Machine Learning Lab1 Block2 Group Report

Group A10, combined by Jooyoung Lee

Fengjuan Chen, Jooyoung Lee, Weng Hang Wong

2019 12 3

Assignment 1

Really nice and precise assignment!

Coded by Fengjuan Chen, explained by Jooyoung Lee

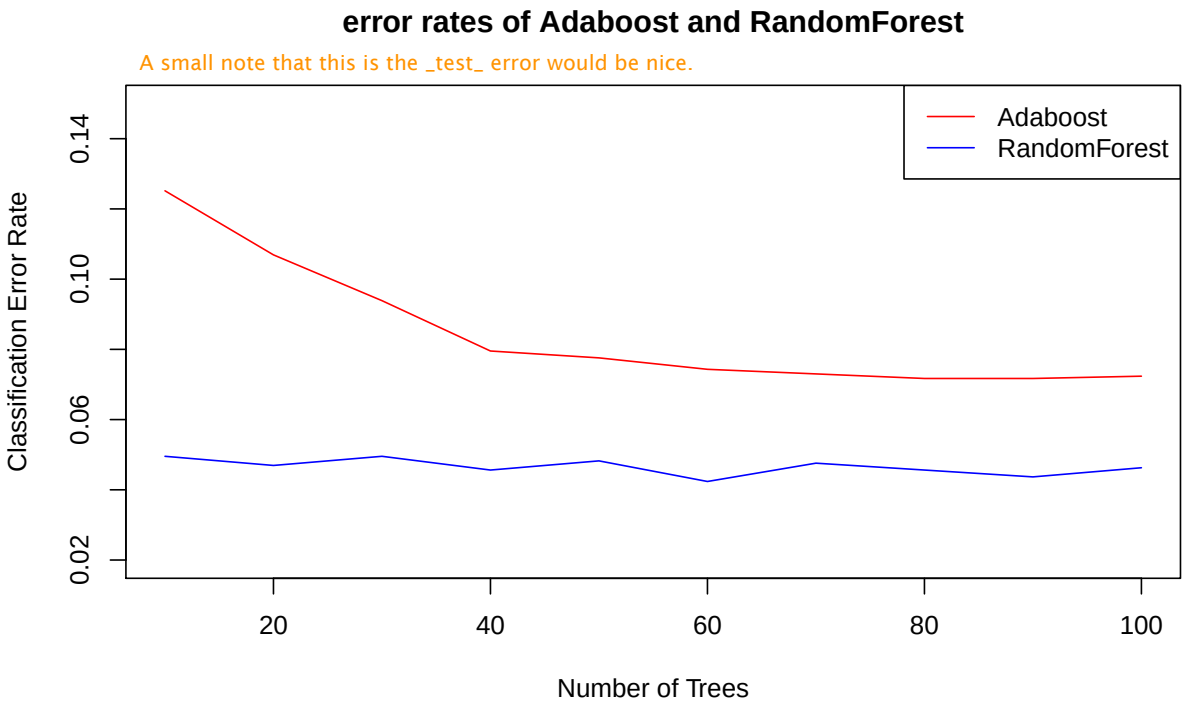


Table 1: The Classifcation Error Rates

treeNumber	AdaboostErrorRate	RandomForestErrorRate
10	0.1251630	0.0495437
20	0.1069100	0.0469361
30	0.0938722	0.0495437
40	0.0795306	0.0456323
50	0.0775750	0.0482399
60	0.0743155	0.0423729
70	0.0730117	0.0475880
80	0.0717080	0.0456323
90	0.0717080	0.0436767
100	0.0723598	0.0462842

As the number of trees grows, error rate decreases if Ada boost algorithm is utilized. However, from the number of trees equals to 90, error rate does not show decrease in its amount. Unlike Ada boost algorithm, random forest algorithm does not show very significant decrease in its error rate as the number of trees grows; however, this algorithm nearly always showed error rate less than 6%.

Ada boost algorithm makes the error rate lower by updating the weights for wrongly classified target through iterations. Then it follows majority rule to classify the target, which is based on the result turned out in each iteration. From the point when all targets are correctly classified, weight does not change - error rate stays the same from this number of iteration.

Random forest algorithm picks random bootstrap samples from the training data to build the model. Then it grows a number of decision trees based on `ntree` argument to classify the target based on majority rule. Since the samples are selected randomly, error rate may fluctuate at different number of trees. The error rate decreases as the number of trees increases and stops decreasing when the number of trees exceeds certain value.

In case of classifying spam mail, random forest method seems to be more effective. Although it does not show the decrease in error rate even after increasing the number of trees, its error rates are stable at the level lower than the one comes from Ada boost algorithm.

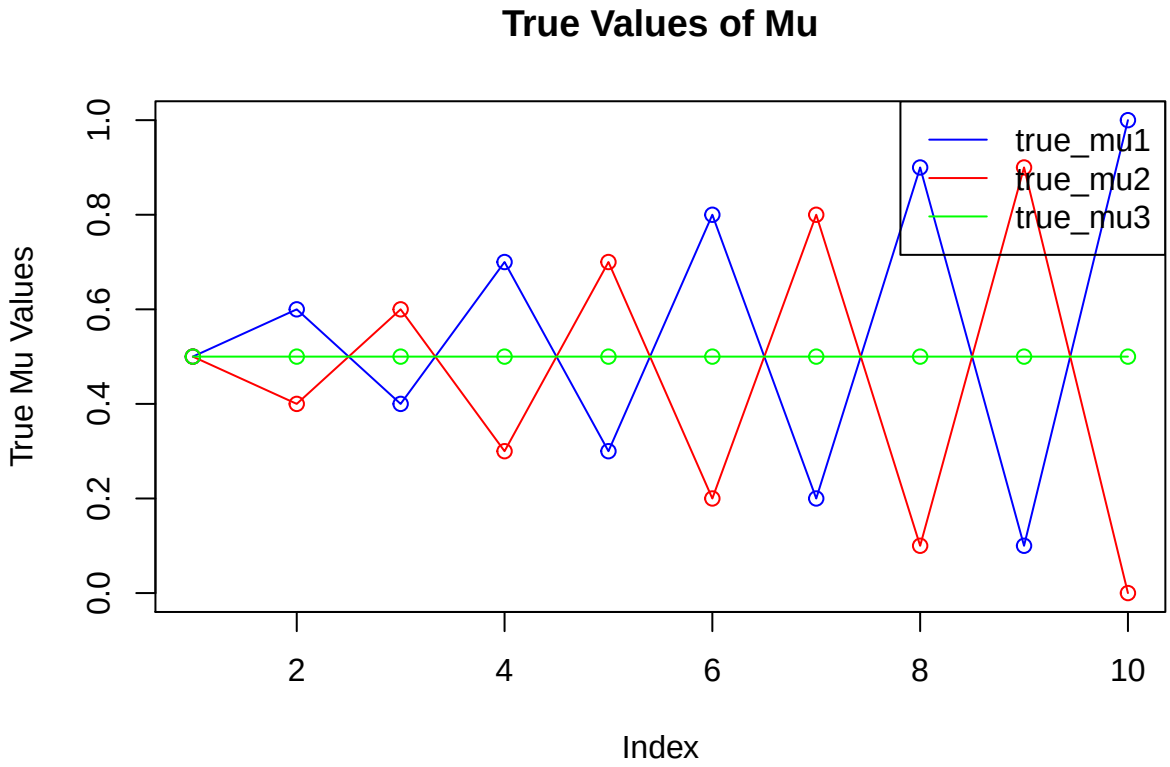
Assignment 2

Your log likelihood is wrong. Check code for comments.

You should add plots for all numbers of assumed components, so 2, 3 and 4.

Coded and explained by Weng Hang Wong and Jooyoung Lee

We use the EM algorithm for this multivariate Benoulli distributions. And the belowed code is using *K=3* as one of the result. That is just the true mu, not your prediction with K=3.

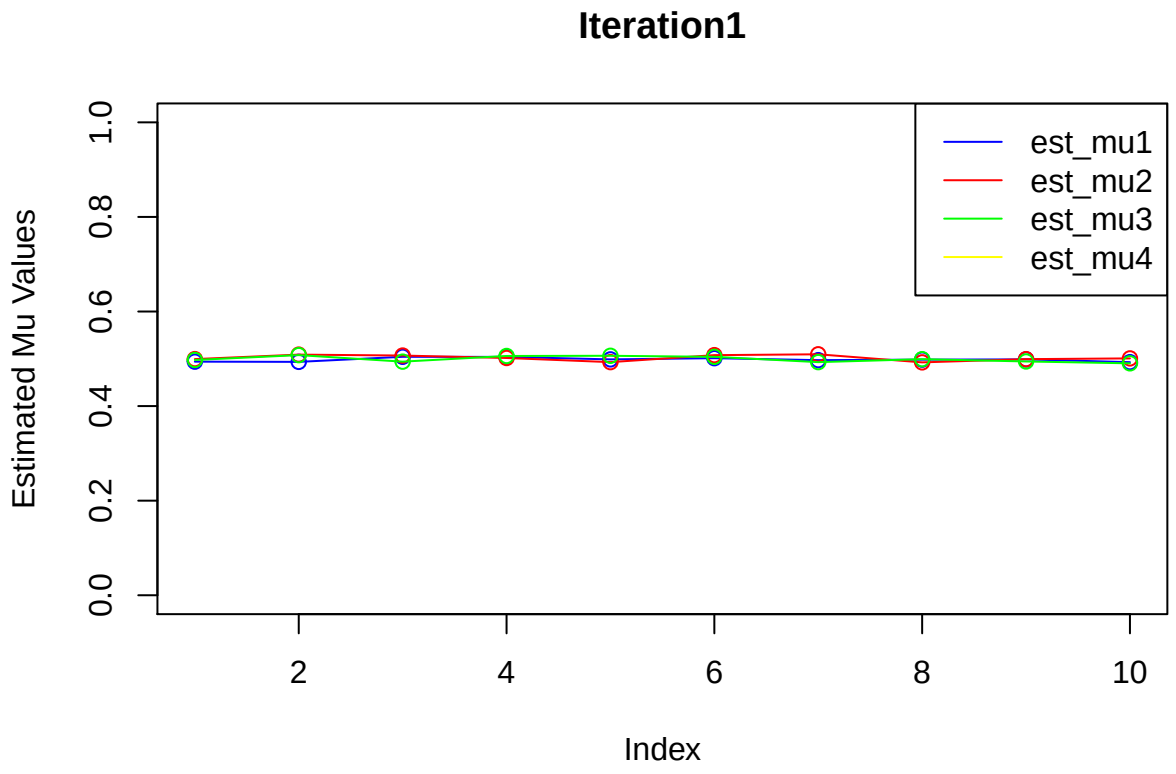


K=3

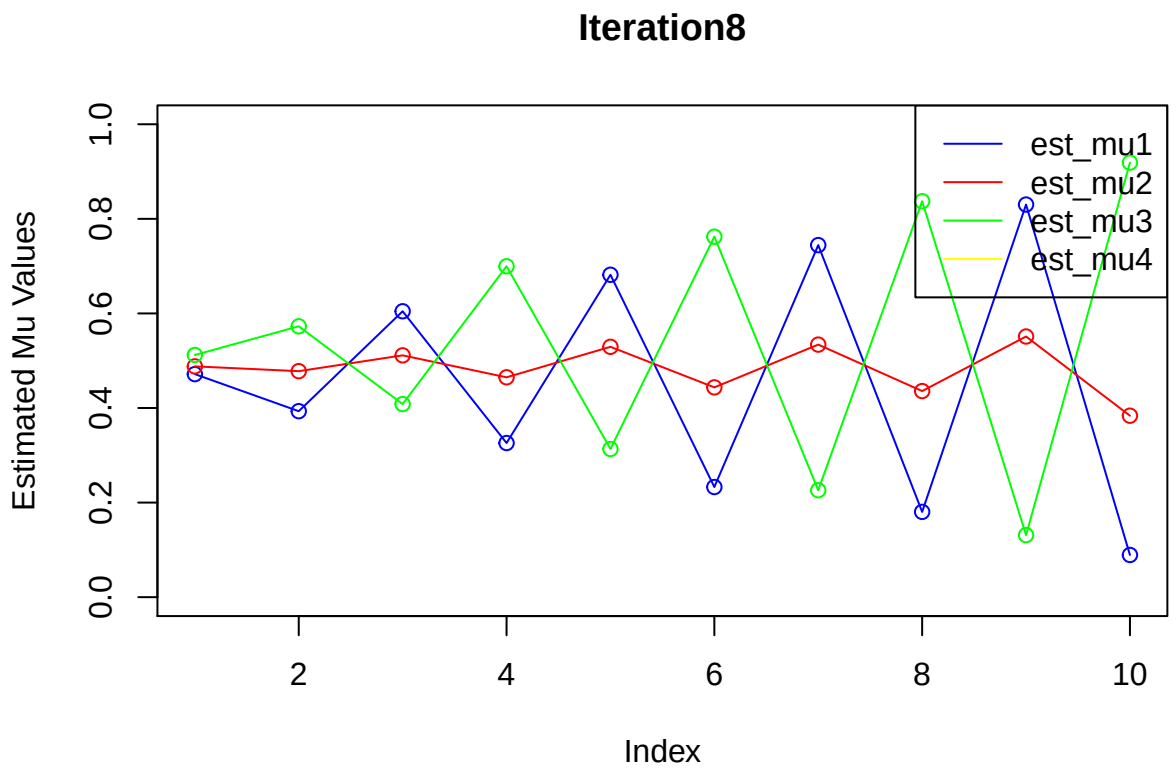
From the 62 iteration mu estimations graphs, we can see the relationship between the maximum likelihood and the estimated mu's of each iteration. On the first iteration, values of mu on each dimension are quite similar to each other, but along with the increasing iteration, we can see the values of mu on each dimension becomes more diverse. Estimated values of pi and mu are not very similar to the true values since EM algorithm finds local maximum of log-ilkelihook instead of the global maximum.

The same procedure can do the same with the increasing K component, the mu increases following the K increases, and the mu estimation will become even more diverse when K is increasing.

```
##
##  Initial pi:

## [1] 0.3326090 0.3336558 0.3337352

##
##  Initial 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
```

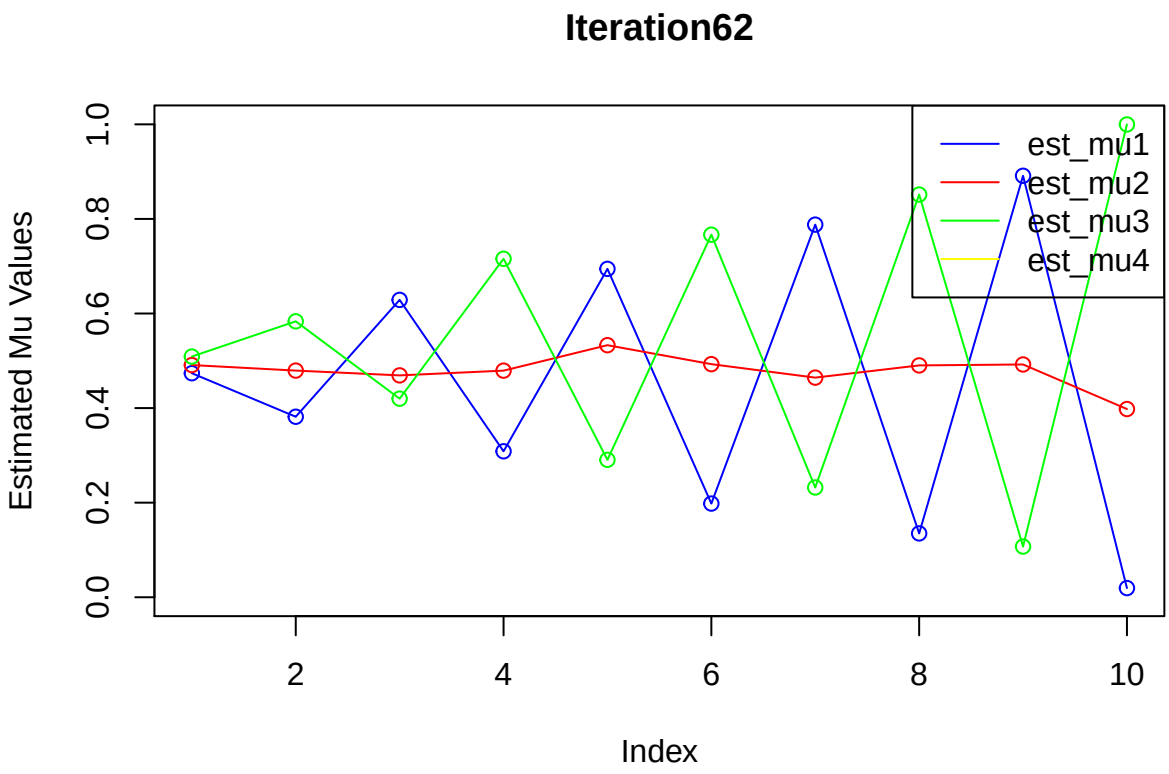


```
## iteration:  1 log likelihood:  -8029.723
## iteration:  2 log likelihood:  -8027.183
## iteration:  3 log likelihood:  -8024.696
## iteration:  4 log likelihood:  -8005.631
## iteration:  5 log likelihood:  -7877.606
## iteration:  6 log likelihood:  -7403.513
## iteration:  7 log likelihood:  -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
```

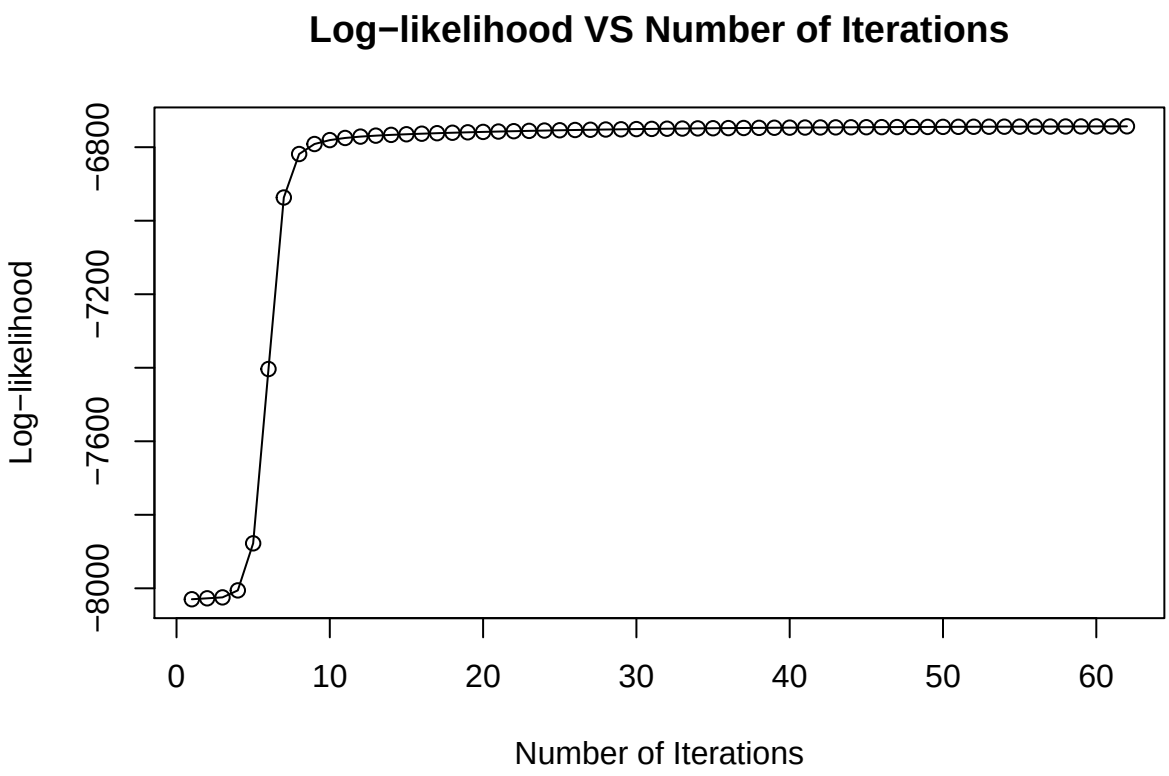



```
## iteration:  62 log likelihood:  -6743.326
```

We have tried different numbers of components. When K=2, the log-likelihood is increasing along with 16 iterations. In this case, when K=3, the log-likelihood is increasing with 62 iteration and when K=4, the log-likelihood is increasing with 66 iterations.

The number of iterations until reaching the local maximum log-likelihood are increasing, while the maximum log-likelihood itself is decreasing as the number of components are increasing.

```
##
##  pi values after running the algorithm :

## [1] 0.3259592 0.3044579 0.3695828

## mu values after running the algorithm :

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4737193 0.3817120 0.6288021 0.3086143 0.6943731 0.1980896 0.7879447
## [2,] 0.4909874 0.4793213 0.4691560 0.4791793 0.5329895 0.4928830 0.4643990
## [3,] 0.5089571 0.5834802 0.4199272 0.7157107 0.2905703 0.7667258 0.2320784
##           [,8]      [,9]      [,10]
## [1,] 0.1349651 0.8912534 0.01937869
## [2,] 0.4902682 0.4922194 0.39798407
## [3,] 0.8516111 0.1072226 0.99981353
```




```
## iteration:  62 log likelihood:  -6743.326
```

Appendix

```
# Assignment 1
library(mboost)
library(randomForest)
library(knitr)
RNGversion('3.5.1')
set.seed(12345)
sp <- read.csv2("C:/Users/Young/Documents/ML/Lab1 b/spambase.csv")
sp$Spam <- as.factor(sp$Spam)
n <- dim(sp)[1]
id <- sample(n,floor(n*0.6667))
train <- sp[id,]
test <- sp[-id,]
treenum <- seq(10,100,10)
errormatrix <- matrix(0,ncol = 3,nrow = 10)
j <- 0
for (i in treenum) {
  j <- j+1
  errormatrix[j,1] <- i
  # construct adaboost by mstop=10~100
```

```
  # calculate the predict error in test
  # store these error rate in an errormatrix
  ada_model <- blackboost(train$Spam~.,
                    data = train,family = AdaExp(),
                    control = boost_control(mstop = i))
  ada_predict <- predict(ada_model,newdata = test,
                         type = "class")
  error <- table(ada_predict,test$Spam)
  error_rate <- (error[1,2]+error[2,1])/sum(error)
  errormatrix[j,2] <- error_rate

  # construct randomforest by ntree=10~100
  # calculate the predict error in test
  # store these error rate in an errormatrix
  ranfore_model <- randomForest(train$Spam~.,
                          data = train,ntree=i)
  ran_predict <- predict(ranfore_model,newdata = test,
                         type = "class")
  error_2 <- table(ran_predict,test$Spam)
  error_rate_2 <- (error_2[1,2]+error_2[2,1])/sum(error_2)
  errormatrix[j,3] <- error_rate_2
 }
colnames(errormatrix) <- c("treeNumber","AdaboostErrorRate",
                           "RandomForestErrorRate")

plot(errormatrix[,1],errormatrix[,2],
     type = "l", ylim = c(0.02,0.15),
     xlab = "Number of Trees",
     ylab = "Classification Error Rate",
     main = "error rates of Adaboost and RandomForest",
     col="red")
lines(errormatrix[,1],errormatrix[,3],
      col="blue")
legend("topright",legend = c("Adaboost","RandomForest"),
        col = c("red","blue"),lty = c(1,1))

kable(caption = "The Classifcation Error Rates",
      errormatrix)
```

```
# Assignment 2
RNGversion('3.5.1')
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 Values of Mu", ylab="True Mu Values")
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
legend("topright", legend=c("true_mu1", "true_mu2", "true_mu3"),
       col=c("blue", "red", "green"), lty=c(1,1,1))

# 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)
}
cat("\n", "Initial pi: ", "\n")
pi
cat("\n", "Initial mu: ", "\n")
mu

for(it in 1:max_it) {
  this <- paste0("Iteration", it)
  plot(mu[1,], type="o", col="blue", ylim=c(0,1), ylab="Estimated Mu Values", main= this)
  points(mu[2,], type="o", col="red")
  points(mu[3,], type="o", col="green")
  # points(mu[4,], type="o", col="yellow")
  legend("topright", legend=c("est_mu1", "est_mu2", "est_mu3", "est_mu4"),
         col=c("blue", "red", "green", "yellow"), lty=c(1,1,1,1))
  Sys.sleep(0.5)
  # E-step: Computation of the fractional component assignments

  # using N number of training data to produce z matrix
  for (n in 1:N) {
    #produce matrix with p(x|muk)
    x_muk <- append(rep(1, times=K), 0)
    #iteration for each component
    for (k in 1:K) {
```

```
    #interation for each dimension
    for (d in 1:D) {
      #bernoulli probability of x given mu_k
      x_muk[k] <- x_muk[k]*(mu[k,d]^x[n,d])*((1-mu[k,d])^(1-x[n,d]))
    }
    #assignning weighted value of bernoulli probability
    x_muk[k] <- pi[k] * x_muk[k]
    x_muk[K+1] <- x_muk[K+1] + x_muk[k]
  }
  #assigning z value
  for (k in 1:K){
    z[n,k] <- x_muk[k]/x_muk[K+1]
  }
}

#Log likelihood computation.

for (n in 1:N){
  for (k in 1:K) {
    #calculating the second summation part
    second_part <- 0
    for (d in 1:D) {
      second_part <- second_part + ((x[n,d]*log(mu[k,d])) +
                                      ((1-x[n,d])*log(1-mu[k,d])))
    }
    #log-likelihood for each iteration
    #after K-number summation is done, for loop on n takes place so all summation
    #is working
    llik[it] <- llik[it]+(z[n,k]*(log(pi[k])+second_part))
  }
}
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the lok likelihood has not changed significantly

#iteration number must be at least 2 for comparison
if (it != 1) {
  #since the difference can be negative value, absolute difference must be compared
  if (abs(llik[it]-llik[it-1]) < min_change ) {break}
}

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

for (k in 1:K) {
  pi[k] <- sum(z[,k]) / N
}

for (k in 1:K) {
  mu[k,] <- 0
  for (n in 1:N) {
    #this is for the numerator
    mu[k,] <- mu[k,] + (x[n,]*z[n,k])
```

Your likelihood is wrong. The first part is logged (second_part variable for you), so you have to take exp() of that again to actually your probability, then you multiply by Pi, sum and THEN log. But why do you log the whole formula again then? If you look at formula 9.14 in Bishops book, you see that the formula is already the log likelihood. So it's pi * p(x), which is already for the _log_likelihhod.

```
    }
    #divide it with the denominator
    mu[k,] <- mu[k,] / sum(z[,k])
  }
}

cat( "\n", "pi values after running the algorithm :", "\n")
pi
cat("\n", "\n")
cat("mu values after running the algorithm :", "\n")
mu
plot(llik[1:it], type="o", main="Log-likelihood VS Number of Iterations",
     ylab="Log-likelihood", xlab = "Number of Iterations")
```