Lab1_block2

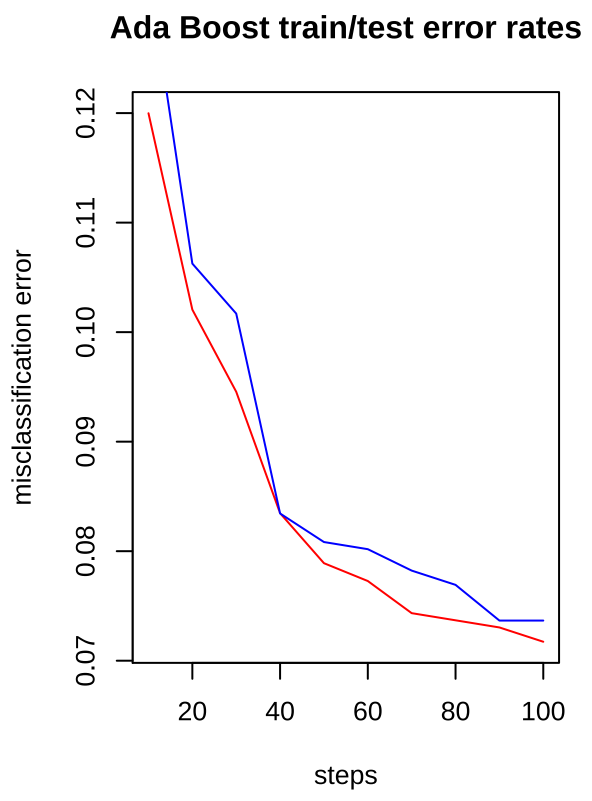
Andreas C Charitos-andch552

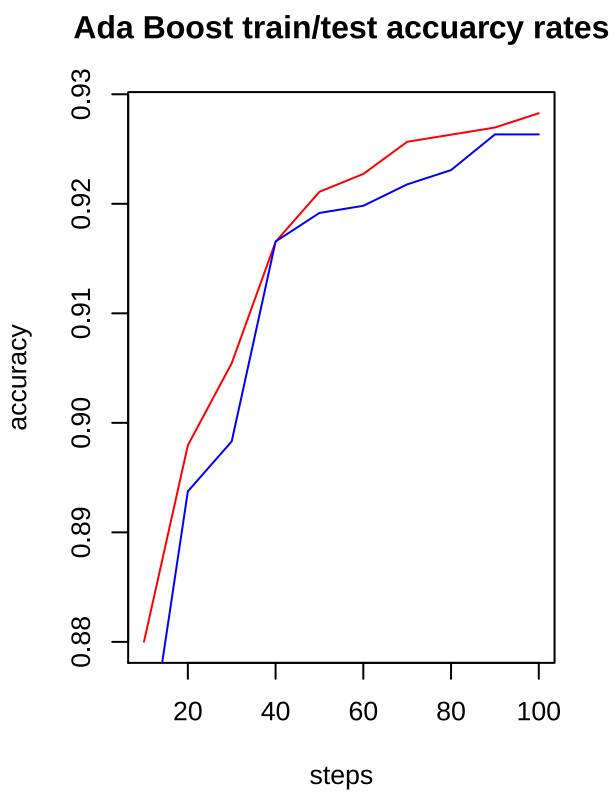
1 Dec 2018

Contents

Ensemble Methods

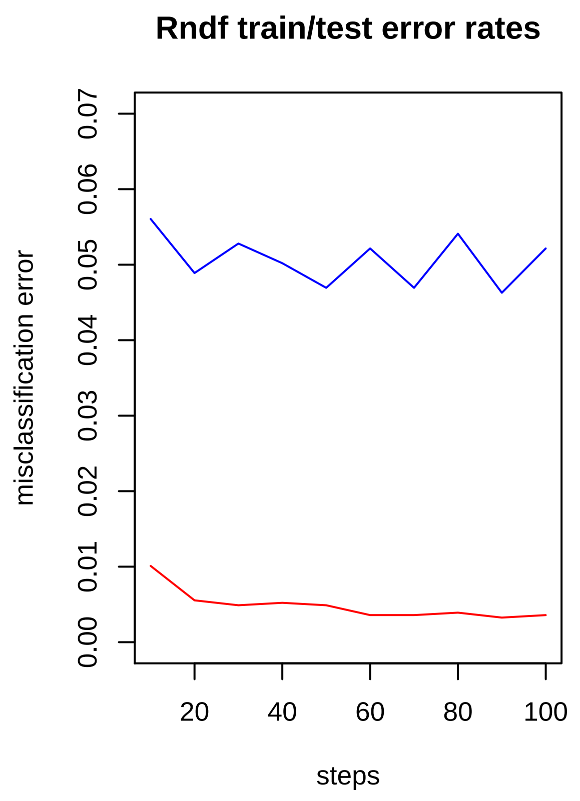
Ada Boost classification Trees

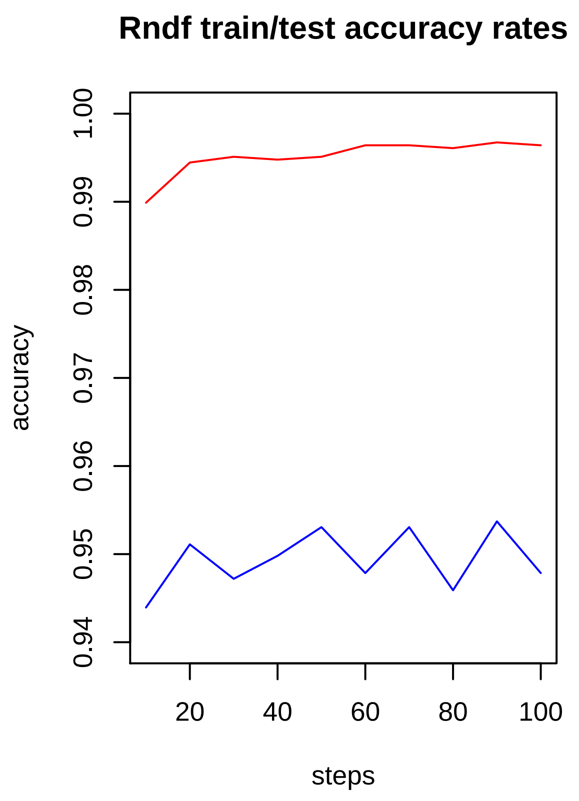




The above plots provide information for about the misclassification rates and accuracy obtained using ada boost algorithm on train and test data. The red lines respresents the misclassification error/accuracy for train data using a range of 10 to 100 trees and the blue lines is the equivalent for test data.From the plots we can see that using Ada boost the misclassification rates for both training and test data are decreasing as the number of trees increases and at 40 trees both train and test errors are equal.On the other side the accuracy is increasing as the number of trees increases.

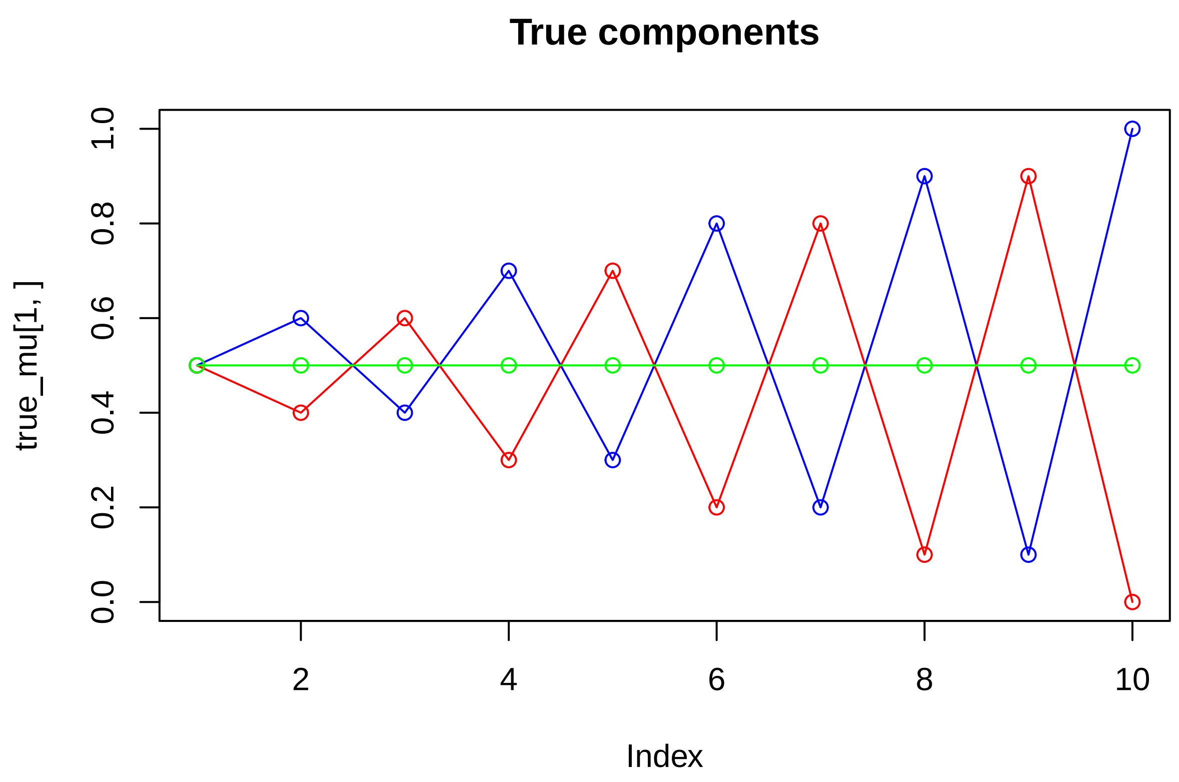
Random Forest Trees

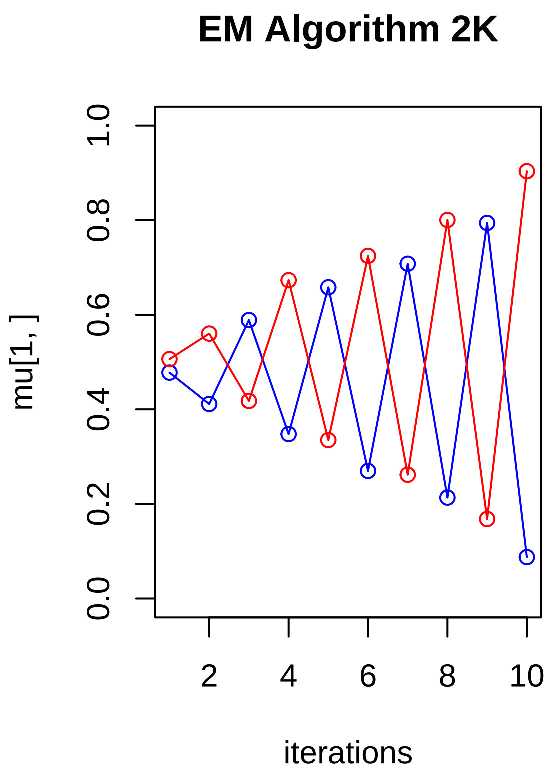


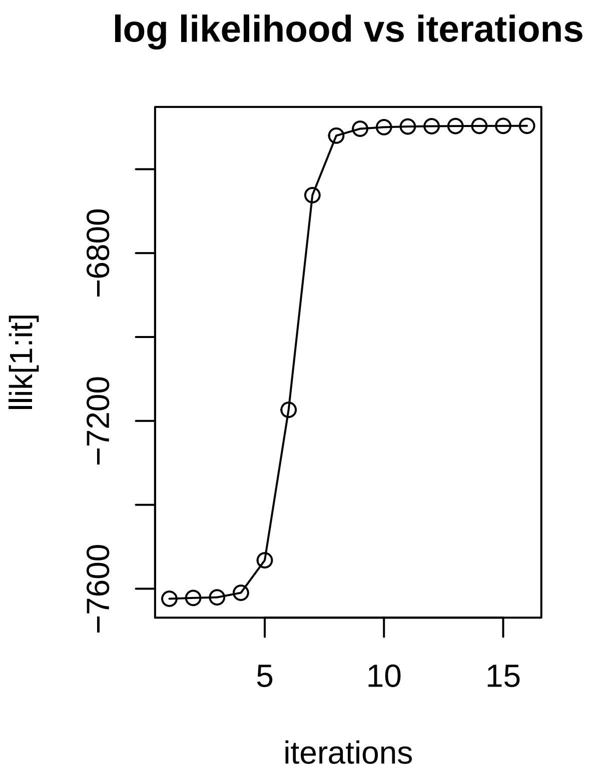


The above plots provide information for about the misclassification rates and accuracy obtained using random forests algorithm on train and test data. The red lines respresents the misclassification error/accuracy for train data using a range of 10 to 100 trees and the blue lines is the equivalent for test data.From the plots we can see that using Random forest the misclassification rates for training is droping until 20 number of trees and then is slightly decreases as number of trees increases.As for test data the misclassification error is flactuating as the number of trees increases and only decreases slightly at the end.On the other side the accuracy for train data is increasing until 20 trees then just increases very little and the accuracy for test data is flactuating as the number of trees increases only to improve a little at the end.

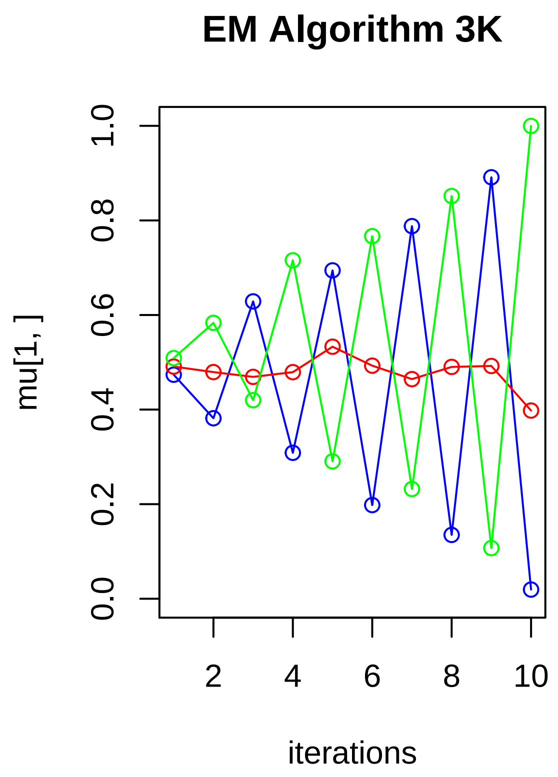
Finaly,comparing the two methods we can canclude that random forest method obtains a very accurate model with 20 trees and as the number of trees grows the model improves only a little reagrading accuracy and misclassification error.On the other hand, the ada boost model keeps improving as the number of trees grows this is because every time the algotrithm combines predictors by adaptively put more weight to the difficult to classify observations and works iteratetively.

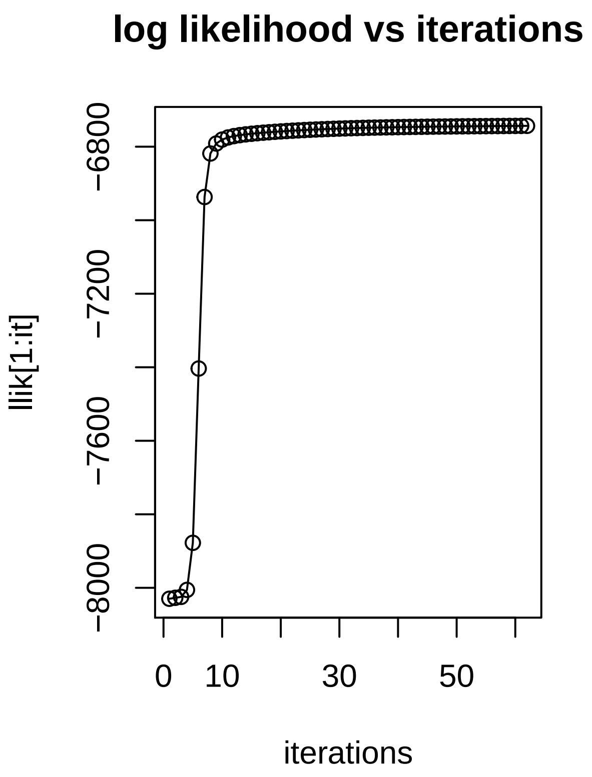

True components
true_mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



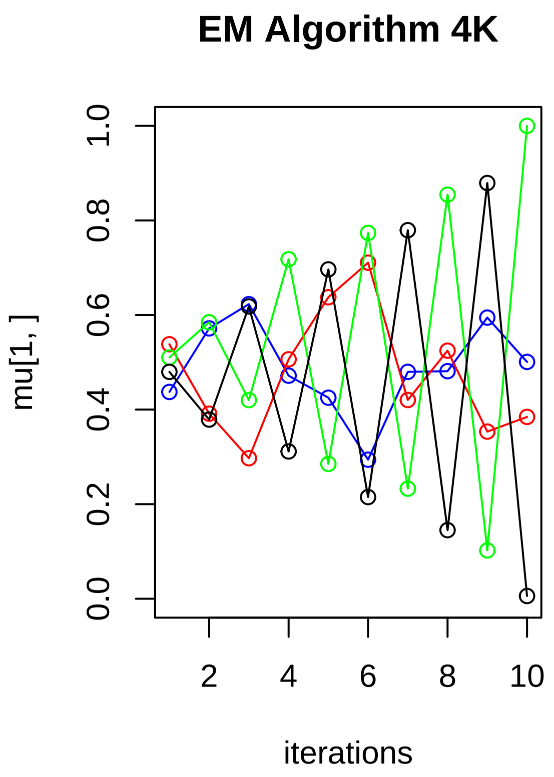


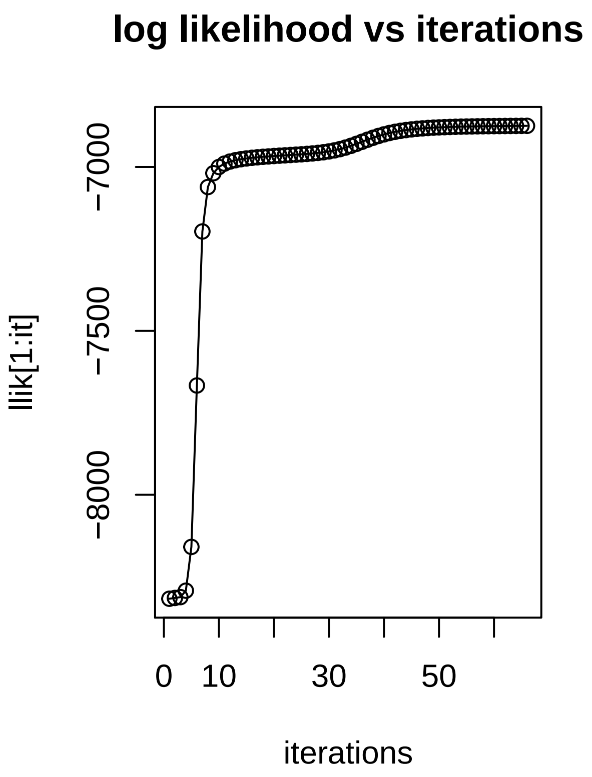
In the above plot we can see the estimated plot why K=2 and the log likelihood vs the number of iterations.We can see the 2 paths provide a good approximation for 2 of the paths of the original plot.We can see that from the pi's that are (0.4981919 0.5018081) for the 2 compoments.Also rearding the log likelihood value we see that drops heavily at iterations 5 to 9 and after the decrease is very small until 15 iterartions where the algorithm converges.





Next we add one more component and K=3 and try to estimate the 3 original plot.We can see that 2 of the lines(grren,blue) seems to fit almost identical to the original and the third one(red) approximates it very well. The pi's in that case are (0.3259592 0.3044579 0.3695828).The log lkelihood again drops heavily from iteration 5 to 9 and then slightly decreases atevery iteration until in converges at 64 iterations.





In the final step we add one more component and K=4 and again try to find 3 compoments that provide a good apporoximation of the original plot. We see that only 2 components (green and black) seems to provide a good approximation for the 2 original components..The red and blue lines in our estimated plot didn't manage to correctly map the third component.Reporting the pi's in that case (0.1614155 0.1383613 0.3609912 0.3392319). The loglikelihood follows approximately the same trend as the one with 3 components.

In conclusion,we see that 2 components play a significant part in the explaination of the data because as we have seen when number of components increses 2 components are allways present as good approximations using EM Algorithm.

Apdendix

```r
#spam=1 , no spam=0
#load libraries
library(mboost)
library(randomForest)
#load data
sp <- read.csv2("spambase.csv")
sp$Spam <- as.factor(sp$Spam)
#split data to 2/3 train , 1/3 test
n=dim(sp)[1]
set.seed(12345)
id=sample(1:n, floor(n*2/3))
train_spam=sp[id,]
test_spam=sp[-id,]

#misclassification error function
mis_error<-function(X,X1){
  n<-length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

#adaboost
steps<-seq(10,100,10)
er_tr_ada<-double(10)
er_tes_ada<-double(10)
#calculate mis.error for train and test
for (i in steps){
  ada<-blackboost(Spam~.,data=train_spam,family = AdaExp(),
                  control=boost_control(mstop =i))#mstop
  preds_tr_ada<-predict(ada,train_spam,type="class")
  preds_tes_ada<-predict(ada,test_spam,type="class")
  er_tr_ada[i/10]<-mis_error(train_spam$Spam,preds_tr_ada)
  er_tes_ada[i/10]<-mis_error(test_spam$Spam,preds_tes_ada)
  ada_mat<-cbind(er_tr_ada,er_tes_ada)
}

#data frame containing errors
ada_df<-as.data.frame(ada_mat)

# you can use par(mfrow=c(1,2)) and shrink the title size with cex.main=0.9
layout(matrix(c(1,2,1,2), 2, 2, byrow = TRUE))
#plots of mis.errors for train and test
plot(steps,ada_df[,1],type="l",col="red",main="Ada Boost train/test error rates",
     ylab="misclassification error")
lines(steps,ada_df[,2],type="l",col="blue")
#plots of accuracy for train and test
plot(steps,(1-ada_df[,1]),type="l",col="red",main="Ada Boost train/test accuarcy rates",
     ylab = "accuracy")
lines(steps,(1-ada_df[,2]),type="l",col="blue")


#random forests
set.seed(12345)
```

```r
er_tr_rnd<-double(10)
er_tes_rnd<-double(10)
#calculate mis.error for train and test
for (i in steps){
  rndforest<-randomForest(Spam~.,data=train_spam,ntree=i)
  preds_tr_rnd<-predict(rndforest,train_spam,type="class")
  preds_tes_rnd<-predict(rndforest,test_spam,type="class")
  er_tr_rnd[i/10]<-mis_error(train_spam$Spam,preds_tr_rnd)
  er_tes_rnd[i/10]<-mis_error(test_spam$Spam,preds_tes_rnd)
  rand_mat<-cbind(er_tr_rnd,er_tes_rnd)
}

#data frame containing errors
rnd_df<-as.data.frame(rand_mat)
# you can use par(mfrow=c(1,2)) and shrink the title size with cex.main=0.9
layout(matrix(c(1,2,1,2), 2, 2, byrow = TRUE))
#plots of mis.errors for train and test
plot(steps,rnd_df[,1],type="l",col="red",ylim = c(0,0.07),main="Rndf train/test error rates",
     ylab="misclassification error")
lines(steps,rnd_df[,2],type="l",col="blue")
#plots of accuracy for train and test
plot(steps,(1-rnd_df[,1]),type="l",col="red",ylim=c(0.94,1),main="Rndf train/test accuracy rates",
     ylab="accuracy")
lines(steps,(1-rnd_df[,2]),type="l",col="blue")

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),main="True components")
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
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
# 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])
  }
}
```

```r
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)
}

for(it in 1:max_it) {
  #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
  # Your code here
  m=matrix(nrow=1000,ncol=K)
  for (i in 1:1000){
    for (j in 1:K){
      m[i,j]<-prod((mu[j,]^x[i,])*(1-mu[j,])^(1-x[i,]))

      #unlist(lapply(mu[j,],function(y){ifelse(x[i,which(y %in% mu[j,])]==1,1-y,y)}))
    }
  }
  m<-t(t(m)*pi)
  z<-m/rowSums(m)

  l <- 0
  for (n in 1:1000) {
    for (k in 1:K) {
      sum <- 0
      for (D in 1:10) {
        sum =sum+ (x[n, D] * log(mu[k, D]) + (1 - x[n, D]) * log(1 - mu[k, D]))
      }

      l = l+ (z[n,k] * (log(pi[k]) + sum))
    }
  }

  llik[it]<-l
  #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  #flush.console()
  # Stop if the lok likelihood has not changed significantly
  # Your code here
  if(abs(llik[it]-(llik[it-1]))<=min_change&&(it>1)){

    break
  }
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  pi<-colSums(z)/N
  mu<-t(z)%*%x
  mu<-mu/colSums(z)
}
par(mfrow=c(1,2))
plot(mu[1,], type="o", col="blue", ylim=c(0,1),main="EM Algorithm 2K",xlab = "iterations")
points(mu[2,], type="o", col="red")
plot(llik[1:it], type="o",main="log likelihood vs iterations",xlab="iterations")

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)
# 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)
}


for(it in 1:max_it) {
  #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
  # Your code here
  m=matrix(nrow=1000,ncol=K)
  for (i in 1:1000){
    for (j in 1:K){
      m[i,j]<-prod((mu[j,]^x[i,])*(1-mu[j,])^(1-x[i,]))

      #unlist(lapply(mu[j,],function(y){ifelse(x[i,which(y %in% mu[j,])]==1,1-y,y)}))
    }
  }
  m<-t(t(m)*pi)
  z<-m/rowSums(m)

  l <- 0
  for (n in 1:1000) {
    for (k in 1:K) {
      sum <- 0
      for (D in 1:10) {
        sum =sum+ (x[n, D] * log(mu[k, D]) + (1 - x[n, D]) * log(1 - mu[k, D]))
      }

      l = l+ (z[n,k] * (log(pi[k]) + sum))
    }
  }

  llik[it]<-l
  #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  #flush.console()
  # Stop if the lok likelihood has not changed significantly
  # Your code here
  if(abs(llik[it]-(llik[it-1]))<=min_change&&(it>1)){

    break
  }
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  pi<-colSums(z)/N
  mu<-t(z)%*%x
  mu<-mu/colSums(z)
}
par(mfrow=c(1,2))
plot(mu[1,], type="o", col="blue", ylim=c(0,1),main="EM Algorithm 3K",xlab = "iterations")
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
plot(llik[1:it], type="o",main="log likelihood vs iterations",xlab="iterations")

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
# 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)
}

for(it in 1:max_it) {
  #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
  # Your code here
  m=matrix(nrow=1000,ncol=K)
  for (i in 1:1000){
    for (j in 1:K){
      m[i,j]<-prod((mu[j,]^x[i,])*(1-mu[j,])^(1-x[i,]))

      #unlist(lapply(mu[j,],function(y){ifelse(x[i,which(y %in% mu[j,])]==1,1-y,y)}))
    }
  }
  m<-t(t(m)*pi)
  z<-m/rowSums(m)

  l <- 0
  for (n in 1:1000) {
    for (k in 1:K) {
      sum <- 0
      for (D in 1:10) {
        sum =sum+ (x[n, D] * log(mu[k, D]) + (1 - x[n, D]) * log(1 - mu[k, D]))
      }

      l = l+ (z[n,k] * (log(pi[k]) + sum))
    }
  }

  llik[it]<-l
  #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  #flush.console()
  # Stop if the lok likelihood has not changed significantly
  # Your code here
  if(abs(llik[it]-(llik[it-1]))<=min_change&&(it>1)){

    break
  }
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  pi<-colSums(z)/N
  mu<-t(z)%*%x
  mu<-mu/colSums(z)
}
```

```r
par(mfrow=c(1,2))
plot(mu[1,], type="o", col="blue", ylim=c(0,1),main="EM Algorithm 4K",xlab = "iterations")
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
points(mu[4,], type="o", col="black")
plot(llik[1:it], type="o",main="log likelihood vs iterations",xlab="iterations")
```