Block-2,Lab1

Sridhar

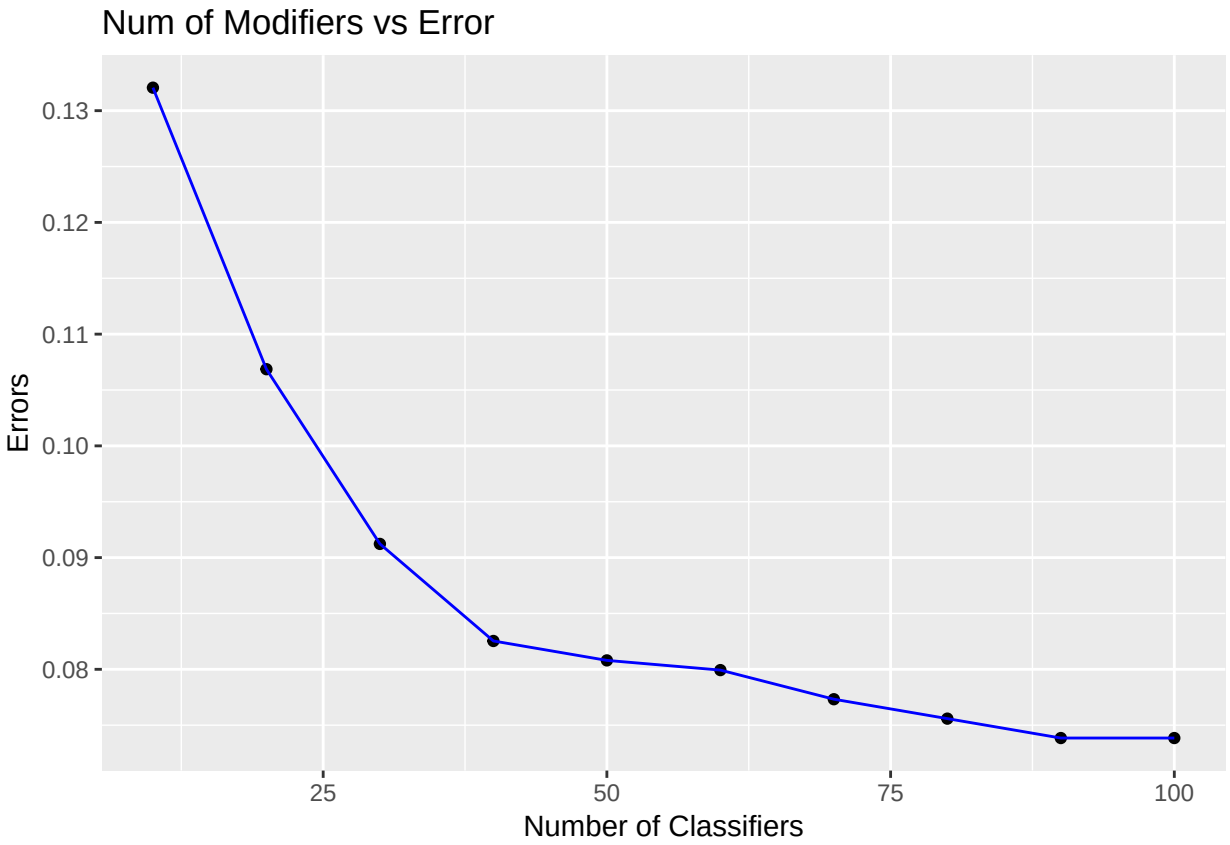
24 November 2018

Contents

Assignment 1

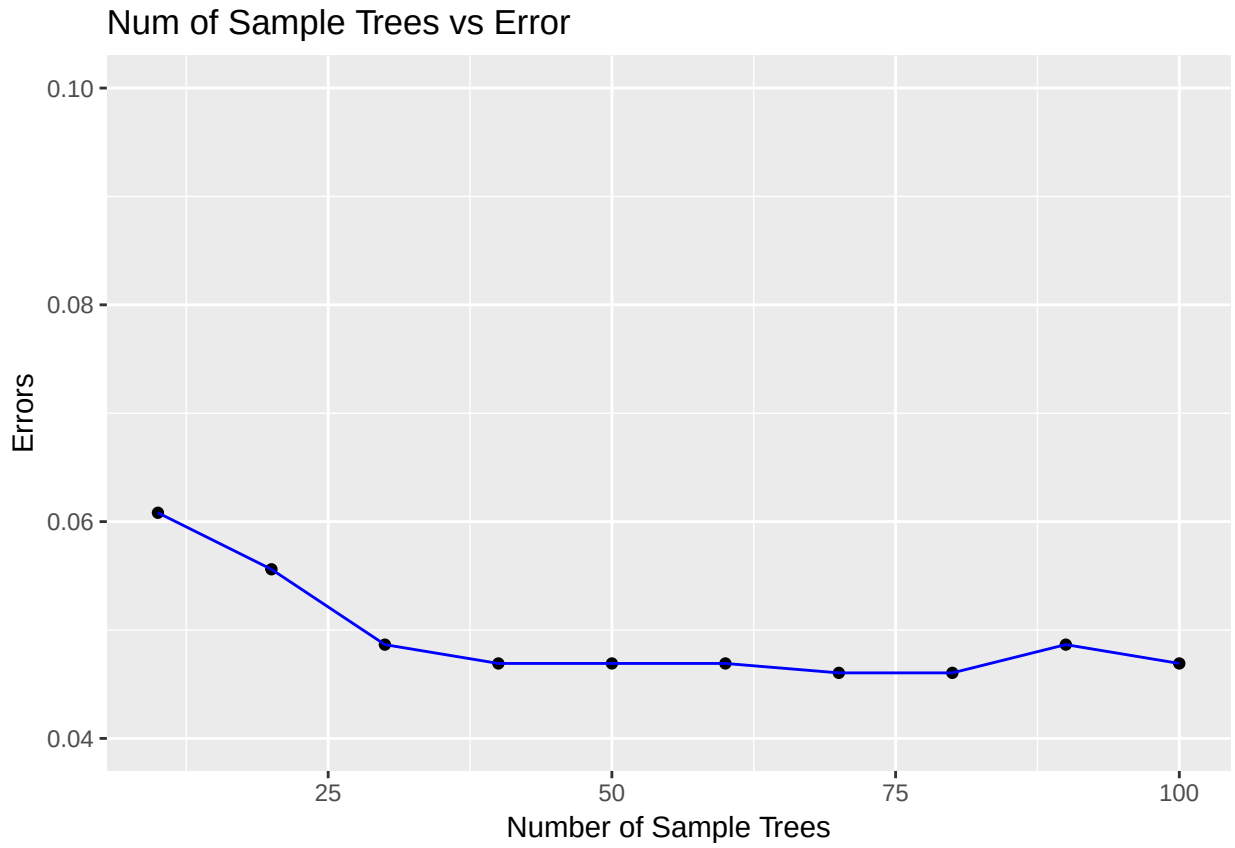
1 Ensemble Method

1 Adaboost

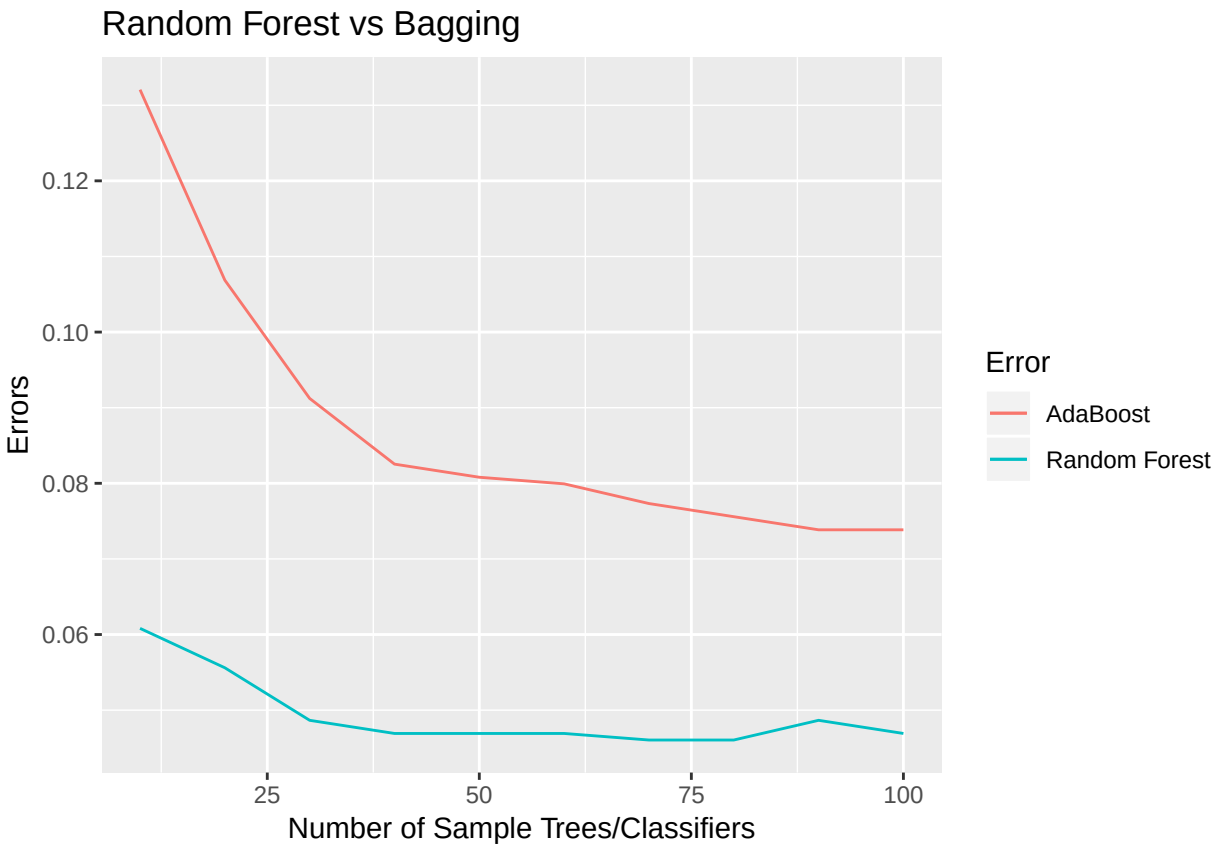


The graph shows clearly that the error rate decreases as the number of iterations increases, thus increasing the accuracy of the model. The Ada Boost algorithm gives higher weight in the next iteration to the parameters that were predicted poorly. This is the reason the models accuracy increases as the number of iterations increase.

2 Random Forest



For random forest we need to find the number of bags needed iteratively. We select the bag size that has the least error rate, and this varies with data. We can see from this plot that the error rate is high initially when the number of bags were vary few(below 25), but after that it is almost stabalized. The optimal number of bags for this dataset would be somewhere around 60 to 70.



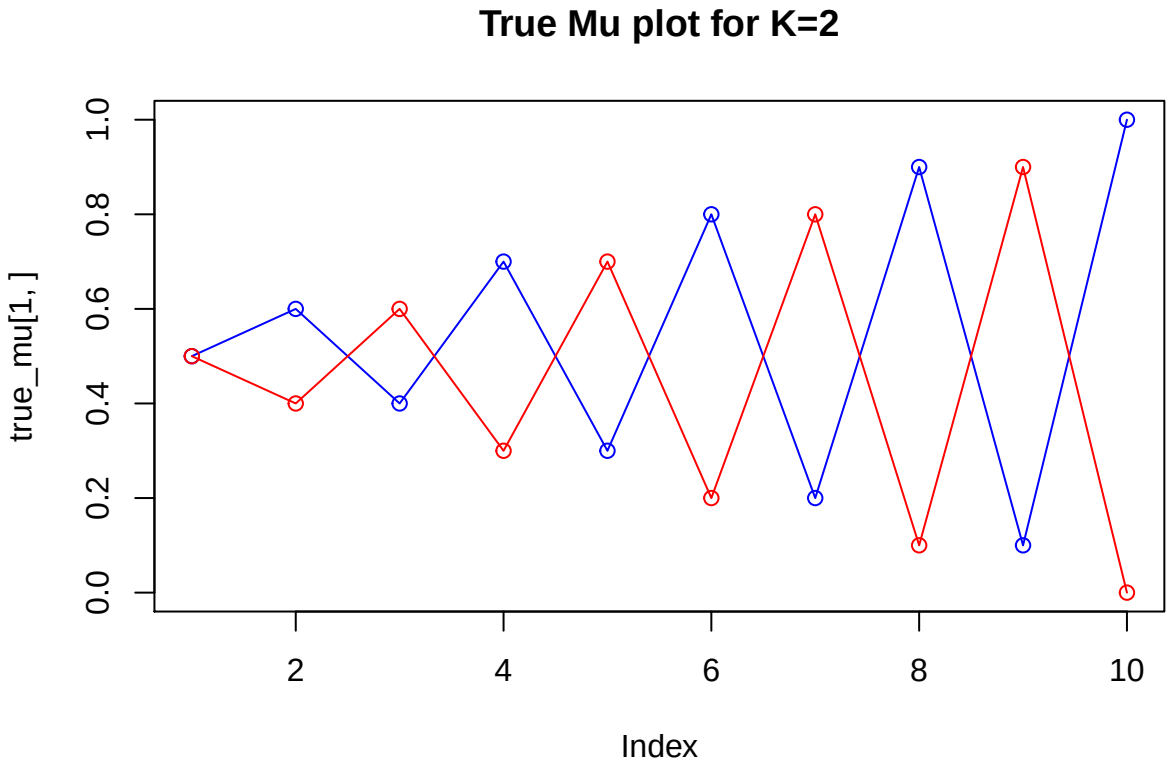
This plot is a comparison between Ada Boost vs Random Forest for the same number of sampling. Random forest performs better than Ada Boost in this case, but I would say it just gets lucky for this data. Ada Boost is a boosting technique in which each example is learning from other which in turn boosts the learning.

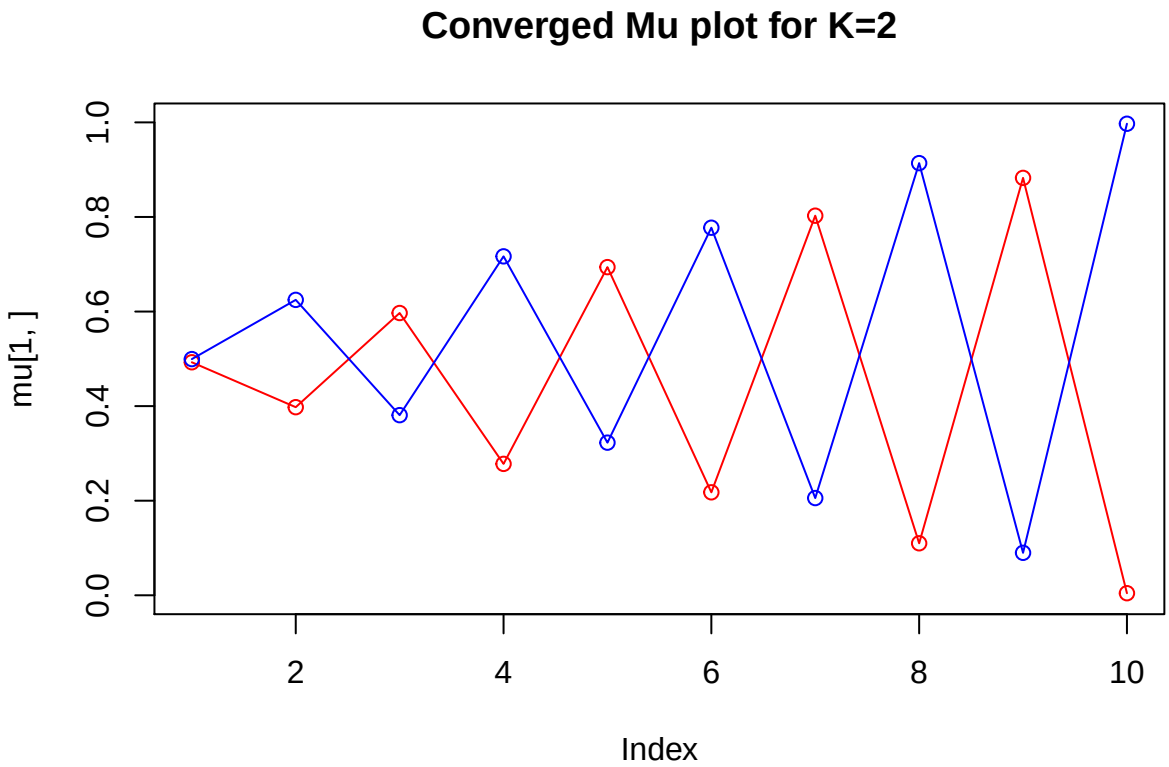
Assignment 2

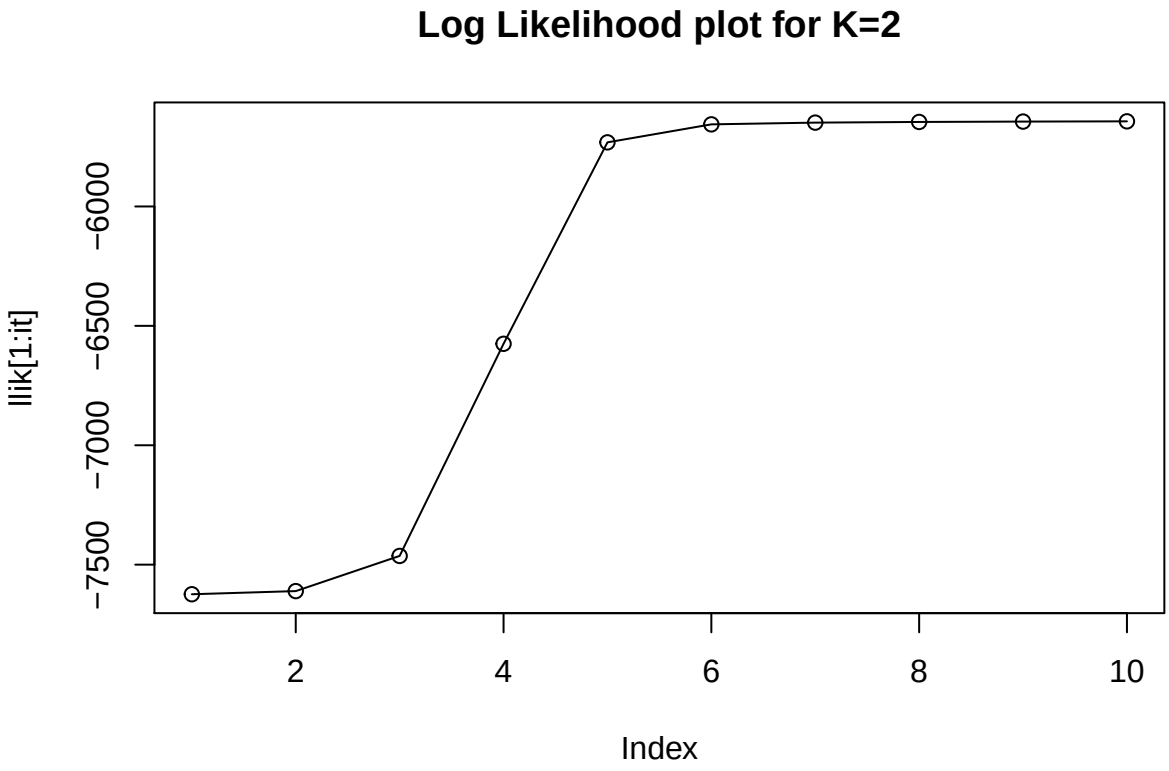
1 EM Function for Bernoulli Distribution

```
#Function for EM algorithm
em_alg = function(pi, mu, llik, min_change, maxit=100){
  old = 0
  for(it in 1:max_it) {
    # E-step: Computation of the fractional component assignments
    total <- matrix(0, nrow = nrow(x), ncol= 1)
    z = matrix(1, nrow = nrow(x), ncol = length(pi))
    for (i in 1:length(pi)) {
      for (j in 1:ncol(x)) {
        z[, i] = z[, i] * dbinom(x[ ,j], 1, mu[i,j])
      }
      z[, i] = z[, i] * pi[i]
      total= total + z[,i]
    }
    for(i in 1:length(pi)){
      z[,i] = z[,i]/total
    }
    # #Log likelihood computation.
    q = matrix(1, nrow = length(total), ncol = 1)
    for(i in 1:nrow(mu)){
      temp = matrix(1, nrow = nrow(x), ncol = 1)
      for(j in 1:ncol(x)){
        temp = temp*(mu[i,j]^x[,j])*((1-mu[i,j])^(1-x[,j]))
      }
      q = q * (temp*pi[i])^(z[,i])
    }
    llik[it] = sum(log(q))
    # Stop if the log likelihood has not changed significantly
    if(abs(old-llik[it])<min_change){
      break
    }
    else{
      old = llik[it]
    }
    #M-step: ML parameter estimation from the data and fractional component assignments
    new_pi = colSums(z)/nrow(x)
    new_mu = matrix(nrow = nrow(mu), ncol = ncol(mu))
    for (i in 1:length(pi)){
      nm = sum(z[,i])
      new_mu[i,] = colSums(x*z[,i])/nm
    }
    mu = new_mu
    pi = new_pi
  }
  return(list(pi, mu, llik, it))
}
```

2 K = 2


True Mu plot for K=2
true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

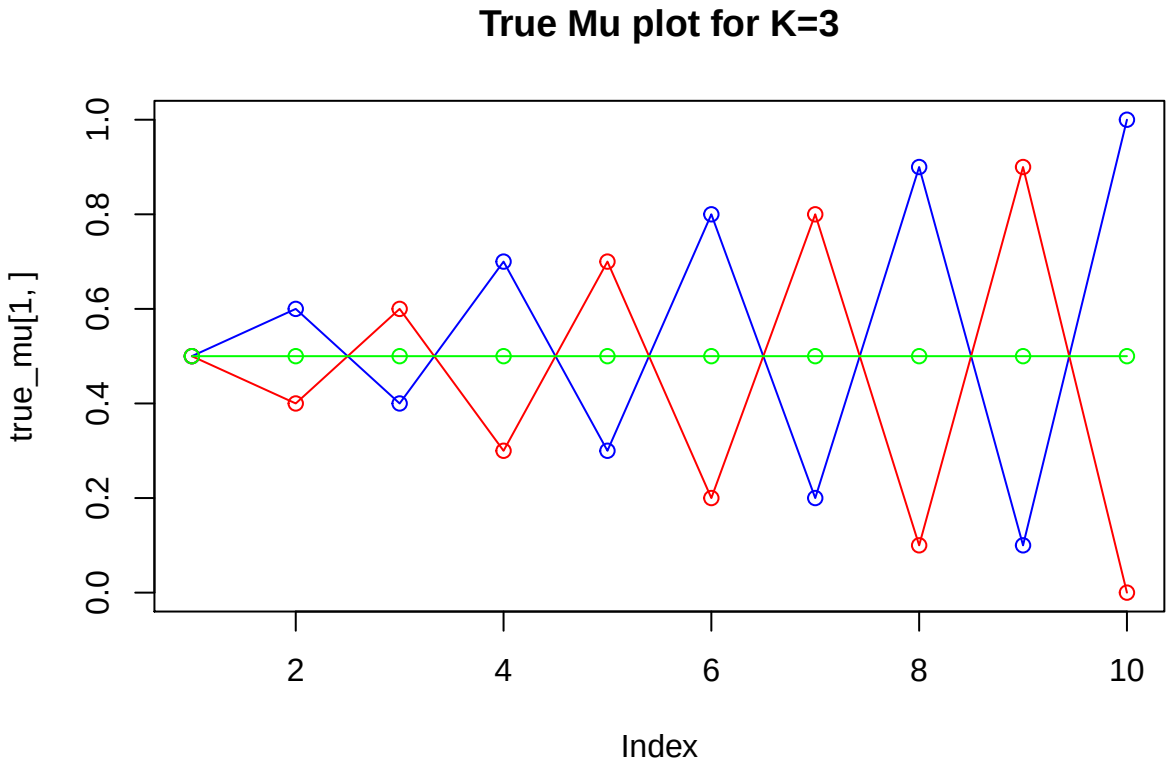

Converged Mu plot for K=2
mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

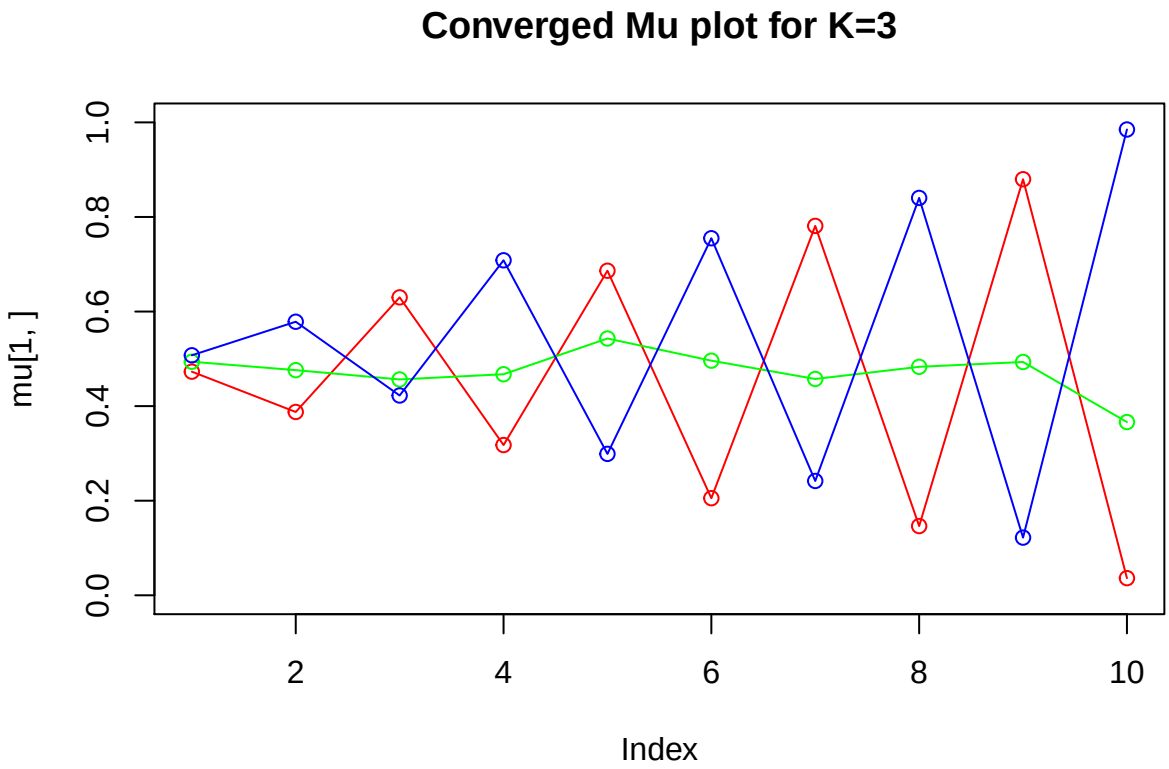


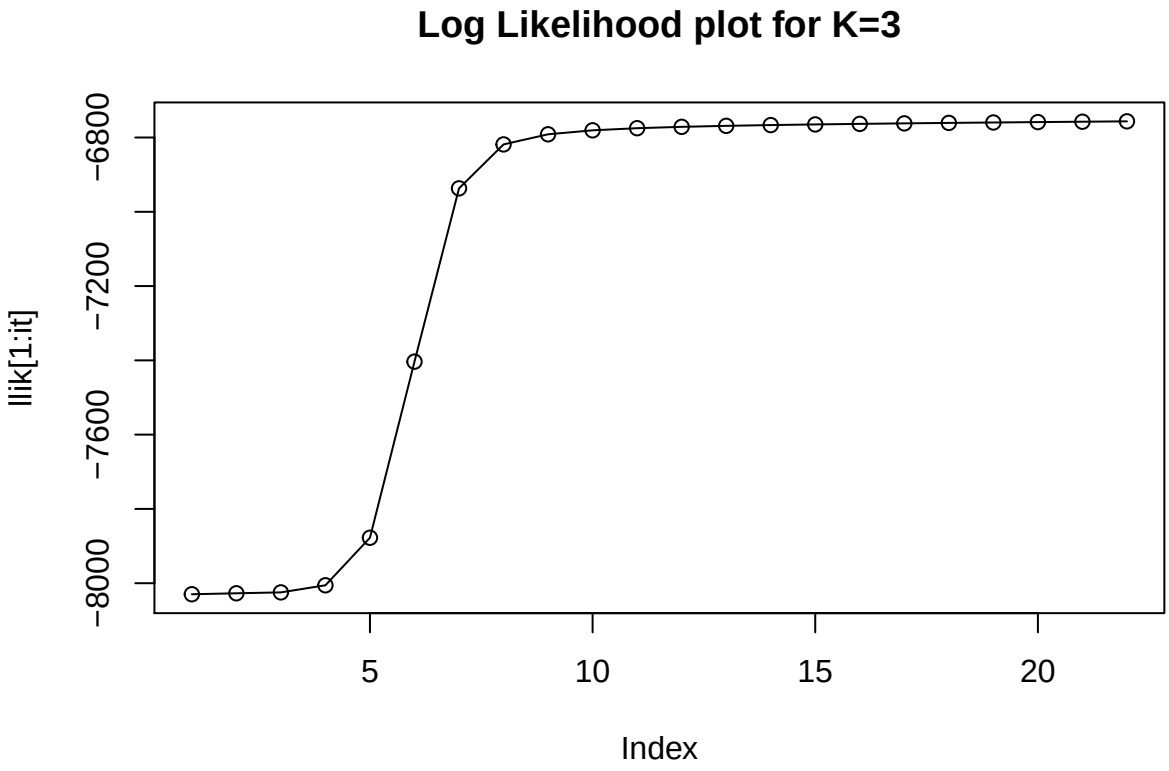

```
## Convergence at iteration:  10  with log likelihood:  -5643.615
```

For K=2 the EM algorithm's convergence is close to the true mu values. This is the reason we are getting very similar plots. The algorithm converged at iteration 10 with log likelihood value of "-5643.615". The minimum threshold value was 1.

3 K = 3


True Mu plot for K=3
true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

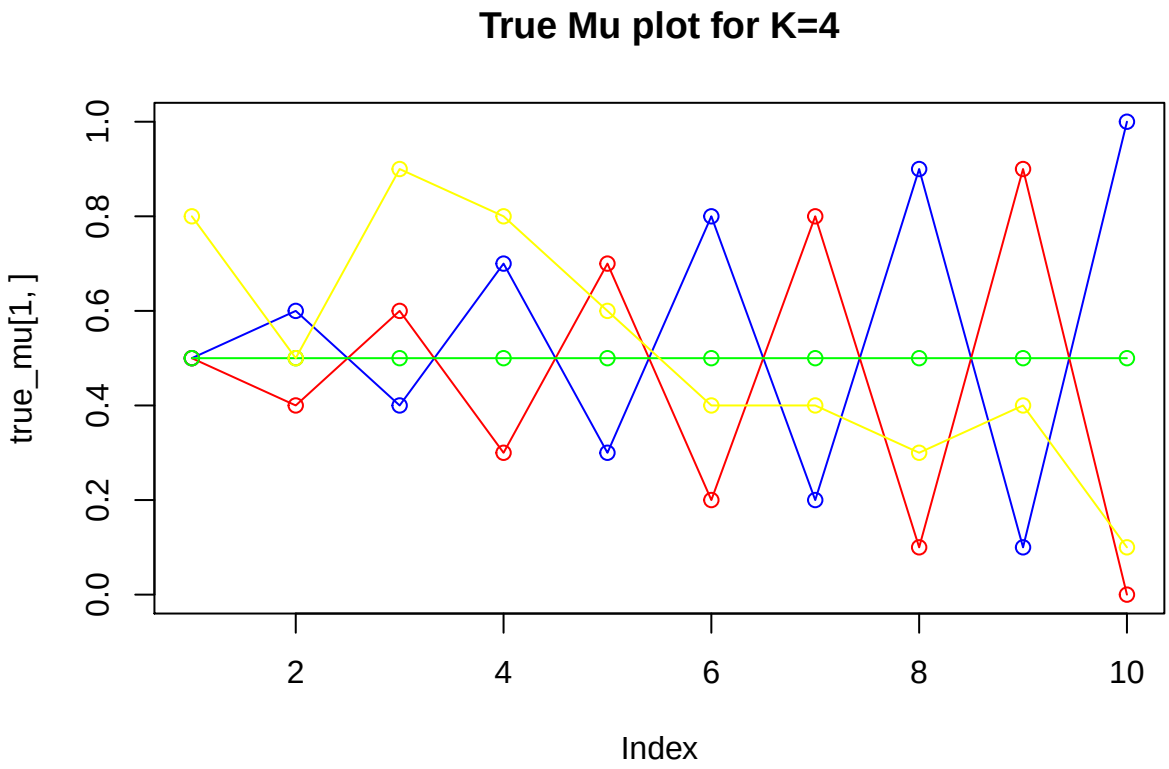

Converged Mu plot for K=3
mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

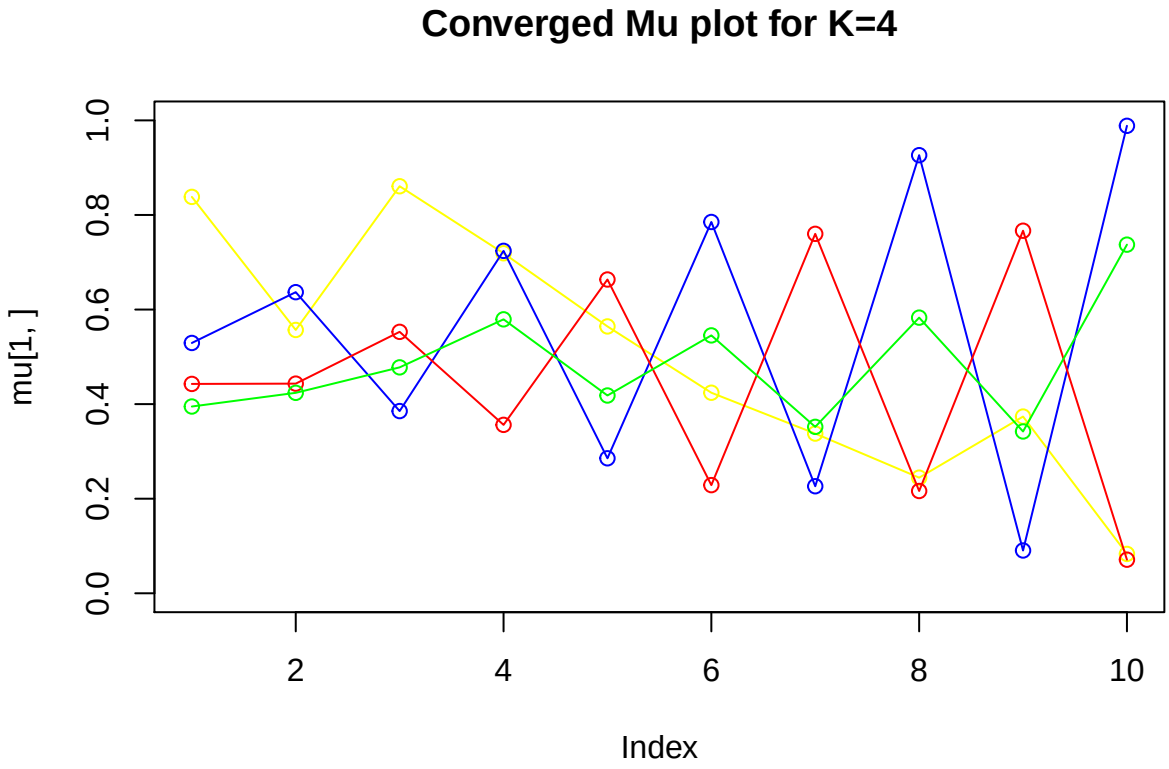


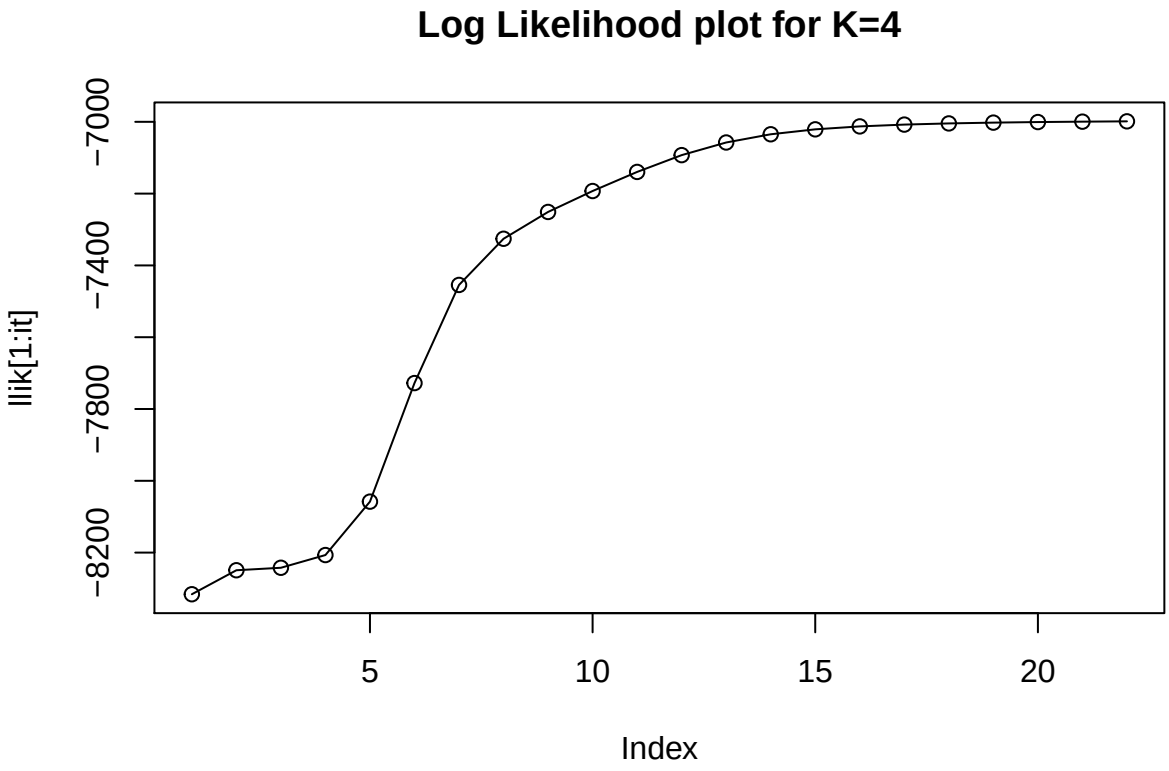

```
## Convergence at iteration:  22  with log likelihood:  -6756.521
```

For K to 3, the convergence is not exact. Since we had equal probabilities for all the rows of mu the convergence is switched between the rows. The algorithm converged at iteration 22 with log likelihood value of "-6756.521". The minimum threshold value was 1.

4 K = 4


True Mu plot for K=4
true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index


Converged Mu plot for K=4
mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index




```
## Convergence at iteration:  22  with log likelihood:  -6998.695
```

This is the convergence we get for K=4. Since the complexcity increases even more the convergence takes much more time, but it is a good approximation of the true model we had for mu. The algorithm converged at iteration 22 with log likelihood value of "-6998.695". The minimum threshold value was 1.

EM algorithm is famous for its conceptual simplicity and ease of implementation. Each iteration of the algorithm improves log likelihood. The rate of convergence is fast at first but slow down as you approach local optima. EM works best when the missing imformation(K) is small and the dimensionality of the data(D) is not too large. EM can require many iterations, and higher dimensionality can dramatically slow down the E-Step.

Appendix

```
knitr::opts_chunk$set(
    echo = FALSE,
    message = FALSE,
    warning = FALSE
)
library(mboost)
library(randomForest)
library(ggplot2)
spam_data <- read.csv2("spambase.csv")
spam_data$Spam <- as.factor(spam_data$Spam)
n=dim(spam_data)[1]
set.seed(12345)
```

```
ids=sample(1:n,floor(0.75*n))
spam_train=spam_data[ids,]
spam_test=spam_data[-ids,]
misclas_test <- c()
comp<-matrix(0,nrow=0,ncol=2)
colnames(comp) <- c("Classes","Error_ada")
for(i in seq(10,100,10)) {
    model_boost <-
        mboost::blackboost(
        Spam ~ .,
        data = spam_train,
        family = AdaExp(),
        control = boost_control(mstop = i)

        )
    predict_test<-predict(model_boost,newdata=spam_test,type="class")

    comp_test<-data.frame("Expected_Value"=spam_test$Spam,
                "Predicted_Value"=predict_test)
    pred<-table(comp_test)
    missclas_test<-(pred[2,1]+pred[1,2])/nrow(spam_test)
    comp <- rbind(comp,c(i,missclas_test))
}
comp<-as.data.frame(comp)

ggplot(comp,aes(Classes,Error_ada)) +geom_point() +
         geom_line(col='blue') + xlab("Number of Classifiers") +
         ylab("Errors")+ggtitle("Num of Modifiers vs Error" )
comp_ran<-matrix(0,nrow=0,ncol=2)
colnames(comp_ran) <- c("Classes","Error_rf")
for(i in seq(10,100,10)) {

    set.seed(1234)
    model_rforest<-randomForest(Spam~.,data=spam_train,ntree=i)
    predict_test<-predict(model_rforest,newdata=spam_test,type="class")

    comp_test<-data.frame("Expected_Value"=spam_test$Spam,
                "Predicted_Value"=predict_test)
    pred<-table(comp_test)
    missclas_test<-(pred[2,1]+pred[1,2])/nrow(spam_test)
    comp_ran <- rbind(comp_ran,c(i,missclas_test))
}

comp_ran<-as.data.frame(comp_ran)

ggplot(comp_ran,aes(Classes,Error_rf)) +geom_point() + geom_line(col="blue")+
  coord_cartesian(ylim=c(0.04,0.1)) + xlab("Number of Sample Trees") +
         ylab("Errors")+ggtitle("Num of Sample Trees vs Error" )
comp_data <- as.data.frame(merge(comp,comp_ran))
ggplot(comp_data,aes(x=Classes))+ geom_line(aes(y=Error_ada,color="blue")) +
          geom_line(aes(y=Error_rf,color="red")) +
          scale_color_discrete(name="Error",labels=c("AdaBoost","Random Forest"))+
          xlab("Number of Sample Trees/Classifiers") +ylab("Errors") +
```

```
            ggtitle("Random Forest vs Bagging" )
#Function for EM algorithm
em_alg = function(pi, mu, llik, min_change, maxit=100){
  old = 0
  for(it in 1:max_it) {
    # E-step: Computation of the fractional component assignments
    total <- matrix(0, nrow = nrow(x), ncol= 1)
    z = matrix(1, nrow = nrow(x), ncol = length(pi))
    for (i in 1:length(pi)) {
      for (j in 1:ncol(x)) {
        z[, i] = z[, i] * dbinom(x[ ,j], 1, mu[i,j])
      }
      z[, i] = z[, i] * pi[i]
      total= total + z[,i]
    }
    for(i in 1:length(pi)){
      z[,i] = z[,i]/total
    }
    # #Log likelihood computation.
    q = matrix(1, nrow = length(total), ncol = 1)
    for(i in 1:nrow(mu)){
      temp = matrix(1, nrow = nrow(x), ncol = 1)
      for(j in 1:ncol(x)){
        temp = temp*(mu[i,j]^x[,j])*((1-mu[i,j])^(1-x[,j]))
      }
      q = q * (temp*pi[i])^(z[,i])
    }
    llik[it] = sum(log(q))
    # Stop if the log likelihood has not changed significantly
    if(abs(old-llik[it])<min_change){
      break
    }
    else{
      old = llik[it]
    }
    #M-step: ML parameter estimation from the data and fractional component assignments
    new_pi = colSums(z)/nrow(x)
    new_mu = matrix(nrow = nrow(mu), ncol = ncol(mu))
    for (i in 1:length(pi)){
      nm = sum(z[,i])
      new_mu[i,] = colSums(x*z[,i])/nm
    }
    mu = new_mu
    pi = new_pi
  }
  return(list(pi, mu, llik, it))
}
##K=2
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 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 = 2) # true mixing coefficients
true_mu <- matrix(nrow=2, ncol=D) # true conditional distributions
true_pi=c(1/2,1/2)
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)
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1), main = 'True Mu plot for K=2')
points(true_mu[2,], type="o", col="red")
# Producing the training data
for(n in 1:N) {
  k <- sample(1:2,1,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)
}
#Call the EM-algorithm function
ret = em_alg(pi, mu, llik, min_change)
pi = ret[[1]]
mu = ret[[2]]
llik = ret[[3]]
it = ret[[4]]
plot(mu[1,], type="o", col="red", ylim=c(0,1), main = 'Converged Mu plot for K=2')
points(mu[2,], type="o", col='blue')
plot(llik[1:it], type="o", main = 'Log Likelihood plot for K=2')
cat("Convergence at iteration: ", it, " with log likelihood: ", llik[it], "\n")
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 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), main = 'True Mu plot for K=3')
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)
}
#Call the EM-algorithm function
ret = em_alg(pi, mu, llik, min_change)
pi = ret[[1]]
mu = ret[[2]]
llik = ret[[3]]
it = ret[[4]]
cols = c('green', 'blue')
plot(mu[1,], type="o", col="red", ylim=c(0,1), main = 'Converged Mu plot for K=3')
for(i in 2:nrow(mu)){
  points(mu[i,], type="o", col=cols[i-1])
}
plot(llik[1:it], type="o", main = 'Log Likelihood plot for K=3')
cat("Convergence at iteration: ", it, " with log likelihood: ", llik[it], "\n")
##K=4
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 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 = 4) # true mixing coefficients
true_mu <- matrix(nrow=4, ncol=D) # true conditional distributions
true_pi=c(1/4,1/4,1/4,1/4)
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)
true_mu[4,]=c(0.8,0.5,0.9,0.8,0.6,0.4,0.4,0.3,0.4,0.1)
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1), main = 'True Mu plot for K=4')
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
points(true_mu[4,], type="o", col="yellow")
# Producing the training data
for(n in 1:N) {
  k <- sample(1:4,1,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)
}
#Call the EM-algorithm function
ret = em_alg(pi, mu, llik, min_change)
pi = ret[[1]]
mu = ret[[2]]
llik = ret[[3]]
it = ret[[4]]
cols = c('blue', 'red', 'green')
plot(mu[1,], type="o", col="yellow", ylim=c(0,1), main = 'Converged Mu plot for K=4')
for(i in 2:nrow(mu)){
  points(mu[i,], type="o", col=cols[i-1])
}

plot(llik[1:it], type="o", main = 'Log Likelihood plot for K=4')
cat("Convergence at iteration: ", it, " with log likelihood: ", llik[it], "\n")
```