Lab1 Block2

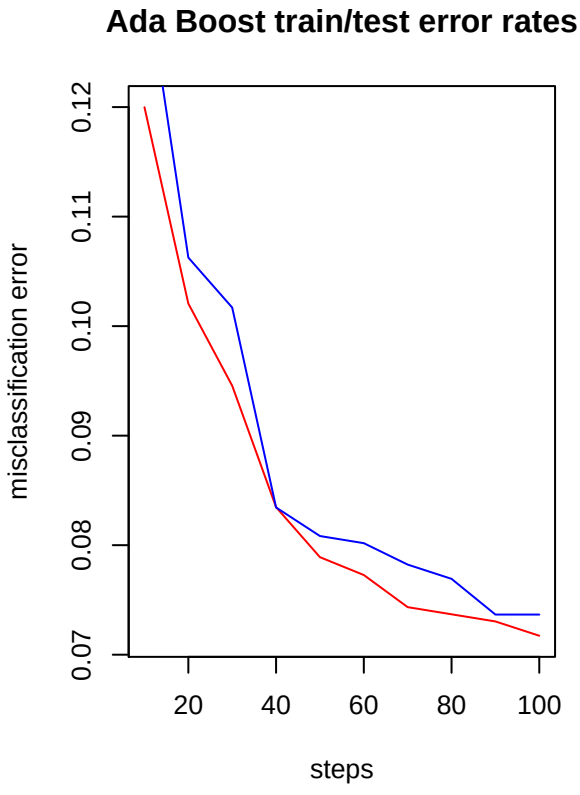
Group A5

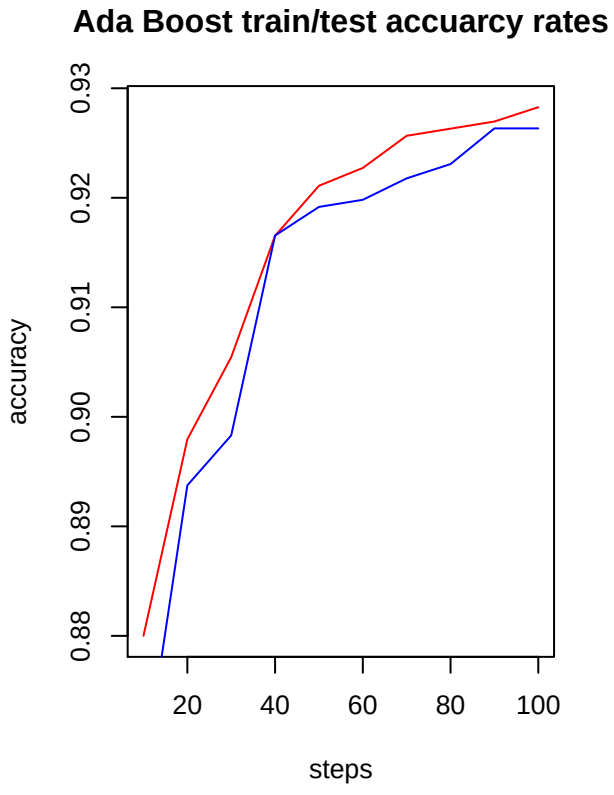
1 Dec 2018

Contents

Assignment 1

Ada Boost classification Trees

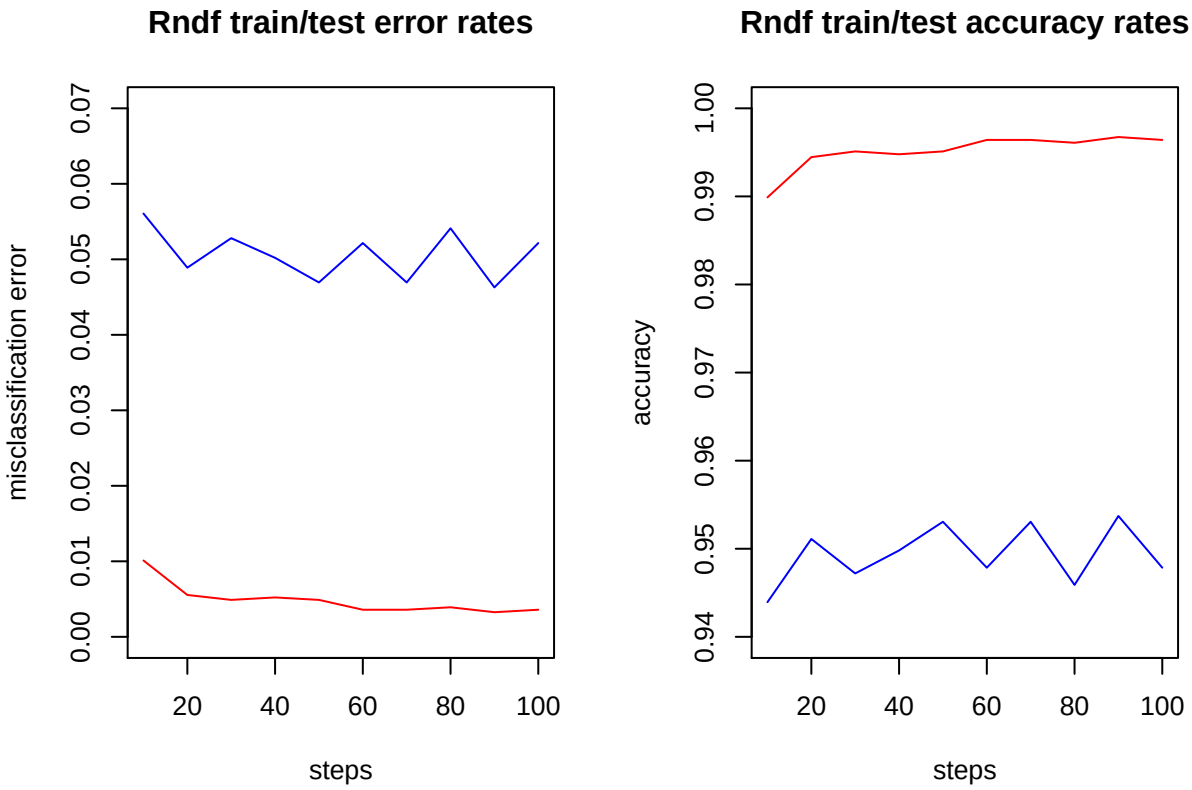




The above plots provide information for about the classification rates and accuracy obtained using ada boost algorithm on train and test data. The red lines represents the classification error/accuracy for train data using a range of 10 to 100 trees and the blue lines is the equivalent for test data. From the plots we can see

that using Ada boost the classification 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.

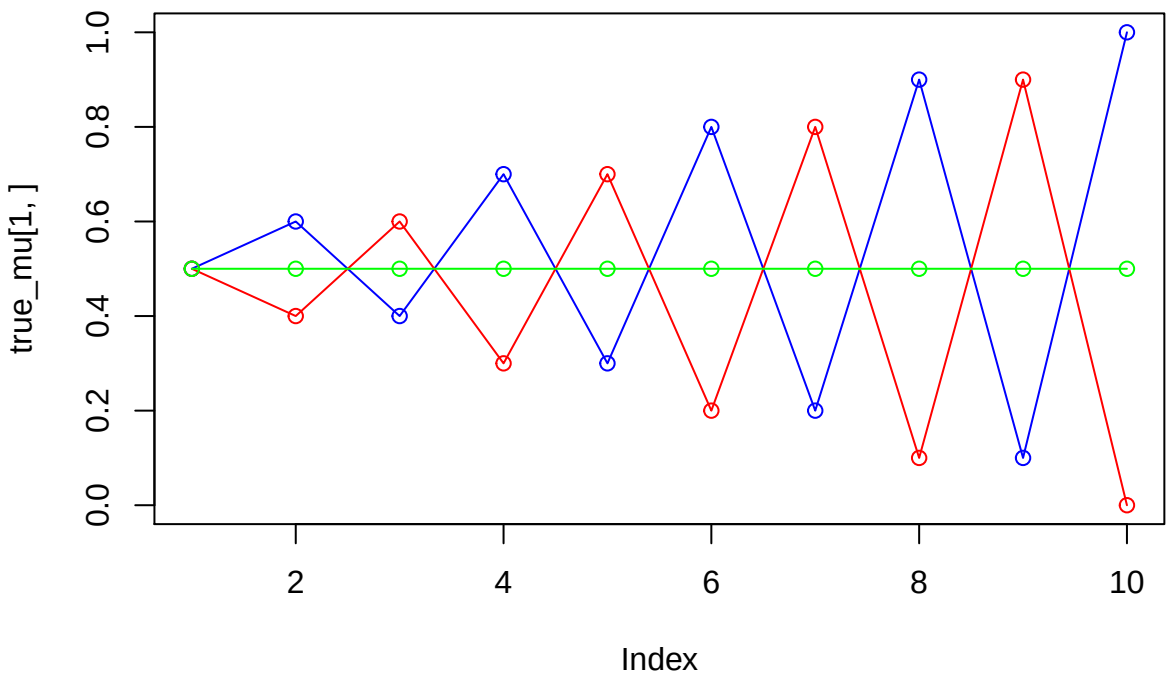
Random Forest Trees



The above plots provide information for about the classification rates and accuracy obtained using random forests algorithm on train and test data. The red lines represents the classification 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 classification rates for training is dropping until 20 number of trees and then is slightly decreases as number of trees increases.As for test data the classification error is fluctuating as the number of trees increases.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 fluctuating as the number of trees increases.

Finally,comparing two methods we can conclude 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 regrading accuracy and classification error. On the other hand, the ada boost model keeps improving as the number of trees grows, because the ada boost model would be modified by each iteration, which represents that more trees are considered into our model to rise the accuracy. Compared with the Ada boost classification trees in this case, the random forest models are better since both test and training error rates (<0.06) of random forest are lower than the ones (>0.07) of Ada boost classification trees.

Assignment 2



As the true μ plot shows, the three conditional distributions are weighted as $\pi = (1/3, 1/3, 1/3)$. And they are all Bernoulli distributions.

```
## [1] 0.4992145 0.5007855
```

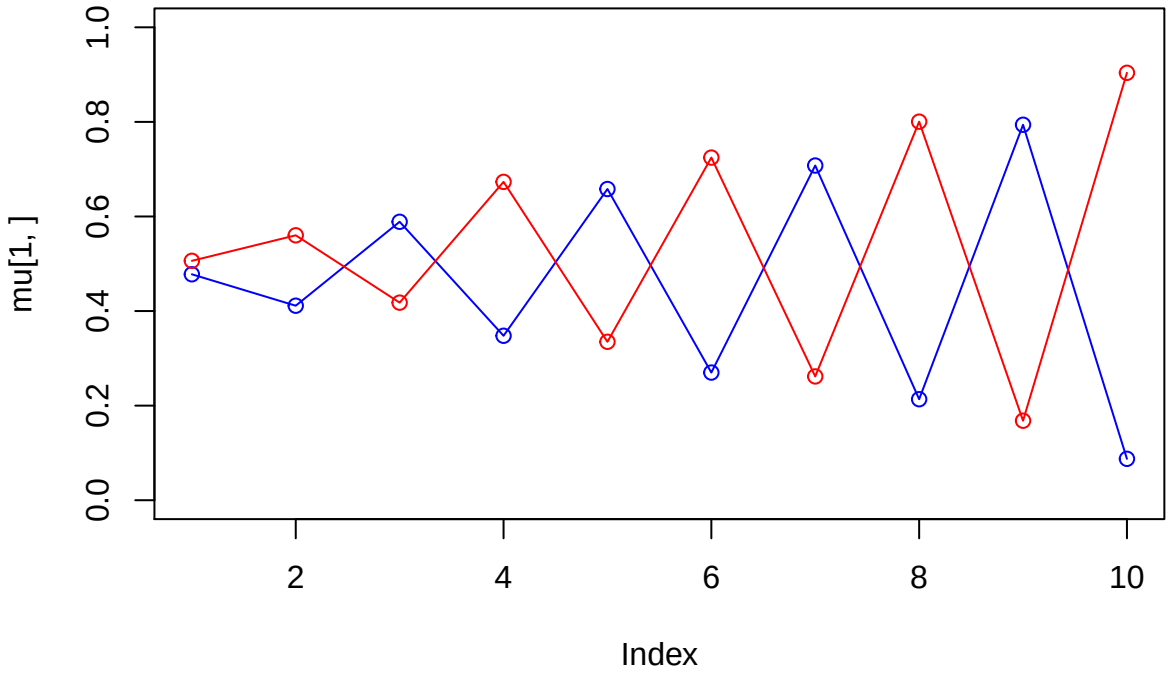
```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4924255 0.4939877 0.4935375 0.5042511 0.5040286 0.4987810 0.5012754
## [2,] 0.4929075 0.4993719 0.5088453 0.5068730 0.5016720 0.4929275 0.5077146
##           [,8]      [,9]     [,10]
## [1,] 0.4971036 0.4982144 0.4987654
## [2,] 0.5095075 0.4924574 0.4992470
```

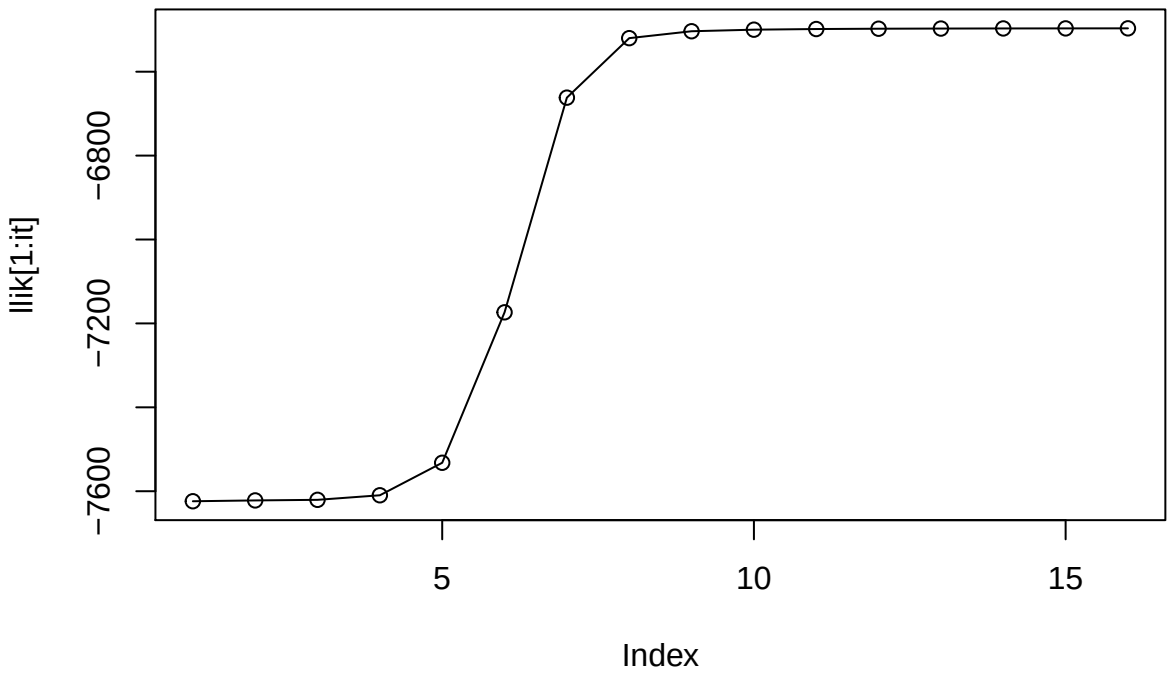
```
## iteration:  1 log likelihood:  -7623.873
## iteration:  2 log likelihood:  -7621.944
## iteration:  3 log likelihood:  -7620.533
## iteration:  4 log likelihood:  -7609.638
## iteration:  5 log likelihood:  -7532.2
## iteration:  6 log likelihood:  -7173.56
## iteration:  7 log likelihood:  -6661.821
## iteration:  8 log likelihood:  -6520.028
## iteration:  9 log likelihood:  -6503.563
## iteration:  10 log likelihood:  -6499.807
## iteration:  11 log likelihood:  -6498.296
## iteration:  12 log likelihood:  -6497.535
## iteration:  13 log likelihood:  -6497.12
## iteration:  14 log likelihood:  -6496.883
```

```
## iteration:  15 log likelihood:  -6496.745
## iteration:  16 log likelihood:  -6496.662
```

```
##         a         a
## 0.4983586 0.5016414
```

```
##        [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## a 0.4777257 0.4113178 0.5887821 0.3477590 0.6580535 0.2699434 0.7077294
## a 0.5061809 0.5601935 0.4177793 0.6731727 0.3350070 0.7245642 0.2617347
##        [,8]      [,9]      [,10]
## a 0.2135532 0.7939559 0.08751855
## a 0.8005199 0.1680987 0.90380175
```





When the K is 2, we can see that the iteration runs 16 times, and the final log likelihood is around -6496. And the weight of these two distribution are around 0.498 and 0.502, which are almost 0.5. And with the μ plot, we can notice that although there are only 2 distributions in this situation, the shapes are almost the same as the two roughest true μ plot shows. And as the loglikelihood value plot shows, the value increases monotonically, and it change much slower after the 8th iteration. And the difference is smaller than 0.1 after 16 iteration.

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

```
##           [,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
```

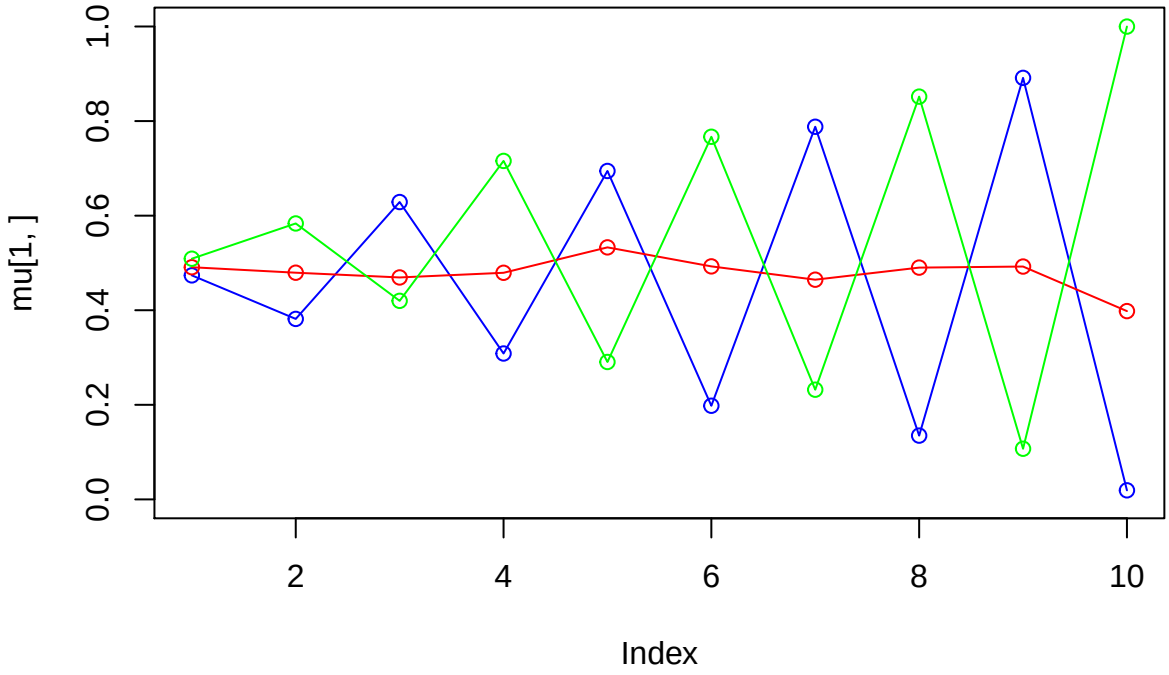
```
## iteration:  1 log likelihood:  -8029.723
## iteration:  2 log likelihood:  -8027.183
## iteration:  3 log likelihood:  -8024.696
## iteration:  4 log likelihood:  -8005.631
## iteration:  5 log likelihood:  -7877.606
## iteration:  6 log likelihood:  -7403.513
## iteration:  7 log likelihood:  -6936.919
## iteration:  8 log likelihood:  -6818.582
## iteration:  9 log likelihood:  -6791.377
## iteration:  10 log likelihood:  -6780.713
```

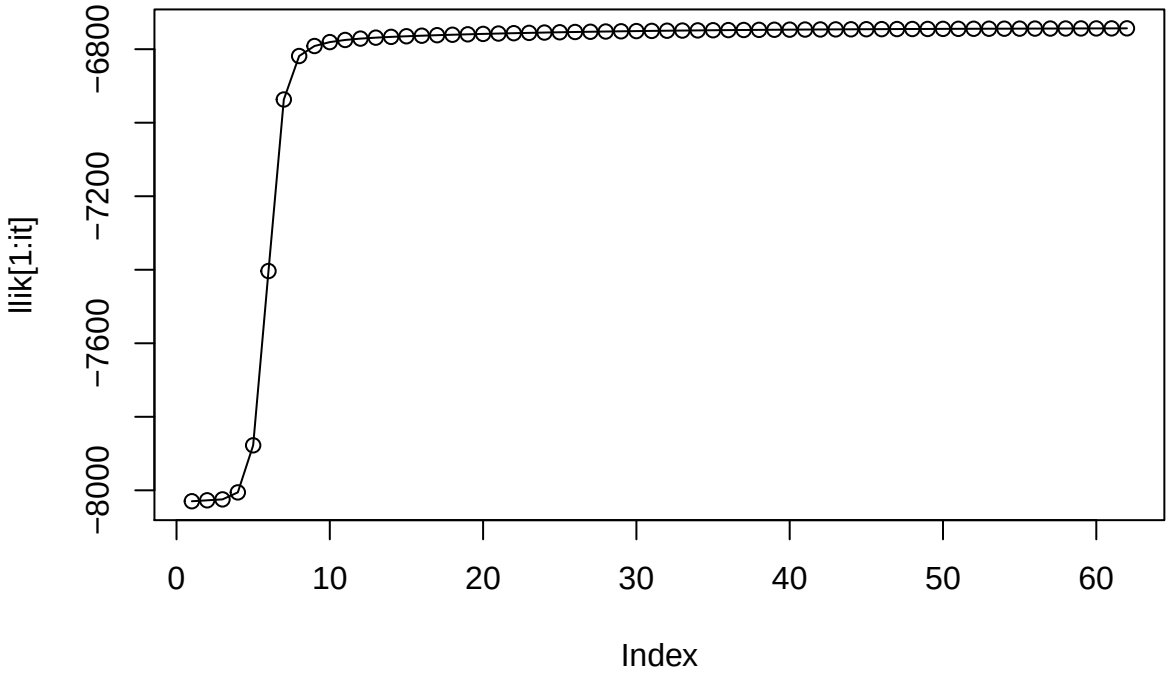
```
## iteration:  11 log likelihood:  -6774.958
## iteration:  12 log likelihood:  -6771.261
## iteration:  13 log likelihood:  -6768.606
## iteration:  14 log likelihood:  -6766.535
## iteration:  15 log likelihood:  -6764.815
## iteration:  16 log likelihood:  -6763.316
## iteration:  17 log likelihood:  -6761.967
## iteration:  18 log likelihood:  -6760.727
## iteration:  19 log likelihood:  -6759.572
## iteration:  20 log likelihood:  -6758.491
## iteration:  21 log likelihood:  -6757.475
## iteration:  22 log likelihood:  -6756.521
## iteration:  23 log likelihood:  -6755.625
## iteration:  24 log likelihood:  -6754.784
## iteration:  25 log likelihood:  -6753.996
## iteration:  26 log likelihood:  -6753.26
## iteration:  27 log likelihood:  -6752.571
## iteration:  28 log likelihood:  -6751.928
## iteration:  29 log likelihood:  -6751.328
## iteration:  30 log likelihood:  -6750.768
## iteration:  31 log likelihood:  -6750.246
## iteration:  32 log likelihood:  -6749.758
## iteration:  33 log likelihood:  -6749.304
## iteration:  34 log likelihood:  -6748.88
## iteration:  35 log likelihood:  -6748.484
## iteration:  36 log likelihood:  -6748.114
## iteration:  37 log likelihood:  -6747.767
## iteration:  38 log likelihood:  -6747.444
## iteration:  39 log likelihood:  -6747.14
## iteration:  40 log likelihood:  -6746.856
## iteration:  41 log likelihood:  -6746.589
## iteration:  42 log likelihood:  -6746.338
## iteration:  43 log likelihood:  -6746.102
## iteration:  44 log likelihood:  -6745.88
## iteration:  45 log likelihood:  -6745.67
## iteration:  46 log likelihood:  -6745.472
## iteration:  47 log likelihood:  -6745.285
## iteration:  48 log likelihood:  -6745.108
## iteration:  49 log likelihood:  -6744.939
## iteration:  50 log likelihood:  -6744.78
## iteration:  51 log likelihood:  -6744.627
## iteration:  52 log likelihood:  -6744.483
## iteration:  53 log likelihood:  -6744.344
## iteration:  54 log likelihood:  -6744.212
## iteration:  55 log likelihood:  -6744.086
## iteration:  56 log likelihood:  -6743.964
## iteration:  57 log likelihood:  -6743.848
## iteration:  58 log likelihood:  -6743.736
## iteration:  59 log likelihood:  -6743.628
## iteration:  60 log likelihood:  -6743.524
## iteration:  61 log likelihood:  -6743.423
## iteration:  62 log likelihood:  -6743.326

##         a         a         a
```

```
## 0.3257375 0.3048066 0.3694558
```

```
##         [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## a 0.4737350 0.3816322 0.6287760 0.3085200 0.6944661 0.1980387 0.7879945
## a 0.4909461 0.4793634 0.4693207 0.4792121 0.5329395 0.4927298 0.4645846
## a 0.5089731 0.5834931 0.4198932 0.7157457 0.2905431 0.7668144 0.2319957
##         [,8]      [,9]      [,10]
## a 0.1348765 0.8913528 0.01915215
## a 0.4901533 0.4923602 0.39817872
## a 0.8516949 0.1071259 0.99983240
```





When K is 3, we can see the iteration runs 62 times and the final log likelihood value is around -6743. And the weight of these three distributions are 0.3257375, 0.3048066 and 0.3694558 which are all around 1/3. And the μ plot is quite similar, especially for the two rough waving paths(blue and green paths), which have the weights 0.3694558 and 0.3257375. And although the remaining path is not so similar to the original one, but it is still trends to be the same. So compared with the true thing, this estimate situation is good enough. And for the loglikelihood value, it also increases monotonically and changes much slower after the 8th iteration.

```
## [1] 0.2491838 0.2499680 0.2500275 0.2508207
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4935375 0.5042511 0.5040286 0.4987810 0.5012754 0.4971036 0.4982144
## [2,] 0.5088453 0.5068730 0.5016720 0.4929275 0.5077146 0.5095075 0.4924574
## [3,] 0.5077926 0.4939841 0.5059821 0.5063490 0.5041462 0.4929400 0.4992362
## [4,] 0.4909320 0.4982342 0.4961948 0.4967226 0.4984220 0.4960055 0.4997165
##           [,8]      [,9]     [,10]
## [1,] 0.4987654 0.4929075 0.4993719
## [2,] 0.4992470 0.5008651 0.4975302
## [3,] 0.4943482 0.4903974 0.5089045
## [4,] 0.5090205 0.5057927 0.4947660
```

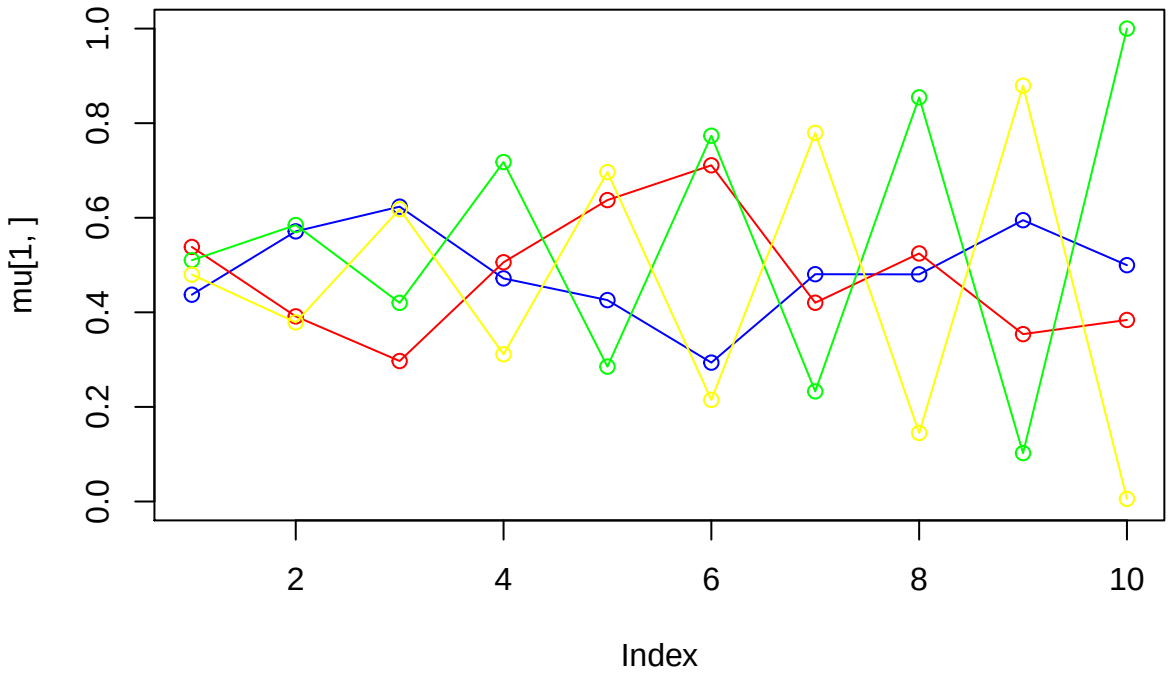
```
## iteration:  1 log likelihood:  -8317.187
## iteration:  2 log likelihood:  -8314.81
## iteration:  3 log likelihood:  -8312.256
## iteration:  4 log likelihood:  -8292.606
## iteration:  5 log likelihood:  -8159.059
## iteration:  6 log likelihood:  -7666.637
## iteration:  7 log likelihood:  -7196.701
## iteration:  8 log likelihood:  -7061.15
```

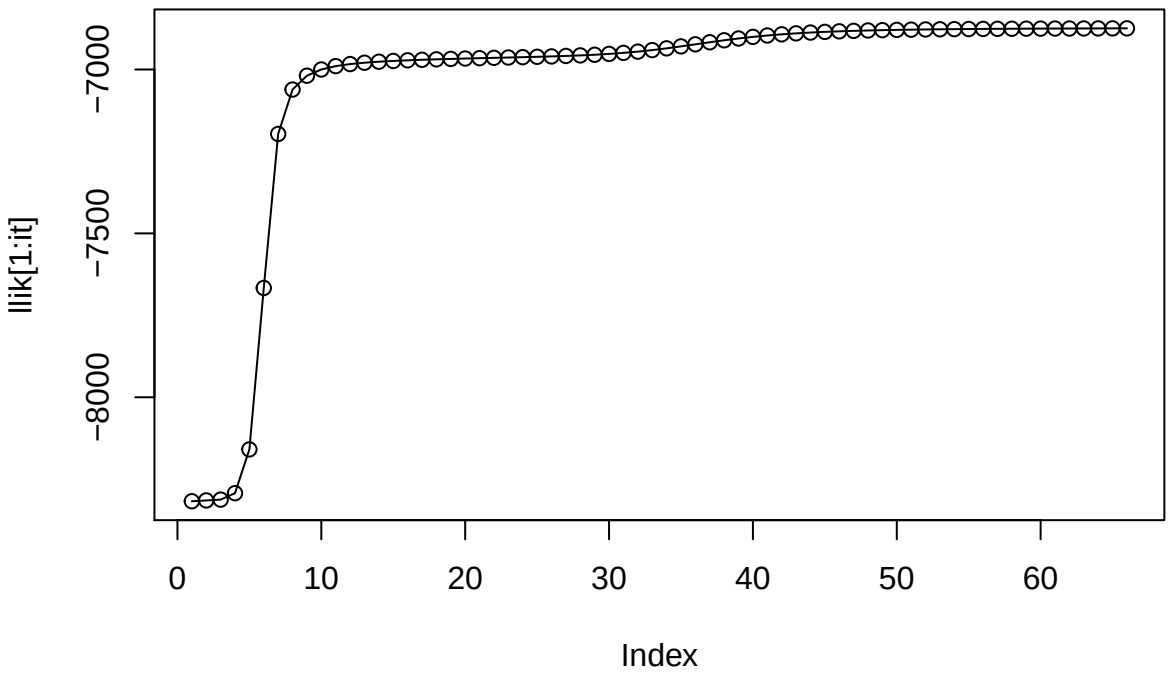
```
## iteration:  9 log likelihood:  -7018.948
## iteration:  10 log likelihood:  -6999.971
## iteration:  11 log likelihood:  -6989.735
## iteration:  12 log likelihood:  -6983.5
## iteration:  13 log likelihood:  -6979.315
## iteration:  14 log likelihood:  -6976.279
## iteration:  15 log likelihood:  -6973.932
## iteration:  16 log likelihood:  -6972.026
## iteration:  17 log likelihood:  -6970.415
## iteration:  18 log likelihood:  -6969.009
## iteration:  19 log likelihood:  -6967.751
## iteration:  20 log likelihood:  -6966.598
## iteration:  21 log likelihood:  -6965.517
## iteration:  22 log likelihood:  -6964.48
## iteration:  23 log likelihood:  -6963.457
## iteration:  24 log likelihood:  -6962.415
## iteration:  25 log likelihood:  -6961.313
## iteration:  26 log likelihood:  -6960.098
## iteration:  27 log likelihood:  -6958.703
## iteration:  28 log likelihood:  -6957.042
## iteration:  29 log likelihood:  -6955.01
## iteration:  30 log likelihood:  -6952.485
## iteration:  31 log likelihood:  -6949.342
## iteration:  32 log likelihood:  -6945.475
## iteration:  33 log likelihood:  -6940.834
## iteration:  34 log likelihood:  -6935.458
## iteration:  35 log likelihood:  -6929.501
## iteration:  36 log likelihood:  -6923.217
## iteration:  37 log likelihood:  -6916.917
## iteration:  38 log likelihood:  -6910.896
## iteration:  39 log likelihood:  -6905.381
## iteration:  40 log likelihood:  -6900.502
## iteration:  41 log likelihood:  -6896.299
## iteration:  42 log likelihood:  -6892.745
## iteration:  43 log likelihood:  -6889.776
## iteration:  44 log likelihood:  -6887.313
## iteration:  45 log likelihood:  -6885.273
## iteration:  46 log likelihood:  -6883.583
## iteration:  47 log likelihood:  -6882.178
## iteration:  48 log likelihood:  -6881.007
## iteration:  49 log likelihood:  -6880.024
## iteration:  50 log likelihood:  -6879.196
## iteration:  51 log likelihood:  -6878.494
## iteration:  52 log likelihood:  -6877.895
## iteration:  53 log likelihood:  -6877.383
## iteration:  54 log likelihood:  -6876.941
## iteration:  55 log likelihood:  -6876.56
## iteration:  56 log likelihood:  -6876.228
## iteration:  57 log likelihood:  -6875.939
## iteration:  58 log likelihood:  -6875.687
## iteration:  59 log likelihood:  -6875.465
## iteration:  60 log likelihood:  -6875.27
## iteration:  61 log likelihood:  -6875.098
## iteration:  62 log likelihood:  -6874.947
```

```
## iteration:  63 log likelihood:  -6874.813
## iteration:  64 log likelihood:  -6874.694
## iteration:  65 log likelihood:  -6874.59
## iteration:  66 log likelihood:  -6874.497
```

```
##         a         a         a         a
## 0.1617479 0.1383200 0.3611987 0.3387334
```

```
##        [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## a 0.4372884 0.5712179 0.6234250 0.4717528 0.4258891 0.2936435 0.4805955
## a 0.5382179 0.3914474 0.2970640 0.5058919 0.6373432 0.7108910 0.4202372
## a 0.5102283 0.5846073 0.4200179 0.7177937 0.2851441 0.7734834 0.2327969
## a 0.4798152 0.3787707 0.6180746 0.3113182 0.6966019 0.2148950 0.7795262
##        [,8]      [,9]      [,10]
## a 0.4802802 0.5948678 0.49971653
## a 0.5245416 0.3538705 0.38388750
## a 0.8545905 0.1023455 0.99999978
## a 0.1449048 0.8793551 0.00553226
```





When K is 4, we can see the iteration runs 66 times and the final log likelihood value is around -6874. And the weight of these four distributions are 0.1617479, 0.1383200, 0.3611987 and 0.3387334. As the μ plot shows, the yellow and green paths are almost the same as the green and blue paths in previous situation. And they have the weights 0.3611987 and 0.3387334 which is also similar to 1/3. But the remaining paths (blue and red lines) have the smaller weight(0.1617479, 0.1383200) and as the plot shows, they seems to be symmetric around the line $\mu = 0.5$. And the log likelihood value increases monotonically and changes much slower after the 8th iteration, just as the first two situations.

As we can see from the reported answers above, when the number of components goes up, the number of running time goes up too, and the final maximum loglikelihood value becomes lower. Compared with those three situations, we can notice there are always two distributions weighted larger or equal to 1/3, which may means this two conditional distributions have vital impacts to the mixtures of multivariate Bernoulli distributions when K is 2, 3 and 4. And when $K = 3$, the estimated result are very closed to the true value. But when K is too few as the situation with $K = 2$ shows, the weight of those two distributions are much higher than the true value (1/3). This may because the distribution may not divided significantly, there may have some other distribution divided from the mix model. And when K is too large as the situation with $K = 4$ shows, two distribution's weights are smaller than the true value. The reason of it may be the over deviation of the model distribution. And it may cause the segmentation of a real distribution just as $K = 4$ situation shows. To sum up, when components is too few, the estimation will cause by the not significant deviation. But when components is too much, it may cause the over deviation. It is very necessary to define an precise number of components for the EM algorithm.

Contributions

Assignment 1 is from *Andreas Christopoulos Charitos* and assignment 2 is from *Jiawei Wu*. Additionally, *Zijie Feng* makes the final conclusion of assignment 1 and 2.

Appendix

```
##Assignment 1#########################################################################
#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(m =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)

#par(mfrow=c(1,2))
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="misclassificatio
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 = "accurac
lines(steps,(1-ada_df[,2]),type="l",col="blue")
```

```r
#random forests
set.seed(12345)
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)
#par(mfrow=c(1,2))
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="misc
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=
lines(steps,(1-rnd_df[,2]),type="l",col="blue")


##Assignment 2#############################################################################

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
set.seed(1234567890)
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)
}
pi
mu
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
  p<-vector()
  for(k in 1:K){
    a<-vector()
    for(j in 1:N){
      a[j]<-prod(mu[k,]^x[j,]*(1-mu[k,])^(1-x[j,]))
    }
    p<-cbind(p,a)
  }
  pk<-pi*t(p)
  pk<-t(pk)
  z<-pk/rowSums(pk)

  # Log likelihood computation.
  # Your code here
  for(n in 1:N){
    v0<-0
    for(k in 1:K){
      u0<-0
      for(i in 1:D){
        u<-x[n,i]*log(mu[k,i])+(1-x[n,i])*(log(1-mu[k,i]))
        u0=u0+u
      }
      v<-z[n,k]*(log(pi[k])+u0)
      v0<-v0+v
    }
    llik[it]<-llik[it]+v0
  }

  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){
    dist<-llik[it]-llik[it-1]
```

```
    if(dist<min_change){
        pi <- colSums(z)/N
        mu <- t(z)%*%(x)/colSums(z)
        break()
    }
  }
  # M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  #
  pi <- colSums(z)/N
  mu <- t(z)%*%(x)/colSums(z)
}
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")
  # points(mu[4,], type="o", col="yellow")
plot(llik[1:it], type="o")
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
set.seed(1234567890)
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
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
  p<-vector()
  for(k in 1:K){
    a<-vector()
    for(j in 1:N){
      a[j]<-prod(mu[k,]^x[j,]*(1-mu[k,])^(1-x[j,]))
    }
    p<-cbind(p,a)
  }
  pk<-pi*t(p)
  pk<-t(pk)
  z<-pk/rowSums(pk)

  # Log likelihood computation.
  # Your code here
  for(n in 1:N){
    v0<-0
    for(k in 1:K){
      u0<-0
      for(i in 1:D){
        u<-x[n,i]*log(mu[k,i])+(1-x[n,i])*(log(1-mu[k,i]))
        u0=u0+u
      }
      v<-z[n,k]*(log(pi[k])+u0)
      v0<-v0+v
    }
    llik[it]<-llik[it]+v0
  }

  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){
    dist<-llik[it]-llik[it-1]
    if(dist<min_change){
        pi <- colSums(z)/N
        mu <- t(z)%*%(x)/colSums(z)
        break()
    }
  }
  # M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  pi <- colSums(z)/N
  mu <- t(z)%*%(x)/colSums(z)
}
```

```
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")
  # points(mu[4,], type="o", col="yellow")
plot(llik[1:it], type="o")
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
set.seed(1234567890)
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)
}
pi
mu
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
  p<-vector()
  for(k in 1:K){
    a<-vector()
```

```
    for(j in 1:N){
      a[j]<-prod(mu[k,]^x[j,]*(1-mu[k,])^(1-x[j,]))
    }
    p<-cbind(p,a)
  }
  pk<-pi*t(p)
  pk<-t(pk)
  z<-pk/rowSums(pk)

  # Log likelihood computation.
  # Your code here
  for(n in 1:N){
    v0<-0
    for(k in 1:K){
      u0<-0
      for(i in 1:D){
        u<-x[n,i]*log(mu[k,i])+(1-x[n,i])*(log(1-mu[k,i]))
        u0=u0+u
      }
      v<-z[n,k]*(log(pi[k])+u0)
      v0<-v0+v
    }
    llik[it]<-llik[it]+v0
  }

  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){
    dist<-llik[it]-llik[it-1]
    if(dist<min_change){
      pi <- colSums(z)/N
      mu <- t(z)%*%(x)/colSums(z)
      break()
    }
  }
  # M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  pi <- colSums(z)/N
  mu <- t(z)%*%(x)/colSums(z)
}
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")
  points(mu[4,], type="o", col="yellow")
plot(llik[1:it], type="o")
```