Group_A11:Block-2,Lab1

Sridhar, Naveen, Obaid

24 November 2018

Contents

Assignment 1 **2**

1 Ensemble Method **2**
1 Adaboost 2
2 Random Forest 3

Assignment 2 **5**
1 EM Function for Bernoulli Distribution 5
2 K = 2 6
3 K = 3 8
4 K = 4 10

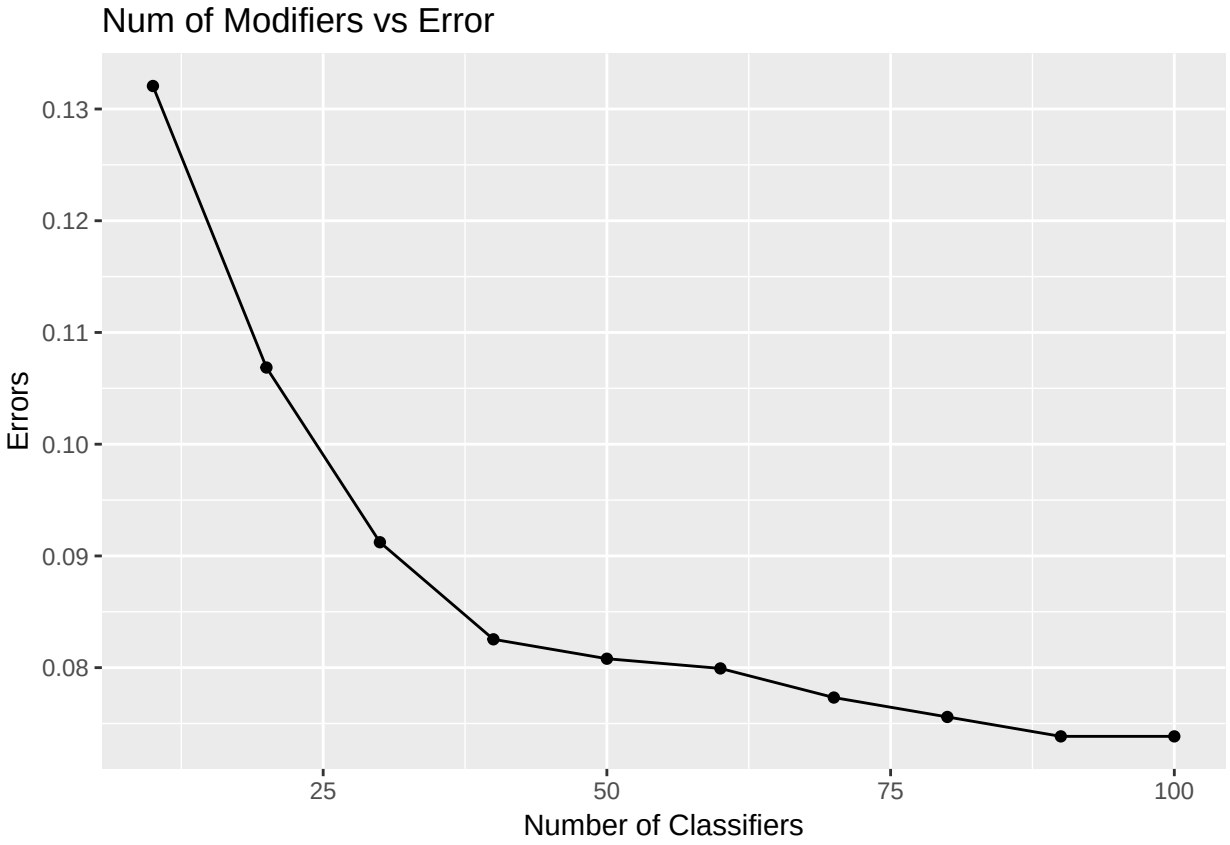
Appendix **11**

Assignment 1

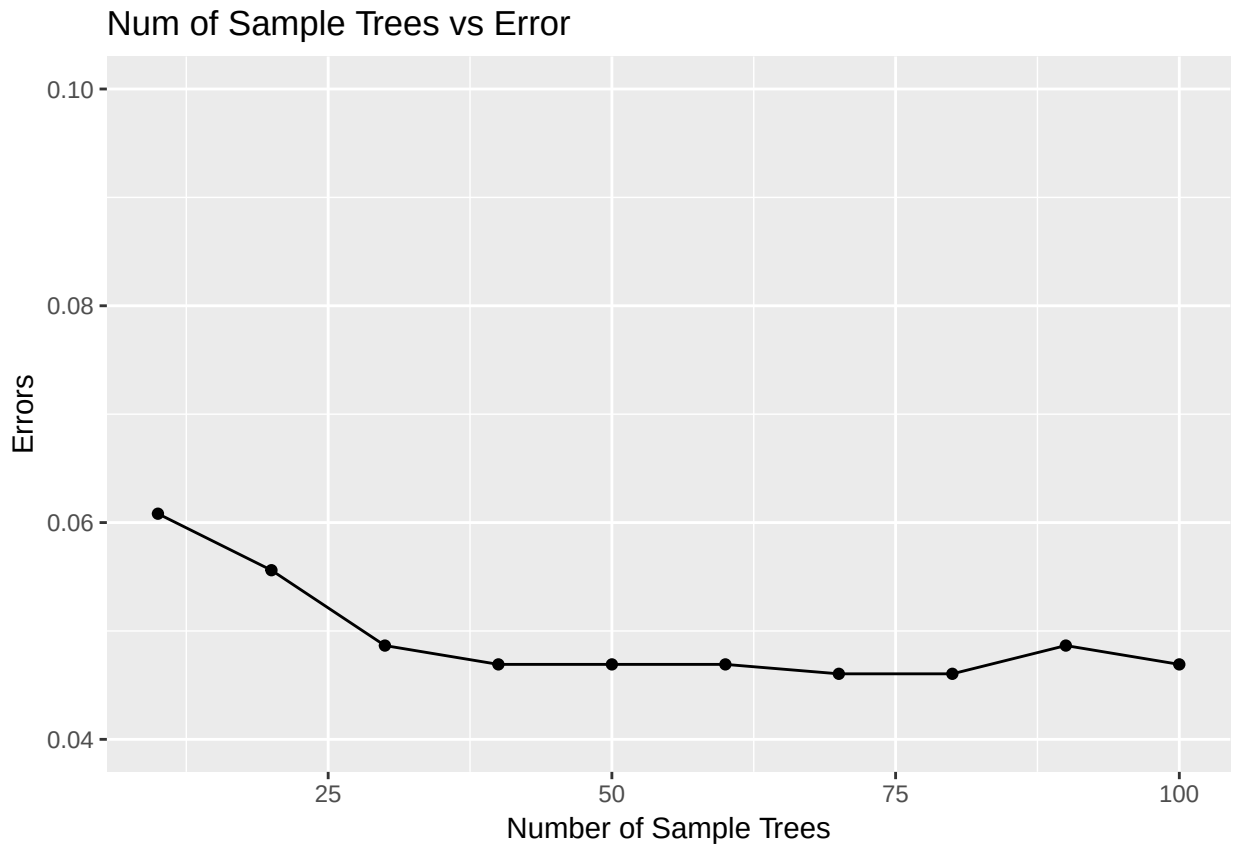
1 Ensemble Method

1 Adaboost

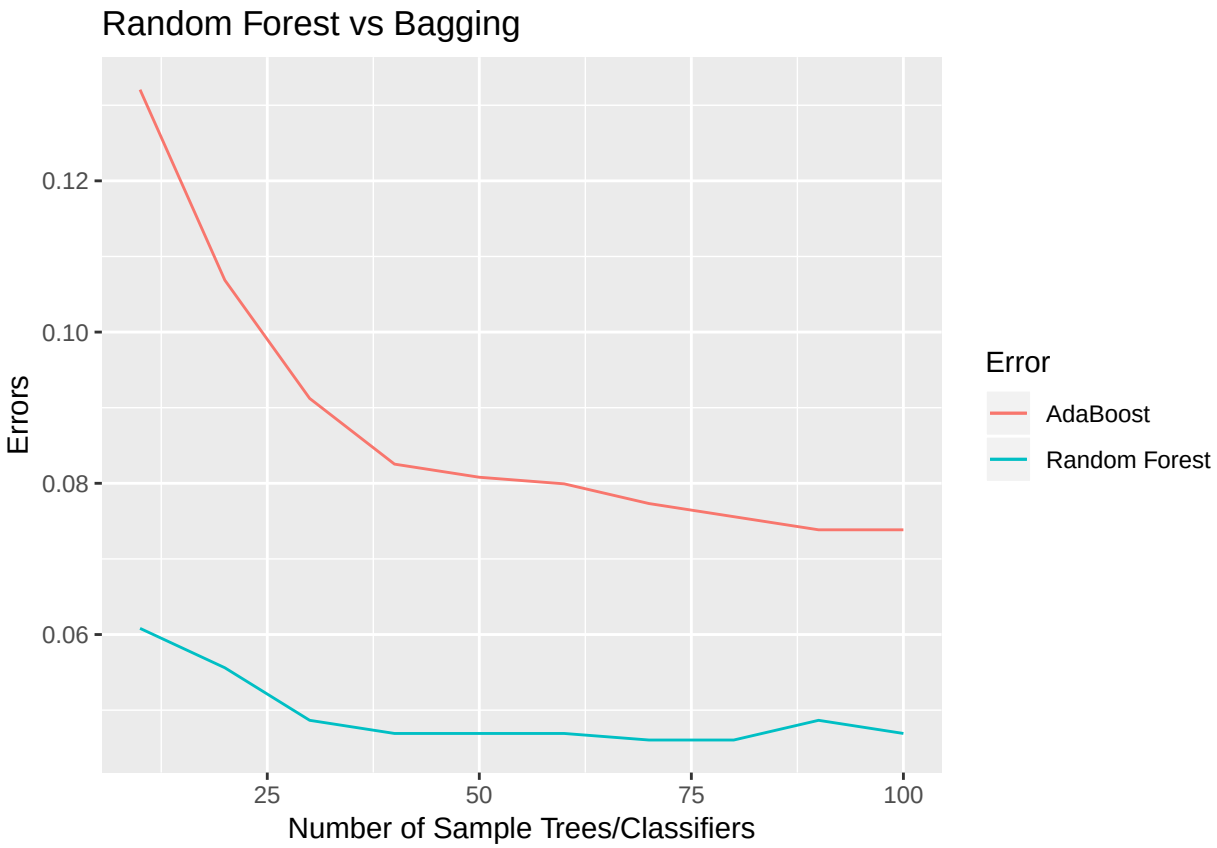
From the graph it can be seen that as the number of iteration increases, the adaboost model improves in accuracy. This is because, at each iteration the weights to those parameter is given more importance which predicts poorly. The updated weight is reflected in next iteration. So as the number of iterations increases the cumulative classifier predictiveness improves. Here the number of iteration consider for the classifier is 10,20..100 and the classfier is decision tree.



2 Random Forest



From the graph it can be seen that as the number of bagging increases, the error doesn't change much on a grand scale. Although, through midway around 50-75 sample trees the error reaches the lowest point and from there the error rate gradualy climbs up.



The plot shows the comparison between Ada Boost vs Random Forest for the same number of classifier/sampling. In this case Random Forest seems to perform better than Ada Boost.

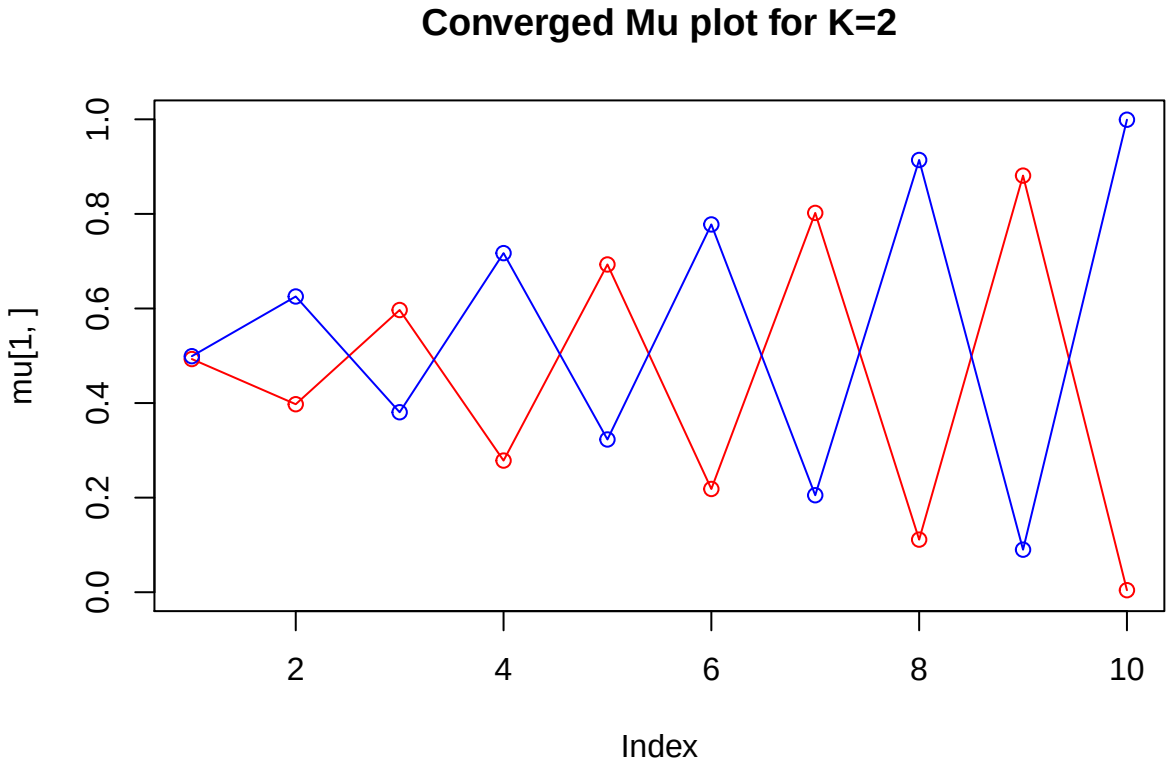
Assignment 2

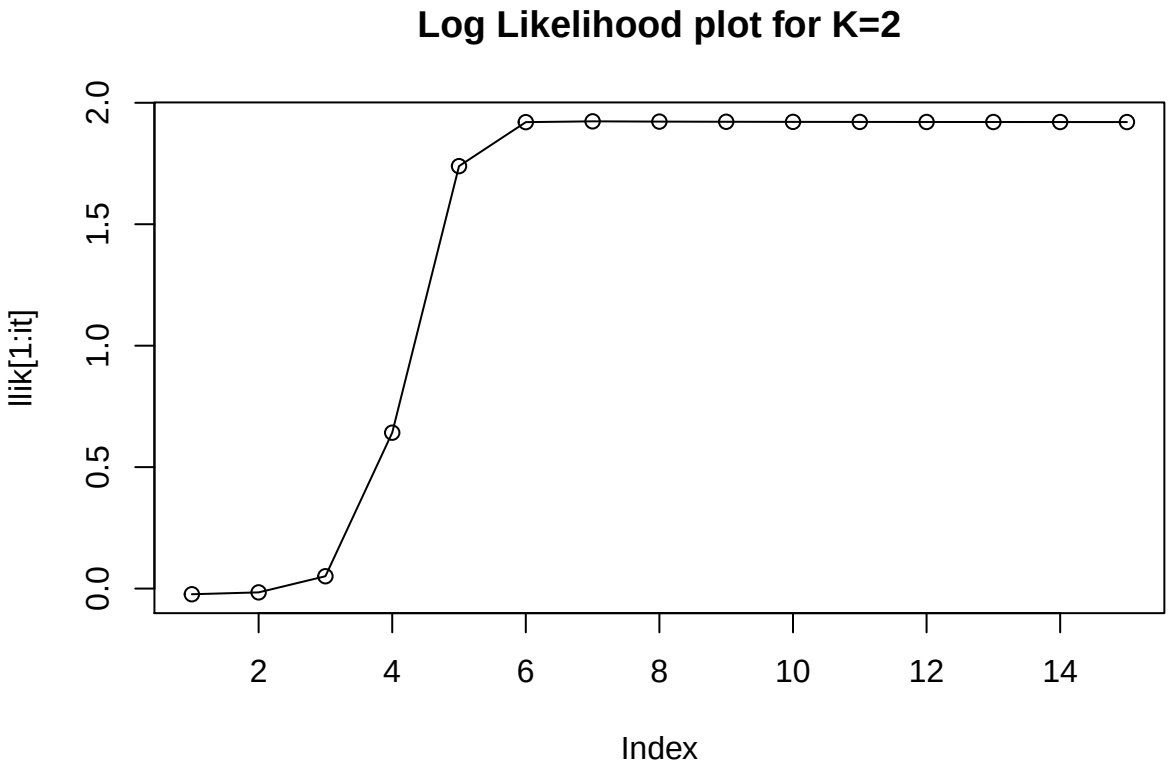
1 EM Function for Bernoulli Distribution

```
#Function for EM algorithm
em_alg = function(pi, mu, llik, min_change, maxit=100){
  old = 0
  cols = c('green', 'blue', 'yellow')
  for(it in 1:max_it) {
    # E-step: Computation of the fractional component assignments
    # Your code here
    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.
    llik[it] = log(sum(total))

    #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    flush.console()
    # Stop if the log likelihood has not changed significantly
    # Your code here
    if(abs(old-llik[it])<min_change){
      break
    }
    else{
      old = llik[it]
    }

    #M-step: ML parameter estimation from the data and fractional component assignments
    # Your code here
    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

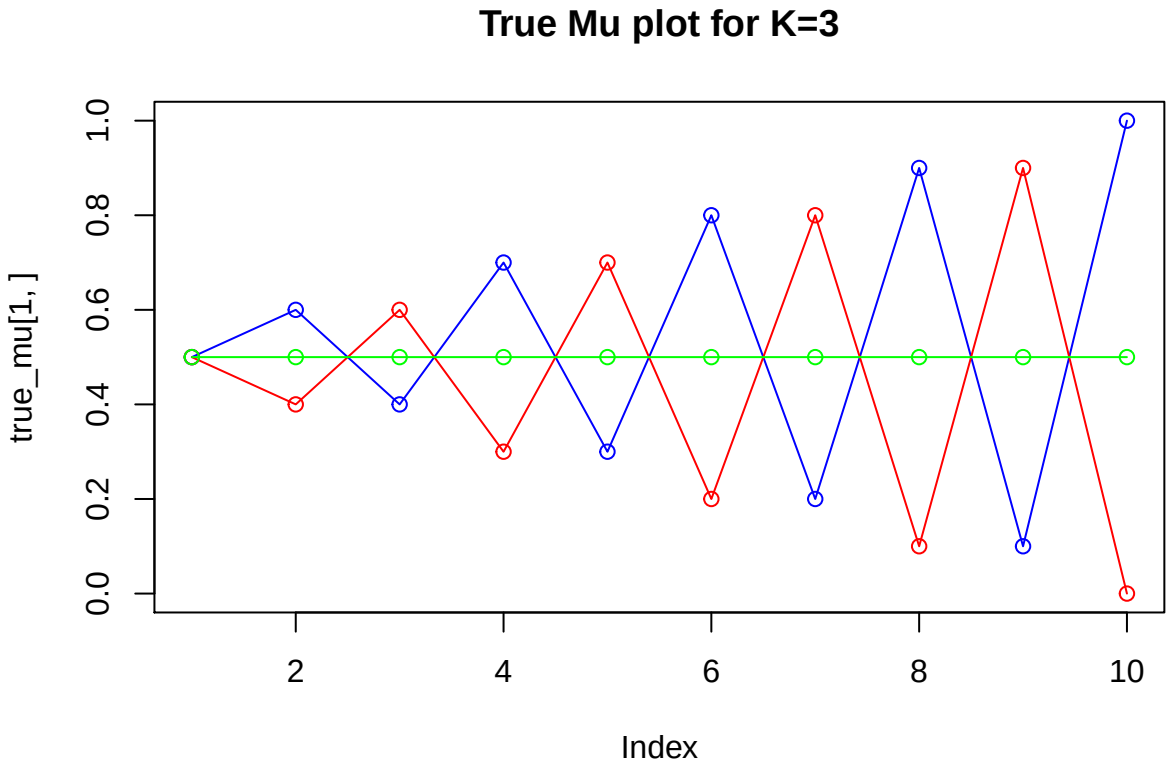

True Mu plot for K=2
true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

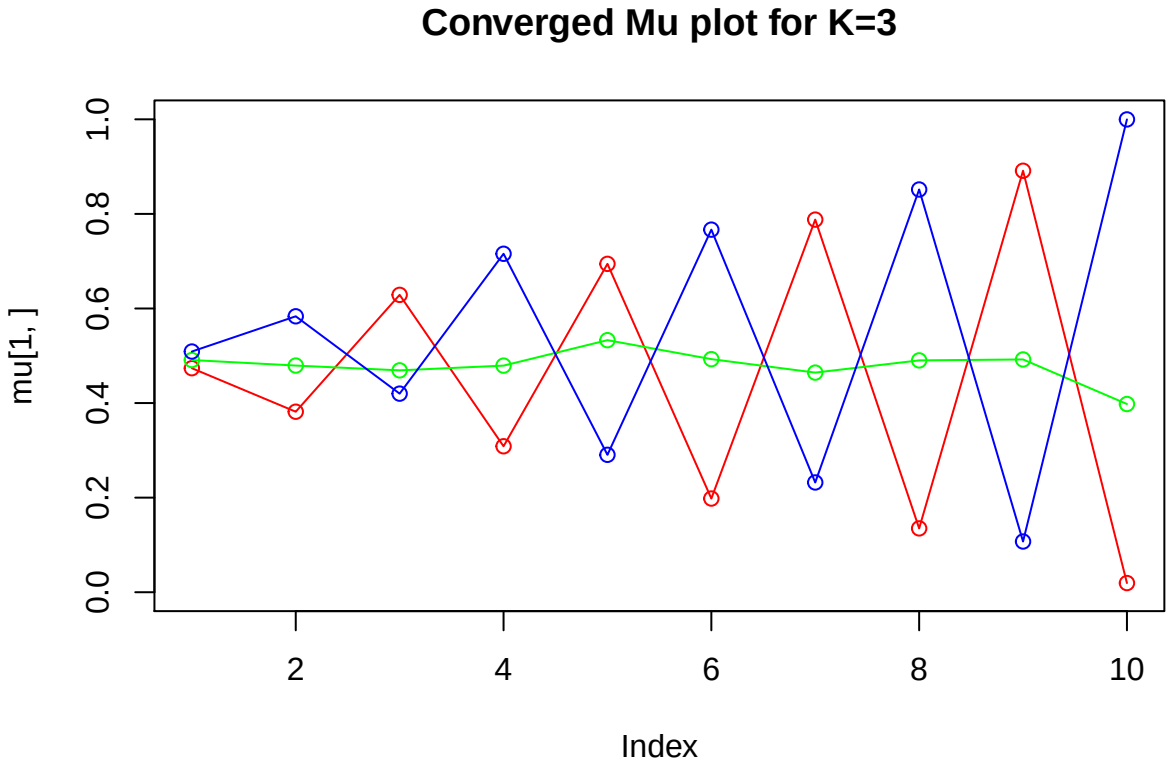

Converged Mu plot for K=2
mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

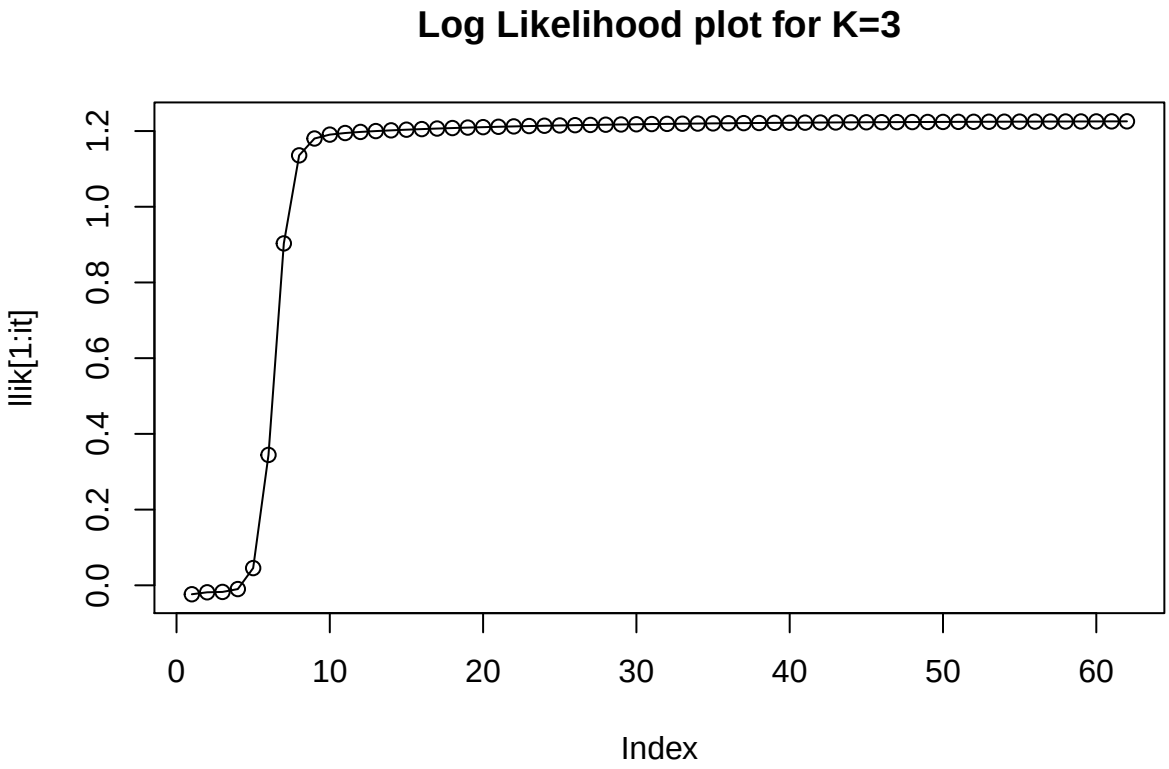


For K=2 the EM algorithm's convergence is close to the true mu values. This is the reason we are getting very similar plots. The EM algorithm is converged near the iteration value 6. There are very small changes in the log likelihood after that but the threshold value set is quiet low (0.0001), this is the reason it goes upto 15th iteration.

3 K = 3

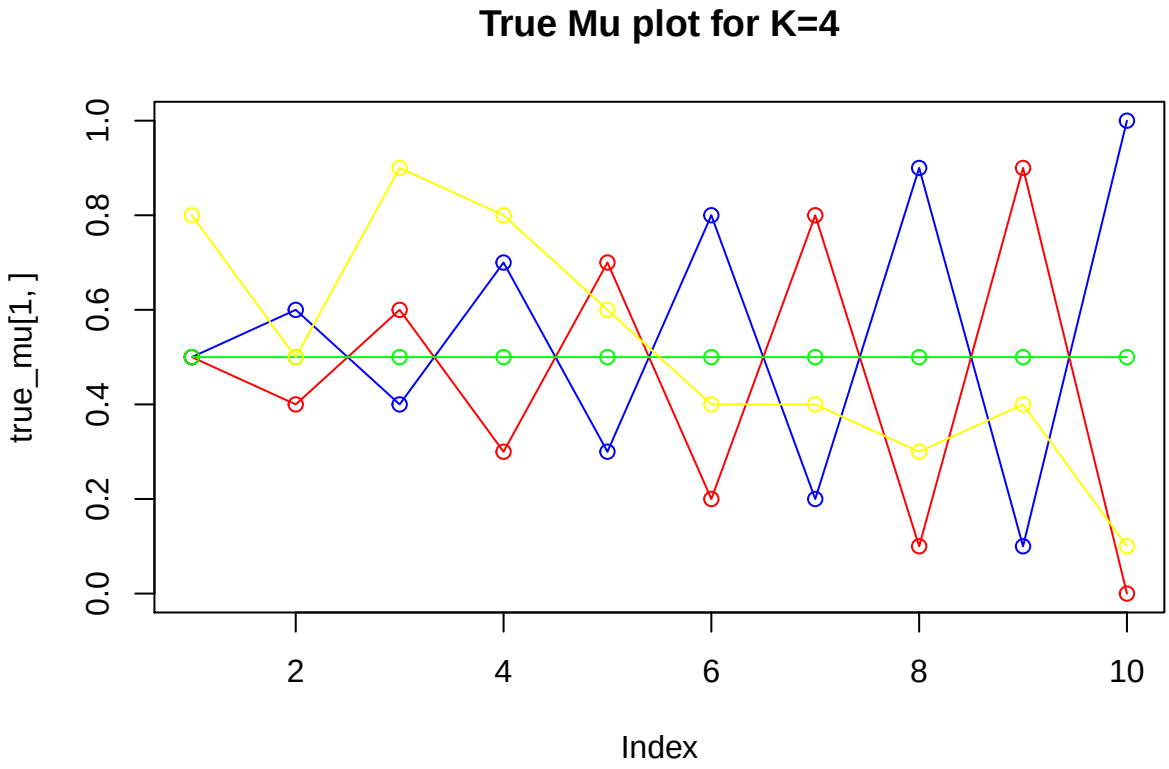

True Mu plot for K=3
true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

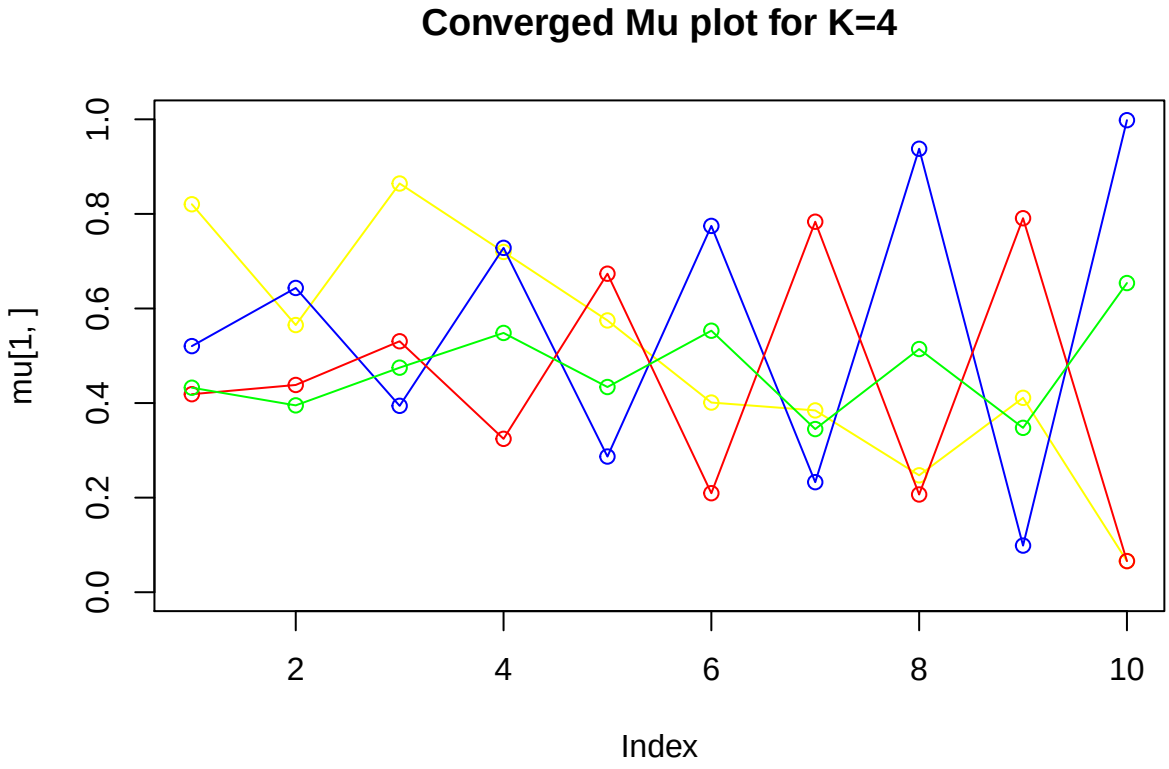

Converged Mu plot for K=3
mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

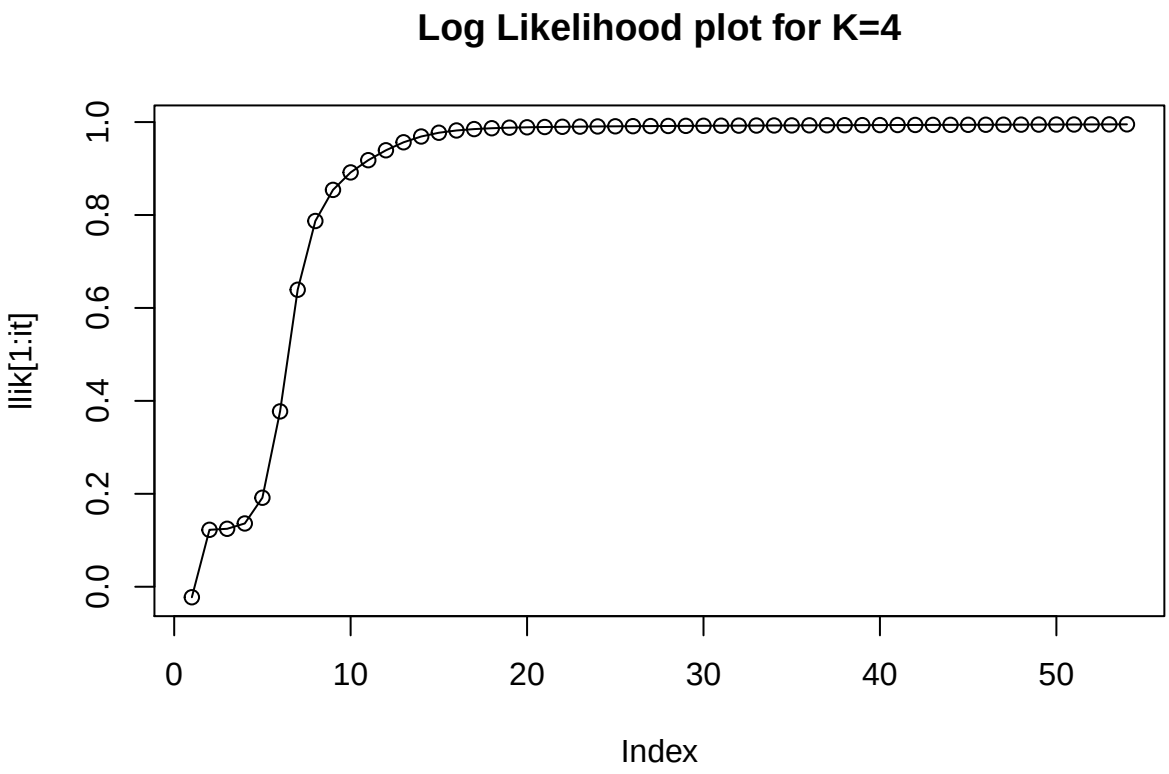


As we increase the K to 3, the convergence is not exact. Since we had equal probabilities for all the rows of mu the convergence is switched between the rows. I think that the reason behind values not converging completely is we increase the value of K and the dimensions remain unchanged. If we also increase the dimensions may be the model could converge better for higher values of K.The EM algorithm is converged near the iteration value 11. There are very small changes in the log likelihood after that but the threshold value set is quiet low (0.0001), this is the reason it goes upto 60th iteration.

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



This is the convergence we get for K=4. Since the complexcity increases even more and the dimensions remain unchanged the convergence is not too good. It is a good approximation of the true model we had for mu. The EM algorithm is converged near the iteration value 20. There are very small changes in the log likelihood after that but the threshold value set is quiet low (0.0001), this is the reason it goes upto 55th iteration.

Given we just had the X matrix and the EM algorithm converges the randomly initialized pi and mu to such an extent is good. It gives us a good approximation of the distribution from which X was generated. We can see from the 3 plots that as the value of K increases the EM algorithm takes more iterations to converge.

-For K=2 it was around 6,

-For K=3 it was around 11,

-For K=4 it was around 20

Appendix

```
knitr::opts_chunk$set(
    echo = FALSE,
    message = FALSE,
    warning = FALSE
)
library(mboost)
library(randomForest)
library(ggplot2)
```

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

n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n,floor(0.75*n))
spam_train=spambase[id,]
spam_test=spambase[-id,]

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)

plot1 <- ggplot(comp,aes(Classes,Error_ada)) +geom_point() +
         geom_line() + xlab("Number of Classifiers") +
         ylab("Errors")+ggtitle("Num of Modifiers vs Error" )

plot1
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()+
  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
  cols = c('green', 'blue', 'yellow')
  for(it in 1:max_it) {
    # E-step: Computation of the fractional component assignments
    # Your code here
    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.
    llik[it] = log(sum(total))

    #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    flush.console()
    # Stop if the log likelihood has not changed significantly
    # Your code here
    if(abs(old-llik[it])<min_change){
      break
    }
    else{
      old = llik[it]
    }

    #M-step: ML parameter estimation from the data and fractional component assignments
    # Your code here
    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 <- 0.01 # 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, 0.0001)
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')
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
```

```
min_change <- 0.001 # 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, 0.0001)
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')
##K=4
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.001 # 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, 0.0001)
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')
```