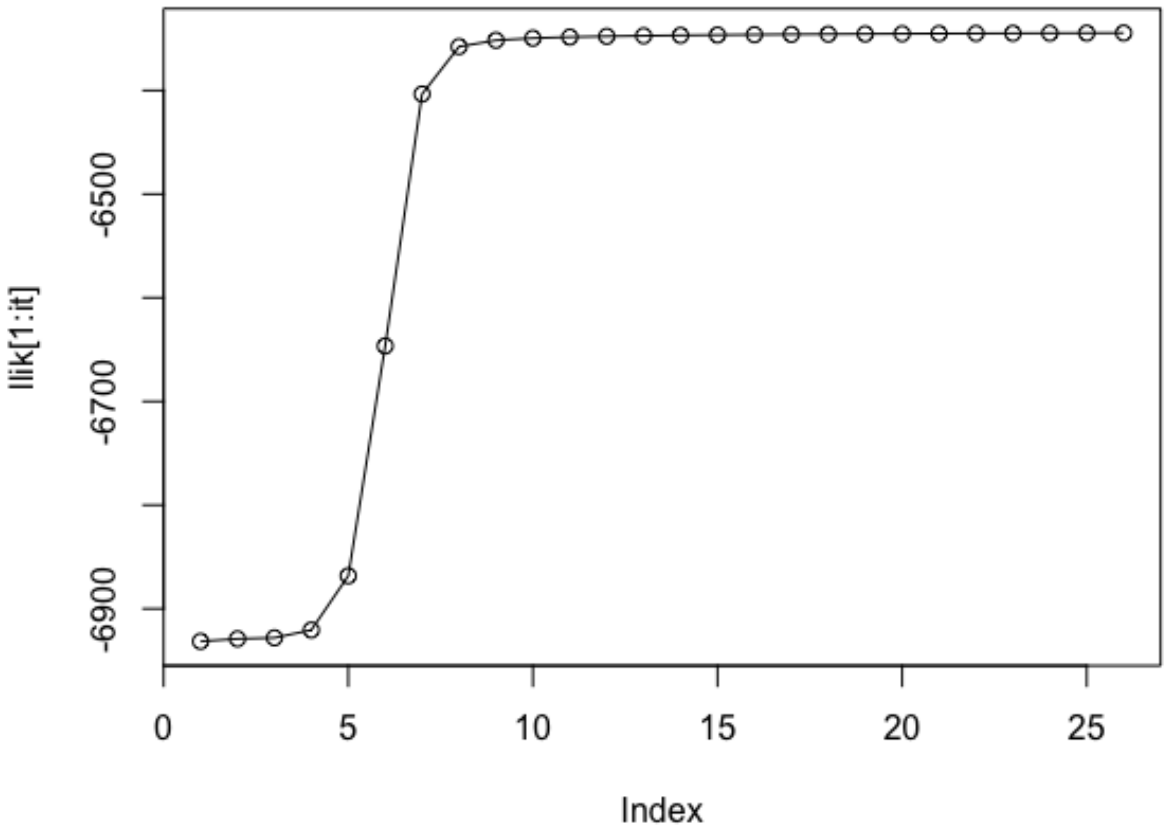


Figure 5: K = 3

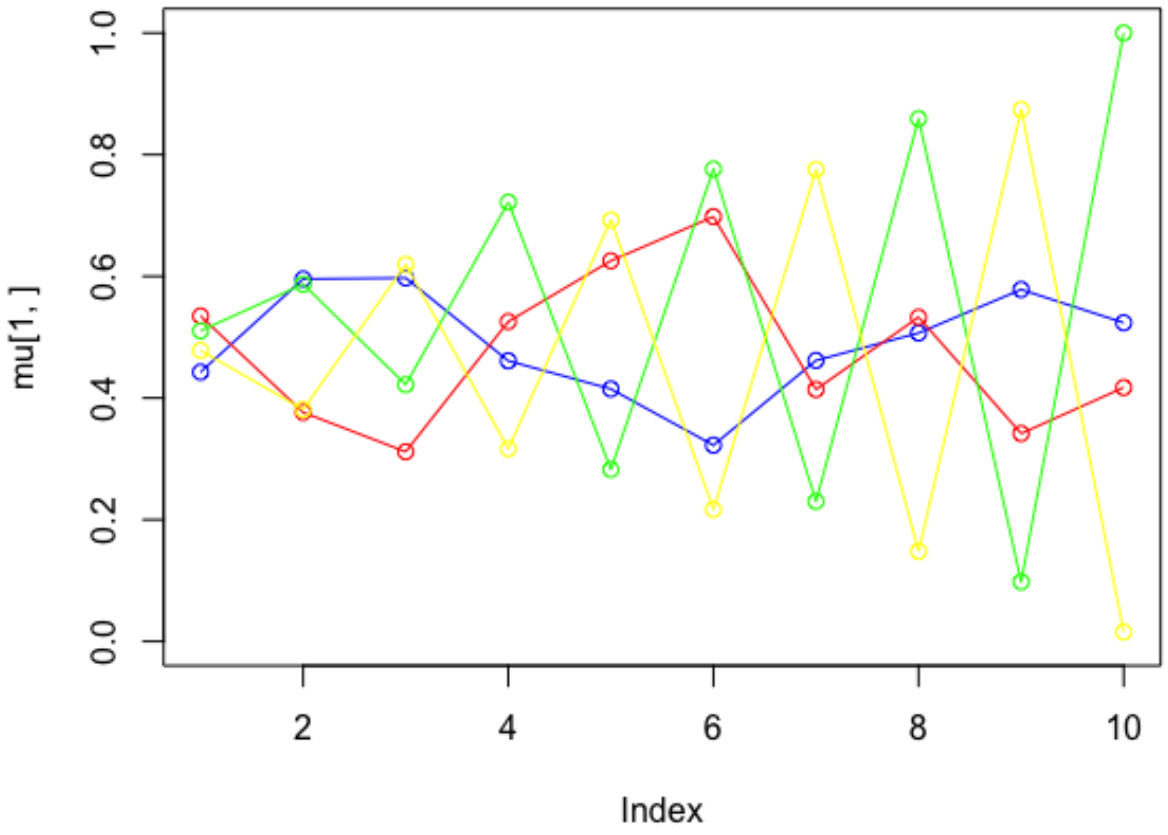


Figure 6: K = 4

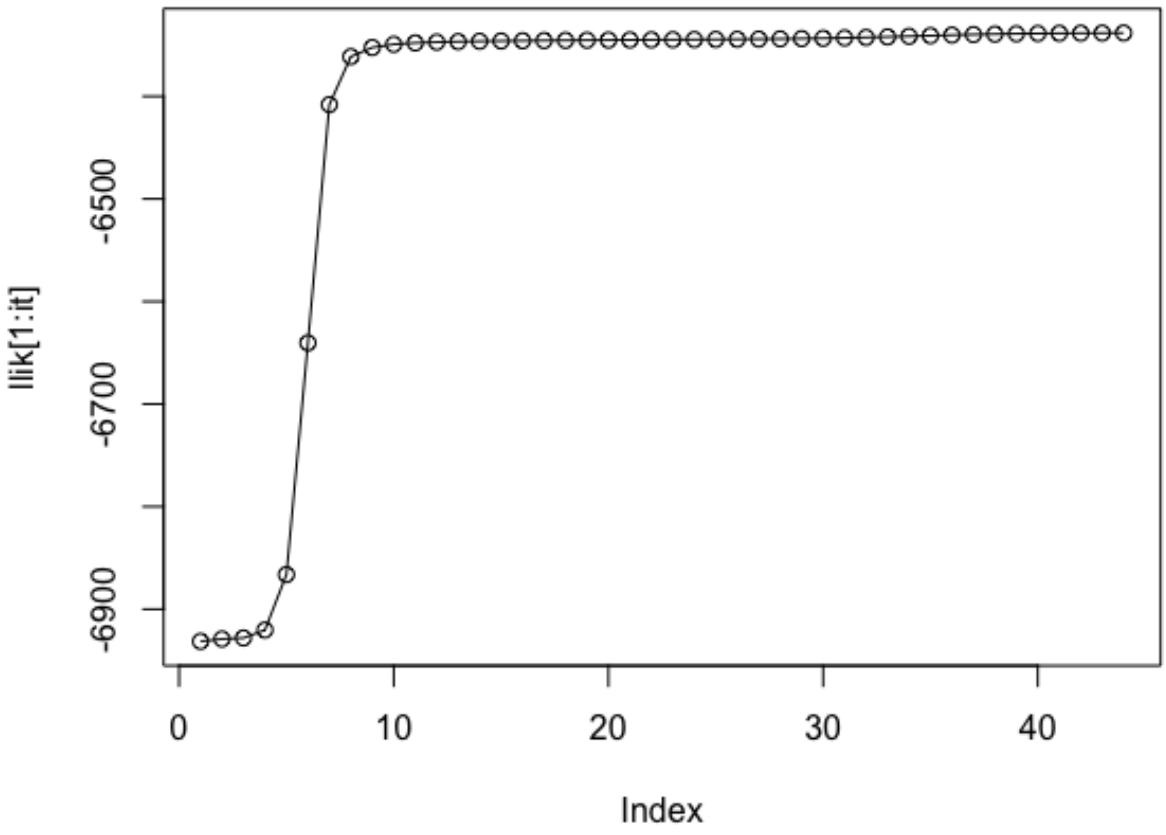


Figure 7: K = 4

```r
knitr::opts_chunk$set(echo = TRUE)
# list of the packages we need for this lab
#install.packages("randomForest")
library(randomForest)
#install.packages("mboost")
library(mboost)
#install.packages("ggplot2")
library(ggplot2)
#install.packages("likelihood")
library(likelihood)
#install.packages("SciViews")
library(SciViews)

# Ensemble Methods ####

# ---- data pre processing  ----

# set working directory

# load the data
sp <- read.csv2("spambase.csv")
sp$Spam <- as.factor(sp$Spam)

# split the data into training and test set
n=dim(sp)[1]
set.seed(1234567890)
id=sample(1:n, floor(n*2/3))
train=sp[id,]
test=sp[-id,]
# while loop to create vector with error rates - Adaboost ----

error_rate_ada_treeNr <- c() # create empty vector
i <- 10 # septs for trees
j <- 1 # index for vector
while (i < 101) {
  # create classifier of blackboost function
  classifier_ada <- blackboost(formula = Spam~.,
                               data = train,
                               family = AdaExp(),
                               control = boost_control(mstop = i))

  # predict values of new data
  y_pred_ada <- predict(classifier_ada,
                        #type = "response",
                        newdata = test[-58]) # remove the y variable

  # from log odds to classification prediction
  y_pred_ada <- ifelse(y_pred_ada > 0, 1, 0)

  # create confustion matrix
  cm_ada <- table(test$Spam, y_pred_ada)

  # calculate missclassification error rate
```

```
  error_rate_ada_treeNr[j] <- round((cm_ada[1,2] + cm_ada[2,1])/sum(cm_ada),4)

  i = i +10 # trees plus 10 (use trees 10,20,...,100)
  j = j +1 # vector index plus 1
}
# Random Forest ####

# while loop to create vector with error rates ----
error_rate_rf_treeNr <- c()
i <- 10
j <- 1
while (i < 101) {
  classifier_rf <- randomForest(x = train[-58], # remove the reposne var of the data set
                                y = train$Spam,
                                ntree = i)

  y_pred_rf <- predict(object = classifier_rf, # remove the reposne var of the data set
                       newdata = test[-58])

  cm_rf <- table(test$Spam, y_pred_rf)
  error_rate_rf_treeNr[j] <- round((cm_rf[1,2] + cm_rf[2,1])/sum(cm_rf),4)

  i = i +10
  j = j +1
}
# create plot

# color for the lines
cols <- c("Random Forest"="#f04546","Adaboost"="#3591d1")

plot_error_rate <- ggplot() + # ggplot object
  # to create a legend we have to specify the color at the aesthetics
  geom_line(aes(x = seq(from = 10, to = 100, by = 10), y = error_rate_rf_treeNr, color = "Random Forest
  geom_line(aes(x = seq(from = 10, to = 100, by = 10), y = error_rate_ada_treeNr, color = "Adaboost")) +
  ylab("Error rate of misclassification") +
  xlab("Number of trees") +
  ggtitle("Misclassification of Adaboost and Random Forest") +
  ylim(0.0400, 0.1200) # limit of the y axes - to have a better overview

# min Adaboost
min(error_rate_aba_treeNr)
which.min(error_rate_ada_treeNr)

# min Random Forest
min(error_rate_rf_treeNr)
which.min(error_rate_rf_treeNr)

# max Adaboost
max(error_rate_aba_treeNr)
which.max(error_rate_aba_treeNr)

# max Random Forest
max(error_rate_rf_treeNr)
```

```r
which.max(error_rate_rf_treeNr)
plot_error_rate

# ------ THIS IS THE CODE FOR K = 3 ------

# EM Algorithm ####
# random code
set.seed(1234567890)


# fix values
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

# create true values of pi and mu
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 mu matrix
# 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])
  }
}

# number of guessed components
K=3

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) #normalize pi
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}
```

```r
# values of pi and mu
# pi
# mu

for(it in 1:max_it) {
  # --- commend the plot - don??t want the output ---
  # 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
  # Berechnung der fraktionierten Komponentenzuordnungen

  # Your code here

  # calculation

  # numerator
  # probability
  prob_x_given_mu <- exp(x %*% log(t(mu)) + (1-x) %*% log(1-t(mu)))

  # create mu matrix with N = 1000
  pi_matrix <- matrix(pi, ncol = K, nrow = N, byrow = TRUE)

  # multiply the values
  numerator_Estep <-  prob_x_given_mu * pi_matrix

  # denominator
  denominator_Estep <- rowSums(numerator_Estep)


  # numerator/denominator
  prob_z <- numerator_Estep / denominator_Estep

  # log likelihood ----
  #Log likelihood computation.
  # Your code here
  llik[it] <- sum(log(denominator_Estep))

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

  # Stop if the lok likelihood has not changed significantly
  # Your code here
  if(it == 1){
  } else {
    if((llik[it] - llik[it-1]) < min_change){
    break
    }
  }
```

```r
  #M-step: ----
  #ML parameter estimation from the data and fractional component assignments
  # Your code here
  # new pi and mi
  pi_matrix_ml <- colSums(prob_z)/ N
  mu_ml <- (t(prob_z) %*% x)/colSums(prob_z)

  # update
  pi <- pi_matrix_ml
  mu <- mu_ml

}

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