Lab1 Block2 group report

Milda Poceviciute, Henrik Karlsson, Ugurcan Lacin

28 November 2017

1. Ensemble Methods

When multiple machine learning algorithms is being used to obtain better results than a single model could do, it's called ensemble methods. The task in this laboration is to predict if a email is spam or not, using Ada Boost classifier (combining boosting and decision trees) and a Random Forest (combining decision trees and bagging). The variable spam is a categorical variable telling the true answer, wheter an email is spam or not. The ada loss function is given by:

$$ln(1 + exp(-yt))$$

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).

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


data <- read.csv("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,]
```

Algorithms will be used with different tree numbers from 10 to 100 increasing by 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.

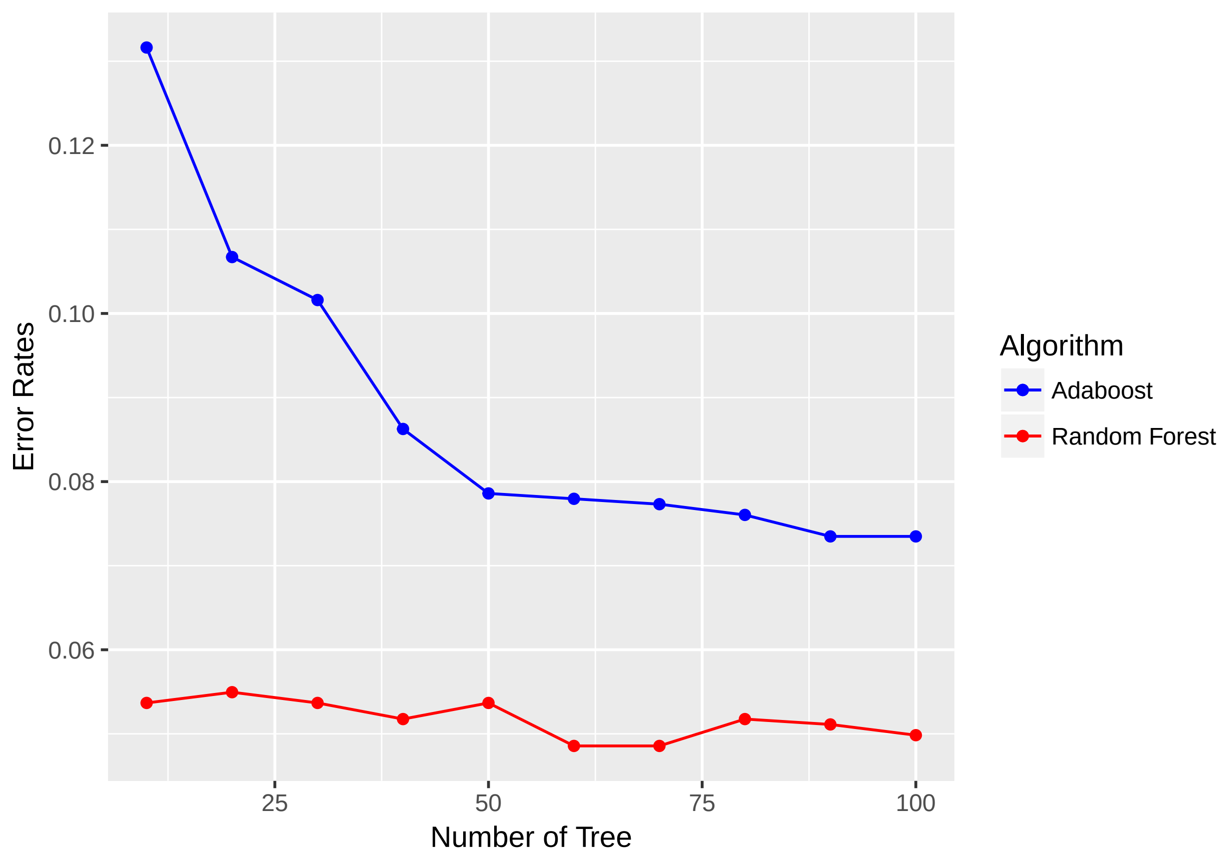
```r
numberOfTrees <- seq(10,100,10)
randomForestErrorRates <- c()
for(i in 1:10){
  numberOfTree <- numberOfTrees[i]
  resRandomForest <- randomForest(as.factor(Spam) ~ .,data = train,ntree=numberOfTree)
  pred <- predict(resRandomForest,newdata=test)
  tab <- table(pred,test$Spam)
  misclassificationRate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

```r
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.

```r
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_line(mapping = aes( x=numberOfTrees ,y = mx[1, ], colour = "red"),data = rfdata) +
geom_point(mapping = aes( x=numberOfTrees ,y = mx2[1, ], colour = "blue"),data = adadata) +
geom_line(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"))
```



The graph above visualize the missclassification rate for Ada Boost (Blue) and Random Forest (Red) for each number of trees. The random forest performs better than the Ada Boost. The error rate is higher when the lower amount of trees is used for classifying the emails. As the number of trees increases the error rate reduces in both algorithms. But this decreasing is taking place with a very low slope and irregularity in the random forest case. 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 trees, it 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.

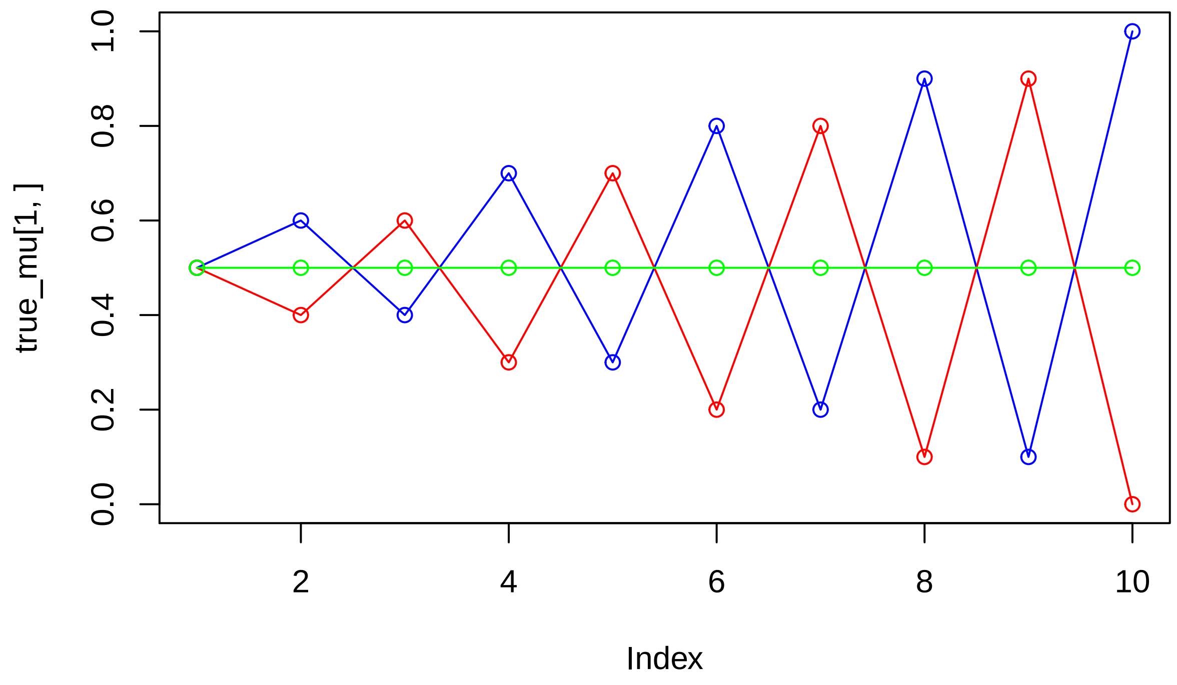
2. MIXTURE MODELS

In this task the EM algorithm is implemented for mixtures of multivariate Benouilli distributions. Mixture models are probablisitic models that estimate whether a data point is belonging to a subpopulation within an overall population.

EM algorithm with 3 components

Firstly, a random data is generated for testing EM algorithm with 3 components. As the true number of components is 3, we can expect this model to perform well.

```r
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 = 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 graph above shows what the true distribution parameters (μ) ad true weights (π) are of the distribution from which my data is sampled. After randomly setting initial weights and parameters of Bernoulli distribution, we get these starting π (the initial weights) and μ (the initial parameter of Bernoulli distribution) for my EM algorithm:

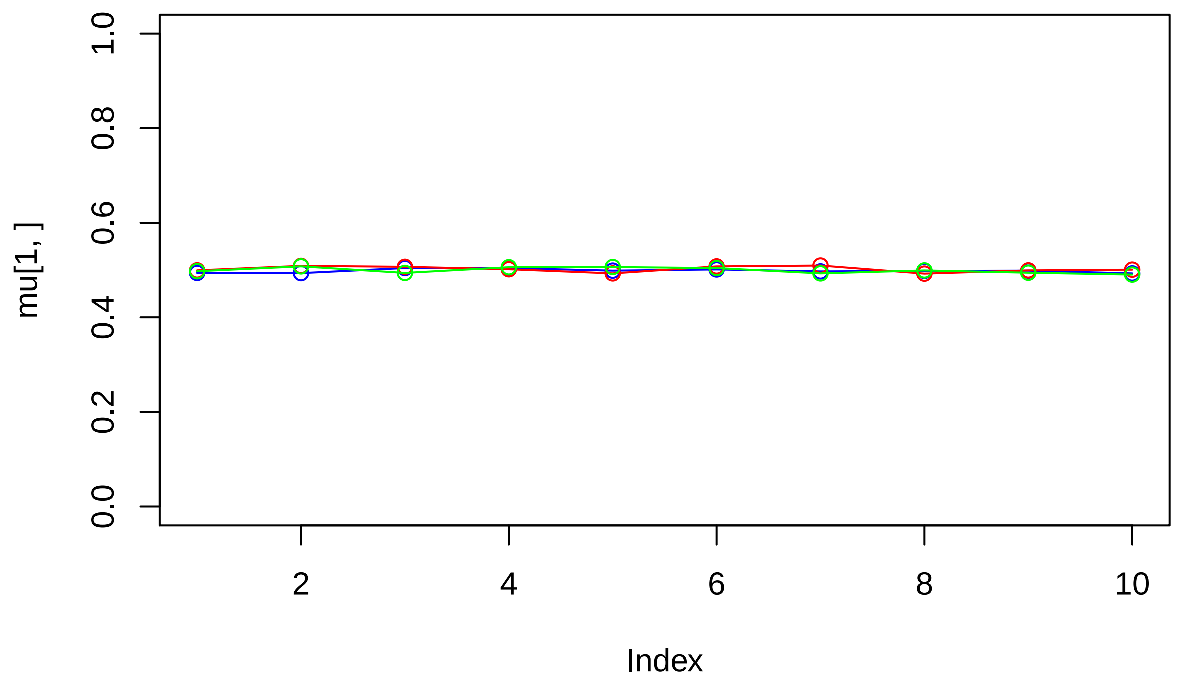
```
pi
```

```
## [1] 0.3326090 0.3336558 0.3337352
```

```
mu
```

```
##           [,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
```

```
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")
```



It is clear from the plot that initial Bernoulli distributions have very different parameters from the true values.

The EM-algorithm is an iterative model with two major steps that is being performed over and over. The first step is the Expectation step, where each point in the data is estimated with a probability of belong to one of the *components* or subpopulations in the data. In the first iteration an arbitrary point for start is choosen.

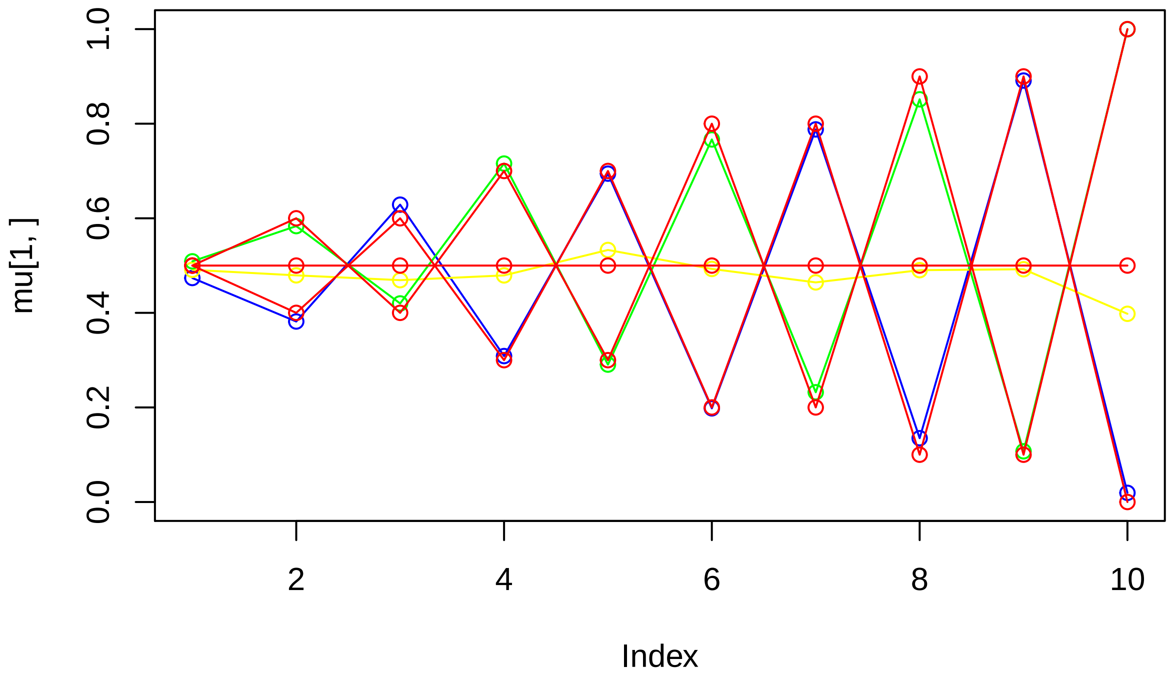
The estimated values in the E-step is then used to maximize the log likelihood of belong to a certain component. The new log likelihood function is now used to re-estimate the probablility of each point belonging to a specific *component*.

The code for my EM algorithm is provided below:

```r
# EM algorithm
mu_plots <- list()
for (it in 1:max_it) {
    # E-step: Computation of the fractional component assignments
    temp <- 1
    temp0 <- 0

    numenator <- c()

    for (n in 1:N) {
        for (k in 1:K) {
            temp <-  dbinom(x[n, ], size = 1, mu[k, ])
            numenator[k] <- pi[k] * prod(temp)
        }

        temp0 <- sum(numenator)
        for (k in 1:K) {
            z[n, k] <- numenator[k] / temp0
        }

    }

    z_check <- apply(z, 1, function(a) {
        sum(a)
    })

    #Log likelihood computation.

    temp2 <- 0
    temp3 <- 0
    logp <- 0

    for (n in 1:N) {
        for (k in 1:K) {
            #slide 7 from Lecture1bBlock2 - log likelihood calculation
            temp1 <- 0
            for (i in 1:D) {
                temp1 <- temp1 + x[n, i] * log(mu[k, i]) + (1 - x[n, i]) * log(1 - mu[k, i])
            }
            temp2 <- temp2 + z[n, k] * log(pi[k])
            temp3 <- temp3 + z[n, k] * temp1
        }
    }

    llik[it] <- temp3 + temp2
    # cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    # flush.console()
    # Stop if the log likelihood has not changed significantly
    if (it >= 2) {
        if (abs(llik[it - 1] - llik[it]) < min_change) {
            break
        }
    }

    #M-step: ML parameter estimation from the data and fractional component assignments
    # MLE pi
    sum_z <- apply(z, 2, function(a) {
        sum(a)
    })
    pi <- (sum_z) / N
    # MLE mu
    mu <- t(z) %*% x

    for (k in 1:K) {
        mu[k, ] <- mu[k, ] / (sum_z[k])
    }
}
```

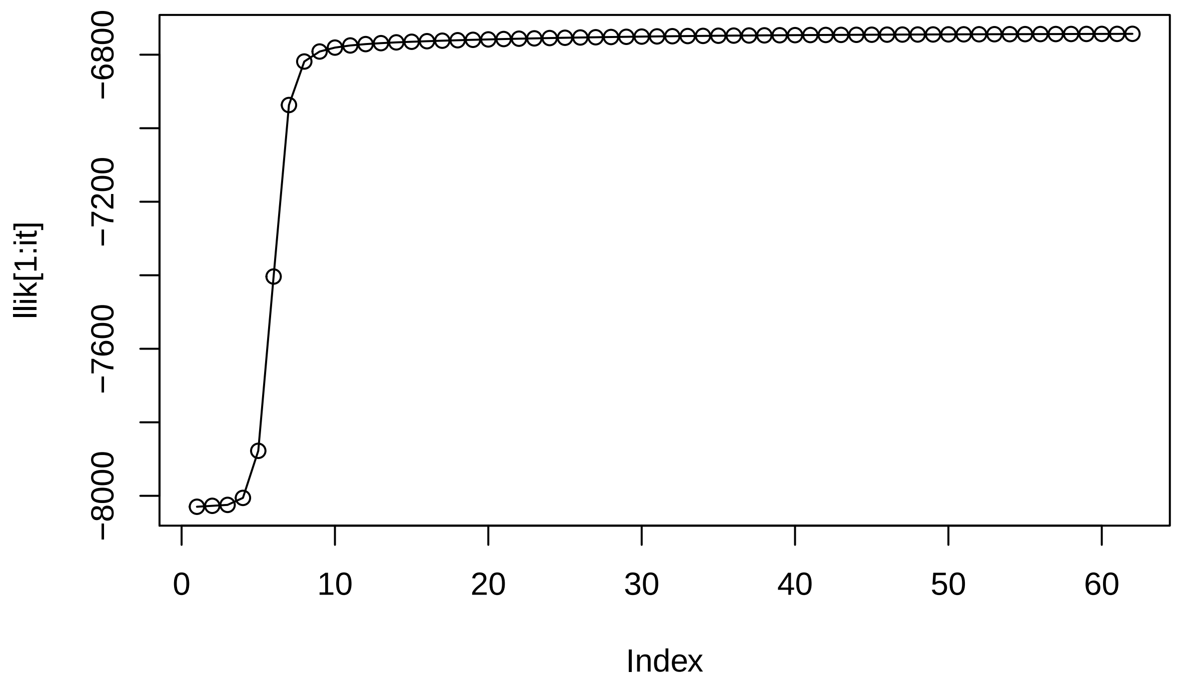
The EM algortihm stopped after 62 iterrations as the change in the log likelihood was below 0.1, having a log likelihood at -6743.326. The π and μ values converged towards the true values. The final π values are close to the true $\pi = 1/3$.

```
## integer(0)
```



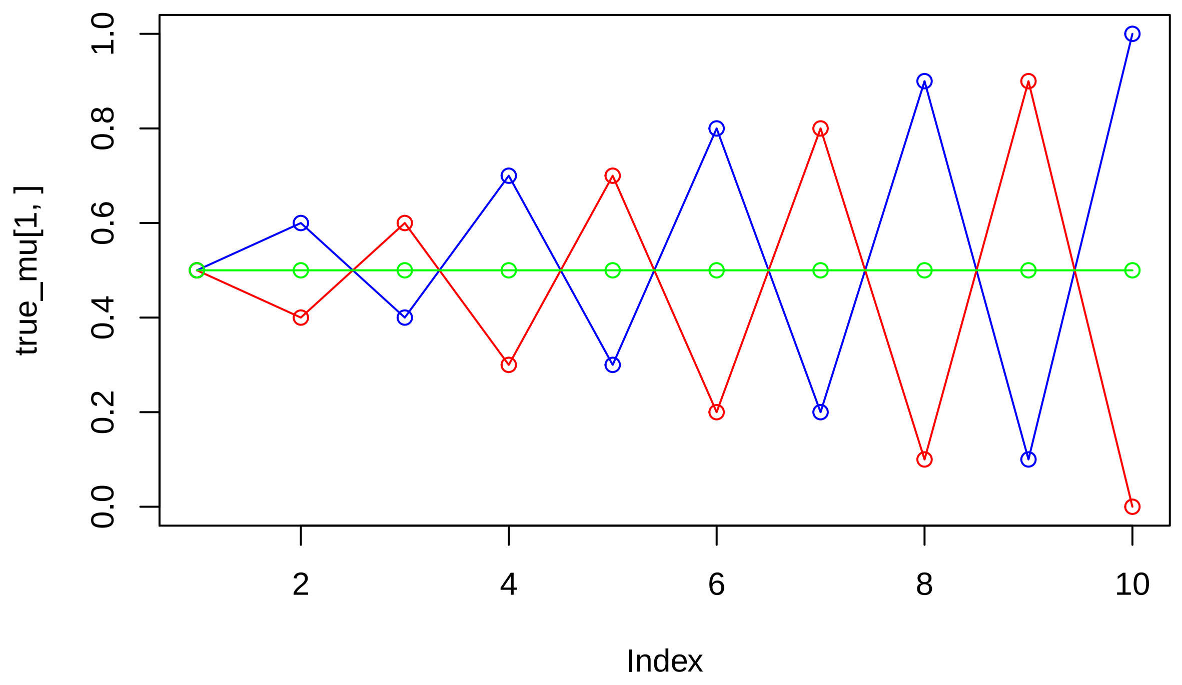
The plot shows the final μ (blue, green and yellow lines) choice by EM algorithm. They also seem to be pretty close to the true μ values (represented by the red lines).

The likelihood plot below, shows how the log likelihod is increasing with each iteration. At first the model improves dramatically - the curve goes up quickly. From around 8th iterration the improvement of the model stabilises and slows down.



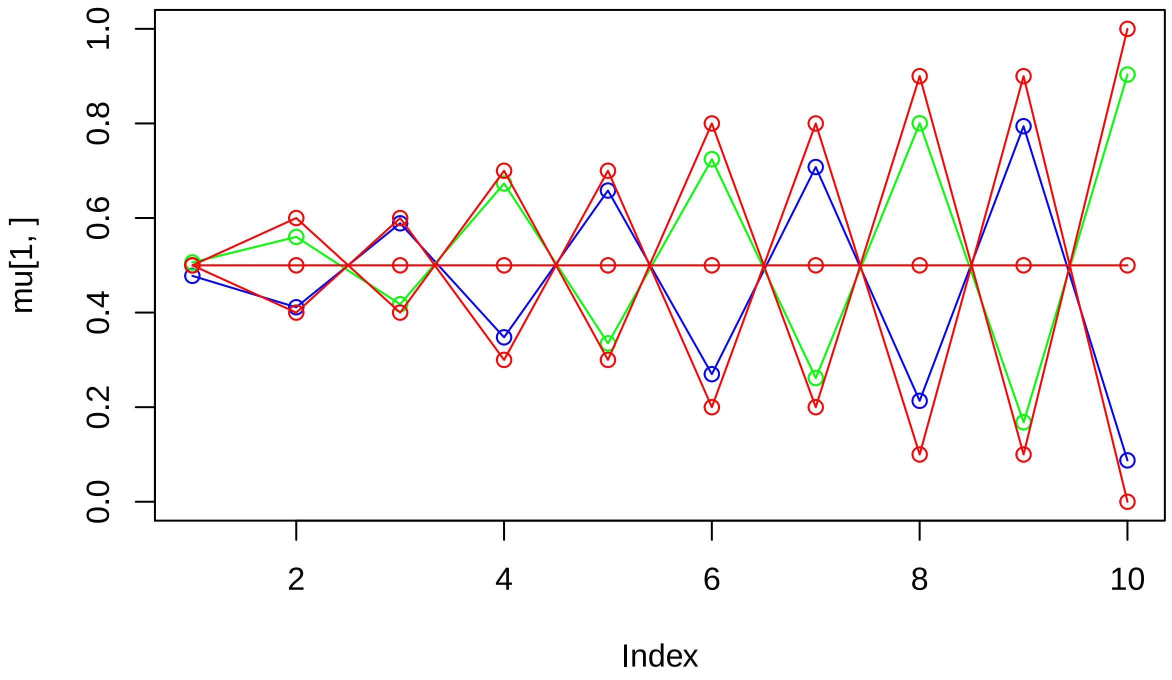
EM algorithm with 2 components

The expected number of components is set to 2 and the same EM algorithm is run. Because we try to fit EM with smaller amount of models in mixture than the true data consists, we can expect this model to be under-fitted and perform poorly.

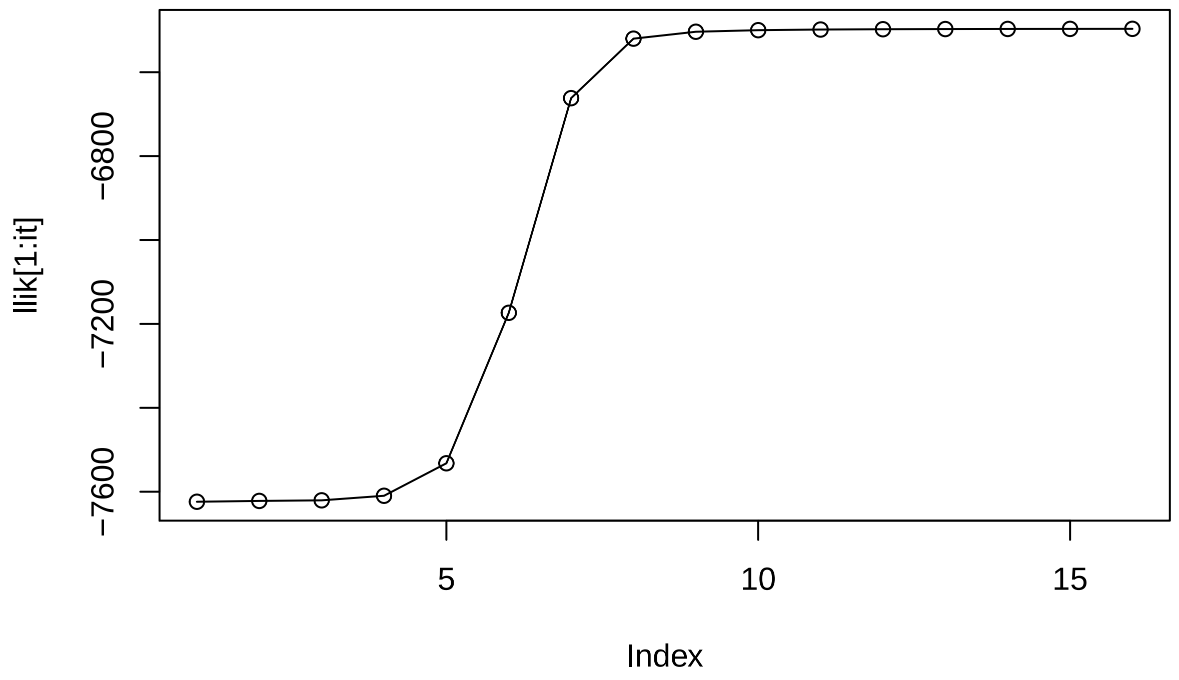


This time, the EM algortihm stopped after 16 iterrations as the change in the log likelihood was below 0.1. The plot shows the final μ choice by EM algorithm (in blue and green colours) as well as the true μ values (in red colour). Clearly, this model is underfitted as the complexity of the true data is not explained - the third component of the true mixture model is not captured by EM. However, the two components that were assumed by this EM model are fitted quite well.

```
## integer(0)
```

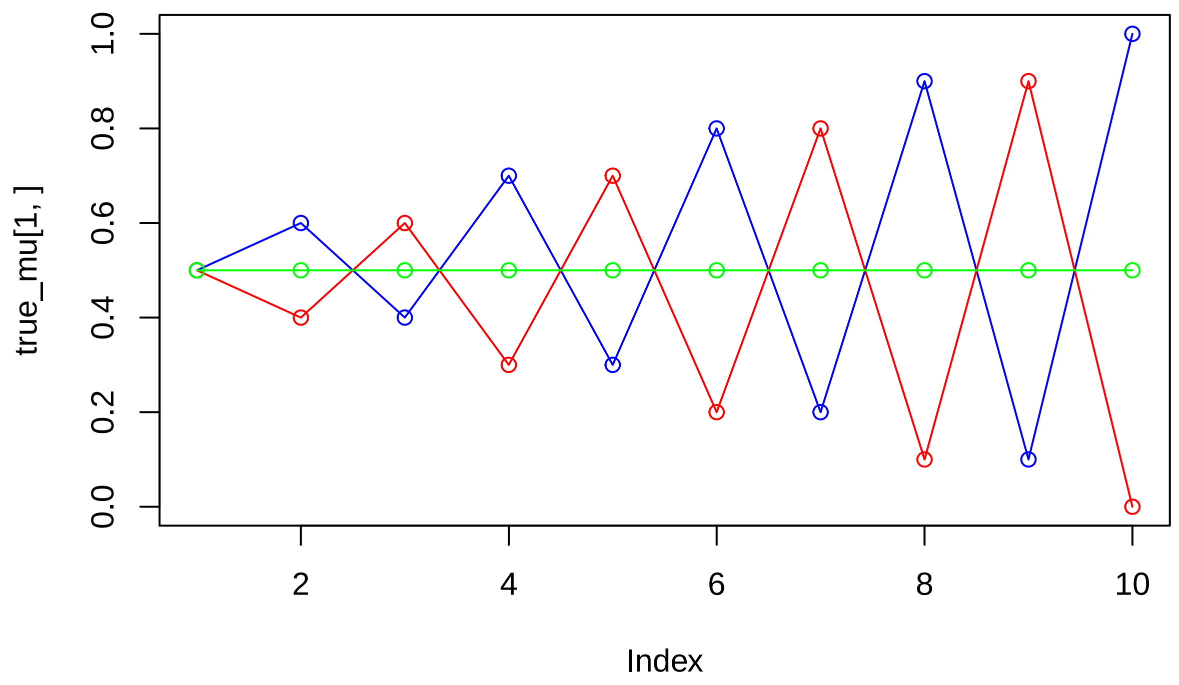


The likelihood plot below, shows how the log likelihod is decreasing with each iteration. At first the model improves dramatically - the curve goes down steeply.



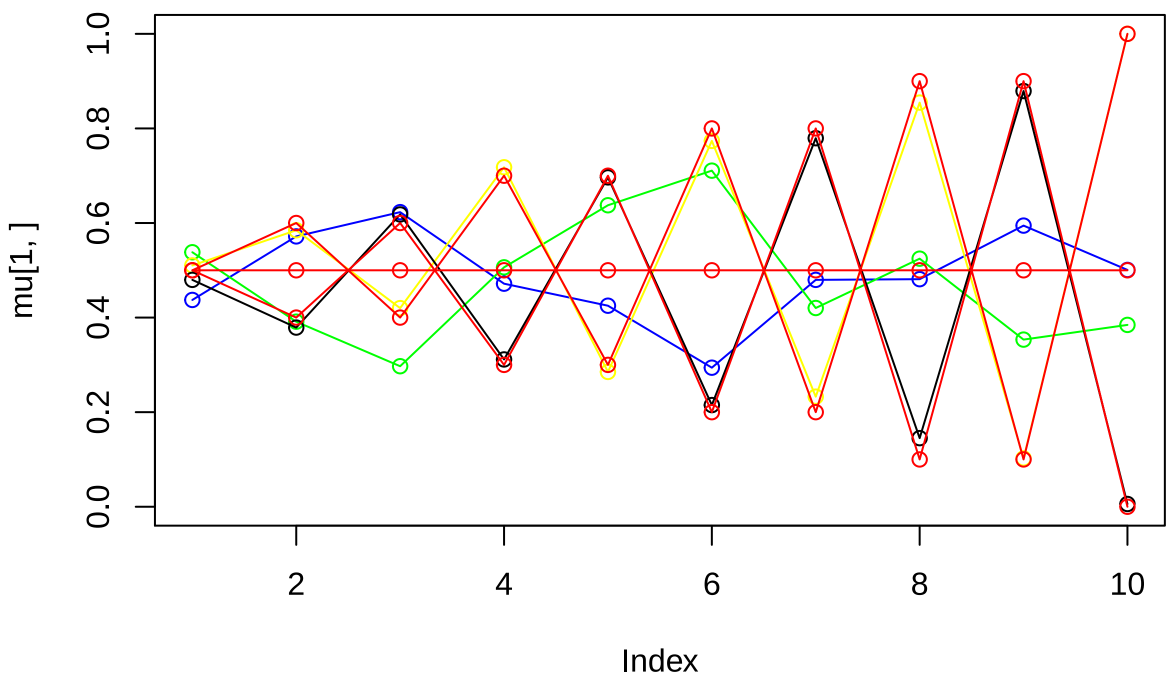
EM algorithm with 4 components

The expected number of components is set to 4 and the same EM algorithm is fitted. Because we try to fit EM with larger number of models in the mixture than the true data consists, we can expect this model to be over-fitted and perform worse the the EM algorithm with 3 components.

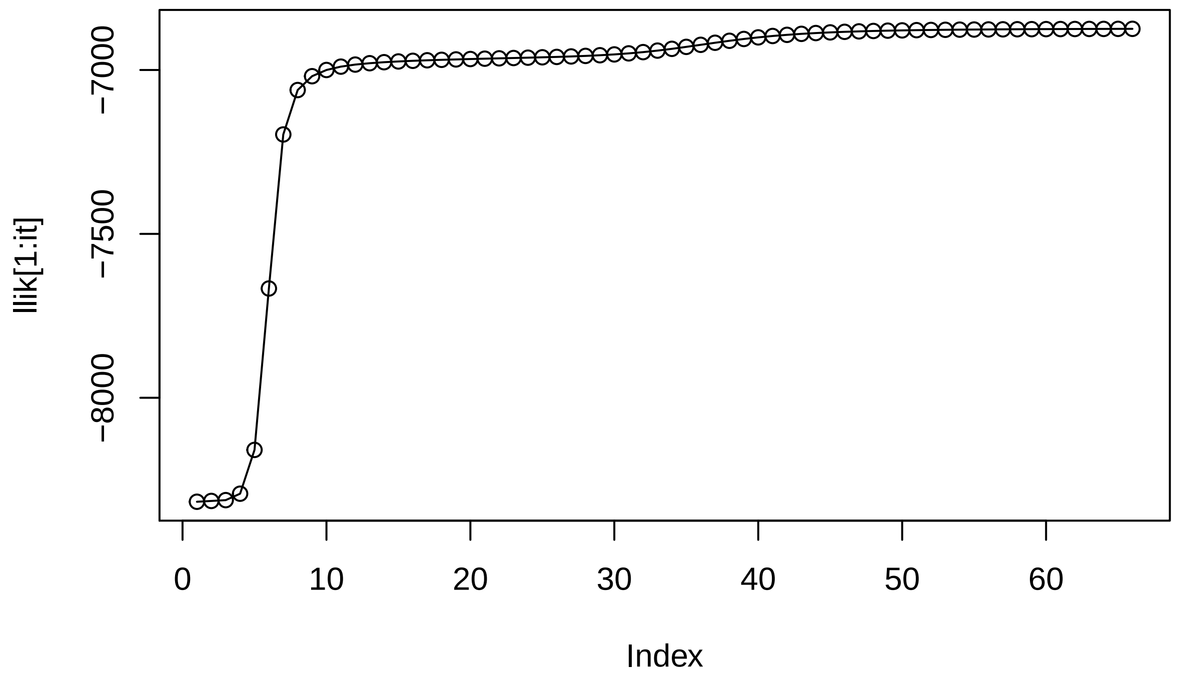


This time, the EM algortihm stopped after 66 iterrations as the change in the log likelihood was below 0.1. The plot shows the final μ choice by EM algorithm (in blue and green colours) as well as the true μ values (in red colour). Clearly, this model is overfitted and captures the random noise as well as the real uncetainty of the data. From the plot below we can see that the estimated components by this model are quite different from the true values.

```
## integer(0)
```



The likelihood plot below, shows how the log likelihod is increasing with each iteration. At first the model improves dramatically - the curve goes down steeply.



Conclusion

The best fitted model is EM algorithm with 3 components. After 62 iterrations the predicted μ and π values by this model were closest to the real values. The EM algorithm with 2 components under-fitted the data while EM algorithm with 4 components over-fitted. In the first case, not all complexity of the real data is captured.

Appendix

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


data <- read.csv("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,]
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$Spam)
  misclassificationRate <- 1-sum(diag(tab))/sum(tab)
  randomForestErrorRates <- cbind(randomForestErrorRates,misclassificationRate)
}
AdaboostErrorRates <- c()
for(i in 1:10){
  numberOfTree <- numberOfTrees[i]
  resAdaboost <- blackboost(as.factor(Spam) ~ ., data = train,
                            control = boost_control(mstop= numberOfTree),family = AdaExp())
  pred <- predict(resAdaboost,newdata=test,type="class")
  tab <- table(pred,test$Spam)
  misclassificationRate <- 1-sum(diag(tab))/sum(tab)
  AdaboostErrorRates <- cbind(AdaboostErrorRates,misclassificationRate)
}
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_line(mapping = aes( x=numberOfTrees ,y = mx[1, ], colour = "red"),data = rfdata) +
geom_point(mapping = aes( x=numberOfTrees ,y = mx2[1, ], colour = "blue"),data = adadata) +
geom_line(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"))
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)
}
pi
mu
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")

# EM algorithm
mu_plots <- list()
for (it in 1:max_it) {
    # E-step: Computation of the fractional component assignments
    temp <- 1
    temp0 <- 0

    numenator <- c()

    for (n in 1:N) {
        for (k in 1:K) {
            temp <-  dbinom(x[n, ], size = 1, mu[k, ])
            numenator[k] <- pi[k] * prod(temp)
        }

        temp0 <- sum(numenator)

        for (k in 1:K) {
            z[n, k] <- numenator[k] / temp0
        }

    }

    z_check <- apply(z, 1, function(a) {
        sum(a)
    })

    #Log likelihood computation.

    temp2 <- 0
    temp3 <- 0
    logp <- 0

    for (n in 1:N) {
        for (k in 1:K) {
            #slide 7 from Lecture1bBlock2 - log likelihood calculation
            temp1 <- 0
            for (i in 1:D) {
                temp1 <- temp1 + x[n, i] * log(mu[k, i]) + (1 - x[n, i]) * log(1 - mu[k, i])
            }
            temp2 <- temp2 + z[n, k] * log(pi[k])
            temp3 <- temp3 + z[n, k] * temp1
        }
    }

    llik[it] <- temp3 + temp2
    # cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    # flush.console()
    # Stop if the log likelihood has not changed significantly
    if (it >= 2) {
        if (abs(llik[it - 1] - llik[it]) < min_change) {
            break
        }
    }

    #M-step: ML parameter estimation from the data and fractional component assignments
    # MLE pi
    sum_z <- apply(z, 2, function(a) {
        sum(a)
    })
    pi <- (sum_z) / N
    # MLE mu
    mu <- t(z) %*% x

    for (k in 1:K) {
        mu[k, ] <- mu[k, ] / (sum_z[k])
    }
}

plot(mu[1,],
         type = "o",
         col = "blue",
         ylim = c(0, 1)) +
    points(mu[2,], type = "o", col = "yellow")
    points(mu[3,], type = "o", col = "green")
    points(true_mu[1, ], type = "o", col = "red")
points(true_mu[2, ], type = "o", col = "red")
points(true_mu[3, ], type = "o", col = "red")

logp <- c()
logp[1] <- llik[it]
plot(llik[1:it], type = "o")
```

```
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 = 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)
}

# EM algorithm
for (it in 1:max_it) {

    # E-step: Computation of the fractional component assignments
    temp <- 1
    temp0 <- 0

    numenator <- c()

    for (n in 1:N) {
        for (k in 1:K) {
            temp <-  dbinom(x[n, ], size = 1, mu[k, ])
            numenator[k] <- pi[k] * prod(temp)
        }
        temp0 <- sum(numerator)

        for (k in 1:K) {
            z[n, k] <- numerator[k] / temp0
        }

    }

    z_check <- apply(z, 1, function(a) {
        sum(a)
    })
    #Log likelihood computation.
    temp2 <- 0
    temp3 <- 0
    logp <- 0

    for (n in 1:N) {
        for (k in 1:K) {
            #slide 7 from Lecture1bBlock2
            temp1 <- 0
            for (i in 1:D) {
                temp1 <- temp1 + x[n, i] * log(mu[k, i]) + (1 - x[n, i]) * log(1 - mu[k, i])
            }
            temp2 <- temp2 + z[n, k] * log(pi[k])
            temp3 <- temp3 + z[n, k] * temp1
        }
    }

    llik[it] <- temp3 + temp2
    # cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    # flush.console()
    # Stop if the log likelihood has not changed significantly
    if (it >= 2) {
        if (abs(llik[it - 1] - llik[it]) < min_change) {
            break
        }
    }

    #M-step: ML parameter estimation from the data and fractional component assignments
    # MLE pi
    sum_z <- apply(z, 2, function(a) {
        sum(a)
    })
    pi <- (sum_z) / N
    # MLE mu
    mu <- t(z) %*% x

    for (k in 1:K) {
        mu[k, ] <- mu[k, ] / (sum_z[k])
    }
}

plot(mu[1,],
```

```r
         type = "o",
         col = "blue",
         ylim = c(0, 1)) +
    points(mu[2,], type = "o", col = "green")
points(true_mu[1, ], type = "o", col = "red")
points(true_mu[2, ], type = "o", col = "red")
points(true_mu[3, ], type = "o", col = "red")

logp[2] <- llik[it]
plot(llik[1:it], type = "o")

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 = 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)
}

# EM algorithm
for (it in 1:max_it) {
```

```r
# E-step: Computation of the fractional component assignments

temp <- 1
temp0 <- 0

numenator <- c()

for (n in 1:N) {
    for (k in 1:K) {
        temp <-  dbinom(x[n, ], size = 1, mu[k, ])
        numenator[k] <- pi[k] * prod(temp)
    }

    temp0 <- sum(numenator)

    for (k in 1:K) {
        z[n, k] <- numenator[k] / temp0
    }

}

z_check <- apply(z, 1, function(a) {
    sum(a)
})
#Log likelihood computation.

temp2 <- 0
temp3 <- 0
logp <- 0

for (n in 1:N) {
    for (k in 1:K) {
        #slide 7 from Lecture1bBlock2
        temp1 <- 0
        for (i in 1:D) {
            temp1 <- temp1 + x[n, i] * log(mu[k, i]) + (1 - x[n, i]) * log(1 - mu[k, i])
        }
        temp2 <- temp2 + z[n, k] * log(pi[k])
        temp3 <- temp3 + z[n, k] * temp1
    }
}

llik[it] <- temp3 + temp2
# cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
# flush.console()
# Stop if the log likelihood has not changed significantly
if (it >= 2) {
    if (abs(llik[it - 1] - llik[it]) < min_change) {
        break
    }
}

#M-step: ML parameter estimation from the data and fractional component assignments
```

```r
    # MLE pi
    sum_z <- apply(z, 2, function(a) {
        sum(a)
    })
    pi <- (sum_z) / N
    # MLE mu
    mu <- t(z) %*% x

    for (k in 1:K) {
        mu[k, ] <- mu[k, ] / (sum_z[k])
    }
}

plot(mu[1,],
         type = "o",
         col = "blue",
         ylim = c(0, 1)) +
    points(mu[2,], type = "o", col = "green")
    points(mu[3,], type = "o", col = "yellow")
    points(mu[4,], type = "o", col = "black")
    points(true_mu[1, ], type = "o", col = "red")
    points(true_mu[2, ], type = "o", col = "red")
    points(true_mu[3, ], type = "o", col = "red")

logp[3] <- llik[it]
plot(llik[1:it], type = "o")
```