Lab 1 Block2

Ahmed Alhasan

12/03/2019

```
## $`Adaboost Least Error`
## [1] 0.0593
##
## $`Adaboost Confusion Matrix`
##          Actual
## Predected   0   1
##         0 882  51
##         1  40 561
##
## $`Random Forest Least Error`
## [1] 0.0469
##
## $`Random Forest Confusion Matrix`
##          Actual
## Predected   0   1
##         0 889  44
##         1  33 568
```

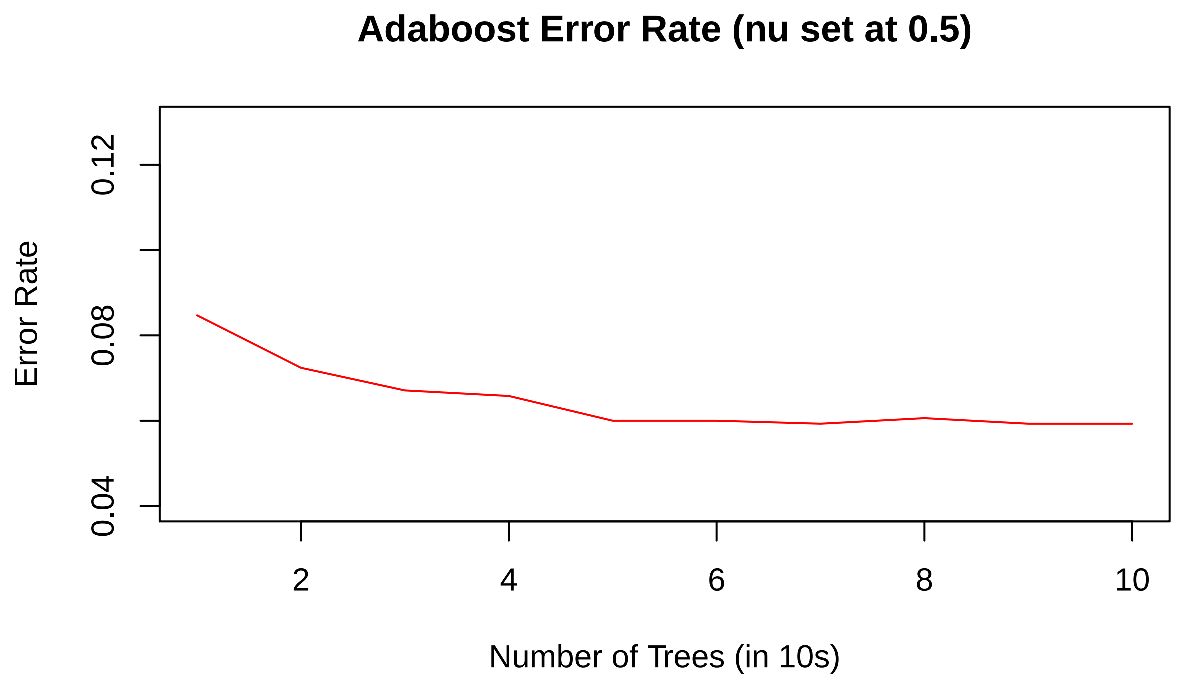
	L=0.1	L=0.2	L=0.3	L=0.4	L=0.5	L=0.6	L=0.7	L=0.8	L=0.9	L=1	R.F.
10 Ts	0.1330	0.1128	0.0900	0.0913	0.0847	0.0808	0.0789	0.0919	0.0874	0.1004	0.0593
20 Ts	0.1063	0.0802	0.0743	0.0724	0.0724	0.0711	0.0737	0.0730	0.0704	0.0698	0.0515
30 Ts	0.1017	0.0776	0.0763	0.0678	0.0671	0.0632	0.0665	0.0717	0.0645	0.0671	0.0528
40 Ts	0.0834	0.0769	0.0724	0.0665	0.0658	0.0606	0.0606	0.0684	0.0574	0.0658	0.0548
50 Ts	0.0808	0.0750	0.0717	0.0658	0.0600	0.0600	0.0600	0.0671	0.0574	0.0652	0.0508
60 Ts	0.0802	0.0717	0.0678	0.0632	0.0600	0.0587	0.0600	0.0619	0.0574	0.0619	0.0495
70 Ts	0.0782	0.0704	0.0665	0.0600	0.0593	0.0554	0.0574	0.0678	0.0561	0.0600	0.0515
80 Ts	0.0769	0.0665	0.0652	0.0587	0.0606	0.0548	0.0567	0.0632	0.0554	0.0580	0.0469
90 Ts	0.0737	0.0658	0.0639	0.0580	0.0593	0.0535	0.0574	0.0613	0.0548	0.0574	0.0541
100 Ts	0.0737	0.0652	0.0619	0.0574	0.0593	0.0515	0.0554	0.0587	0.0535	0.0567	0.0502

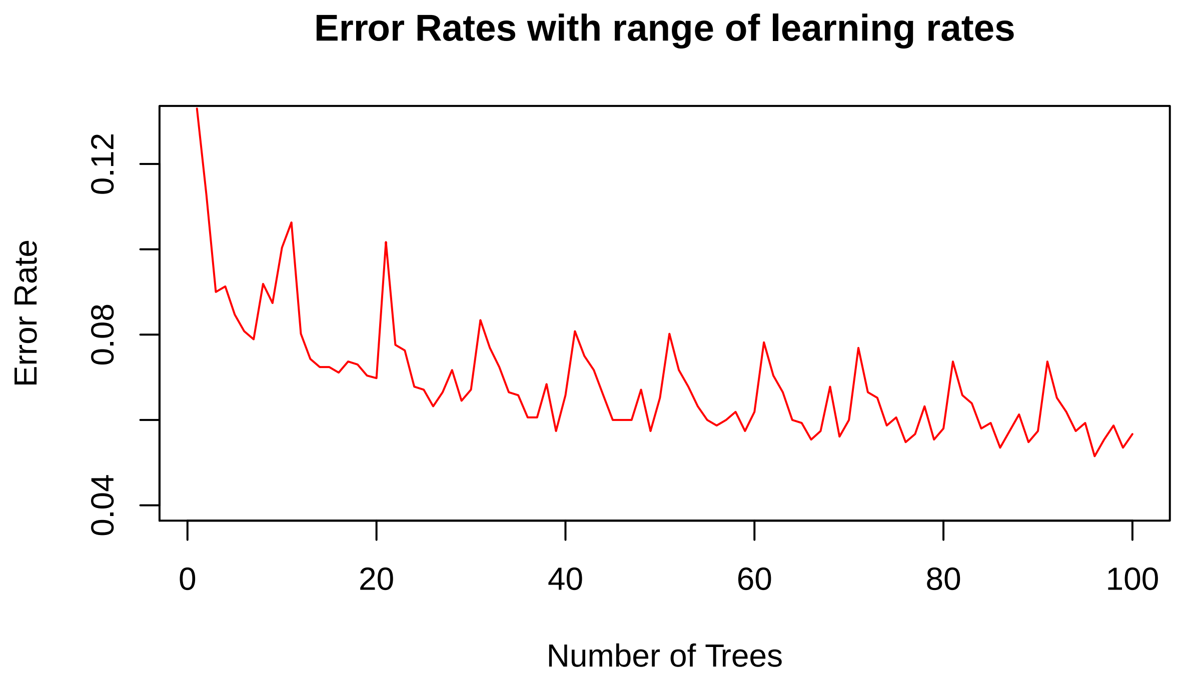
Analysis:

- By going through different values of nu (learning rate / shrinkage parameter) the default value set by the boost_control() function is 0.1 which contradicts with value of 0.5 set in the AdaExp() function (View(AdaExp)). nu had to be set to 0.5 manually to obtain the Exponential Loss function, however cross checking with other packages my be required.

$$w_i \leftarrow w_i.exp\ [\alpha_m.I(y_i \neq G_m(x_i))],\ i = 1, 2, ..., N$$

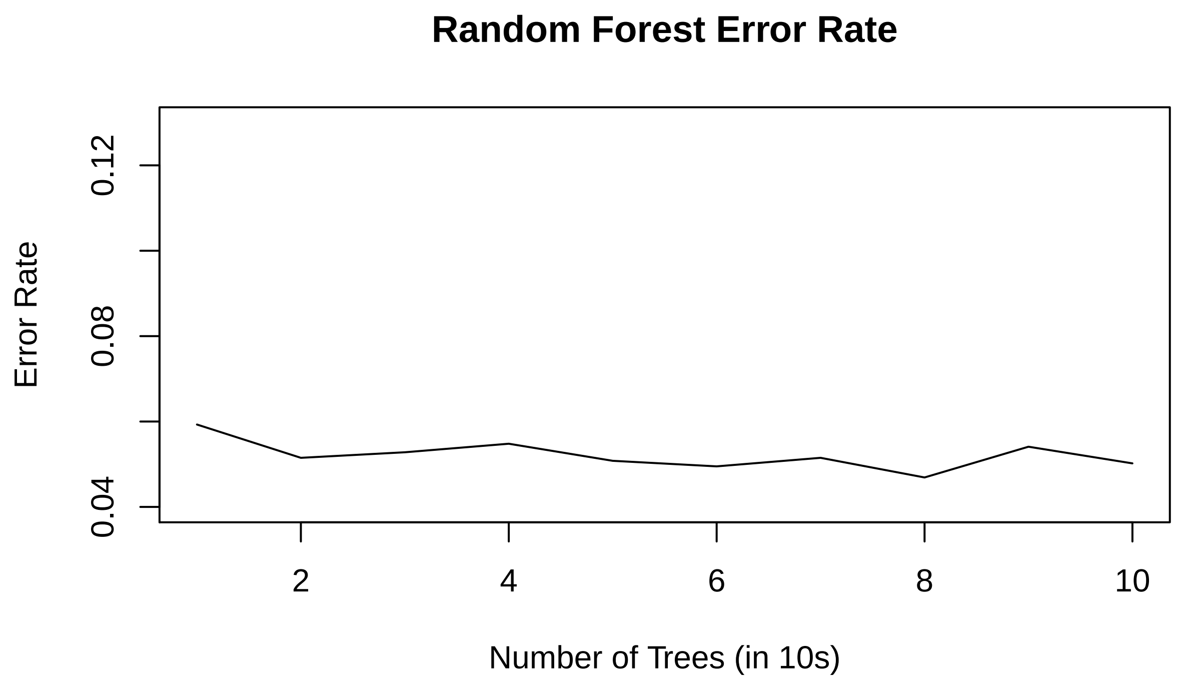
$$\alpha_m = L\ log(\frac{1 - err_m}{err_m})\ where\ L \leq 1\ is\ the\ learning\ rate$$

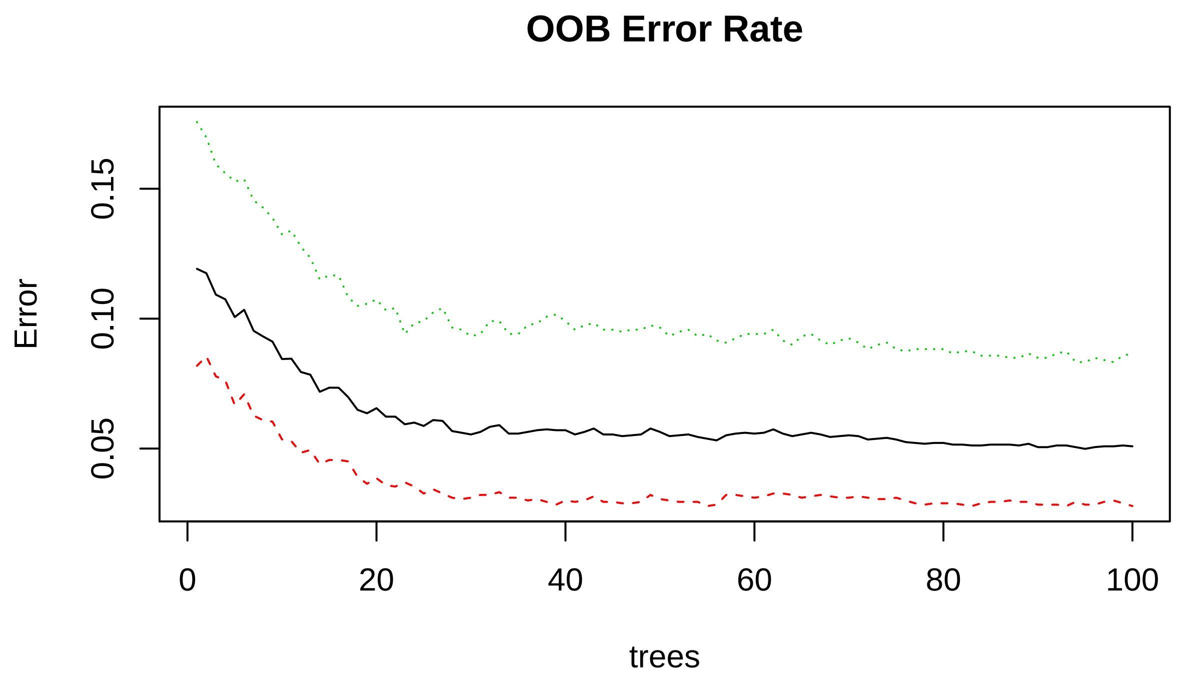




- Adaboost (by setting nu = 0.5) start to blateau when the number of iterations (stumbs/trees) is around 50 and the least error achieved is 0.0593 which is higher but not far from Random Forest least error.
- Using different values for nu the least error rate (0.0515) achieved when number of iterations was highest (100) and Learning Rate = 0.6, which is far from (0.1330) using only 10 stumbs and learning

rate = 0.1, the general trend is that the error decrease with increasing either the number of iterations or Learning Rate, or both. However, the increase in learning rate may cause overfitting.





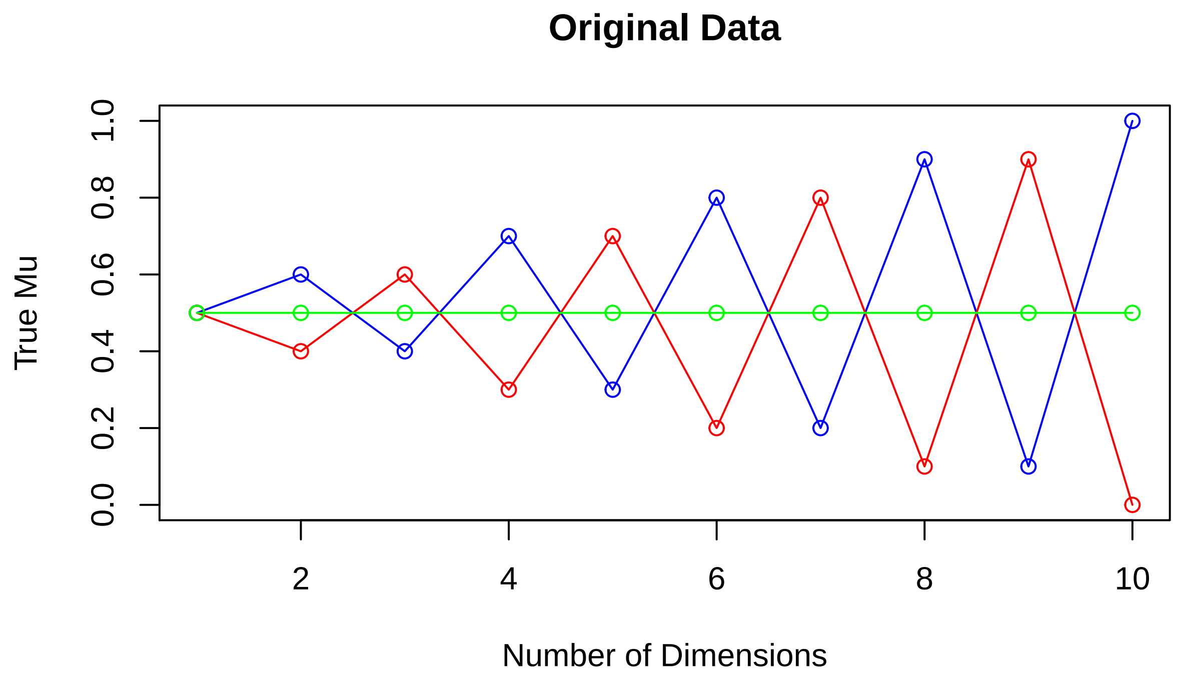
- Random forest performance was more stable and did not fluctuate much despite the number of trees used,it ranged between (0.0469) which is the least error achieved using 80 trees and (0.0593) the highest eror using only 10 trees which is exactly same as Adaboost. This fluctuation can be explained by the randomness of bootstrab sampling and feature sampling.

Char4	173.79
Char5	134.01
Word7	110.60
Capitalrun1	86.98
Word16	85.41
Capitalrun2	83.91
Word21	82.80
Word25	61.96
Capitalrun3	60.19
Word24	52.33
Word5	42.94
Word23	38.64
Word26	34.44
Word19	34.33
Word27	30.42
Word46	26.03
Word3	20.49
Word17	19.32
Word8	18.69
Char2	16.98
Word37	16.68
Word11	16.10
Word12	15.32
Word18	15.04
Word45	12.89
Word2	10.44
Word10	9.78
Word20	9.46
Char1	8.92
Word42	8.33
Word6	8.05
Word28	7.89
Word9	6.85
Word39	6.12
Word36	5.35
Word13	5.22
Word1	5.05
Word30	4.95
Word35	4.82
Word14	4.30
Word33	3.80
Char3	3.52
Char6	3.33
Word22	3.30
Word15	3.21
Word31	3.13
Word44	3.09
Word29	2.25
Word43	2.14
Word48	1.74

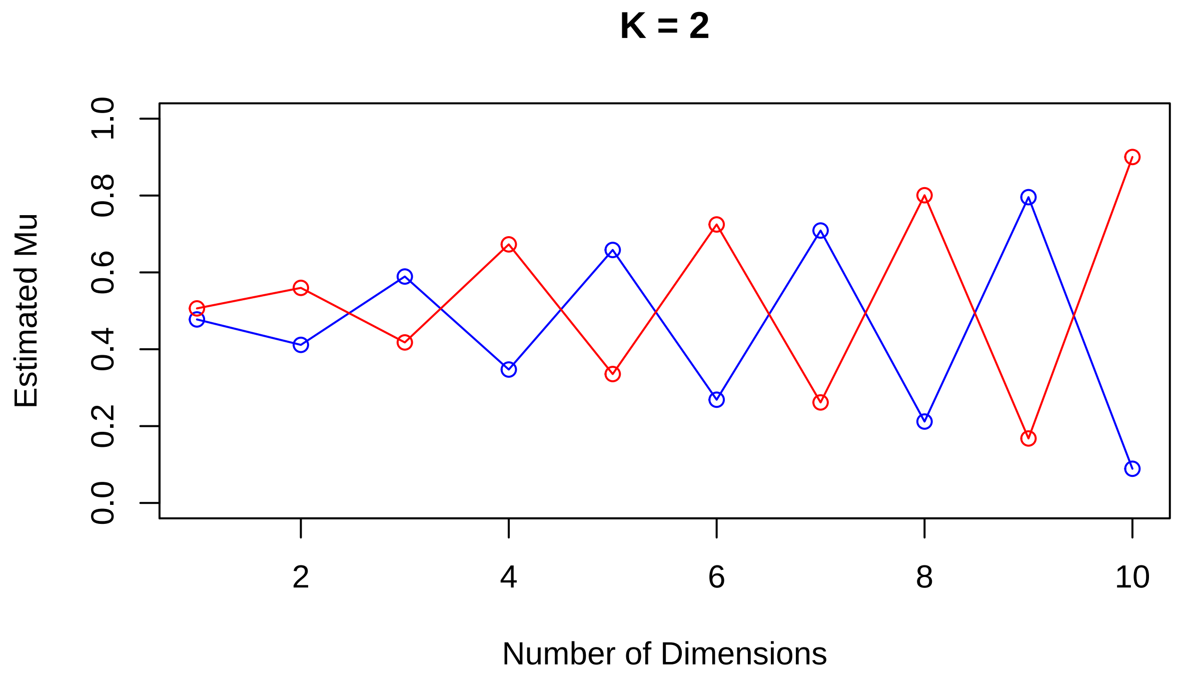
Word40	1.70
Word4	1.38
Word41	1.33
Word34	1.23
Word32	1.07
Word38	0.78
Word47	0.18

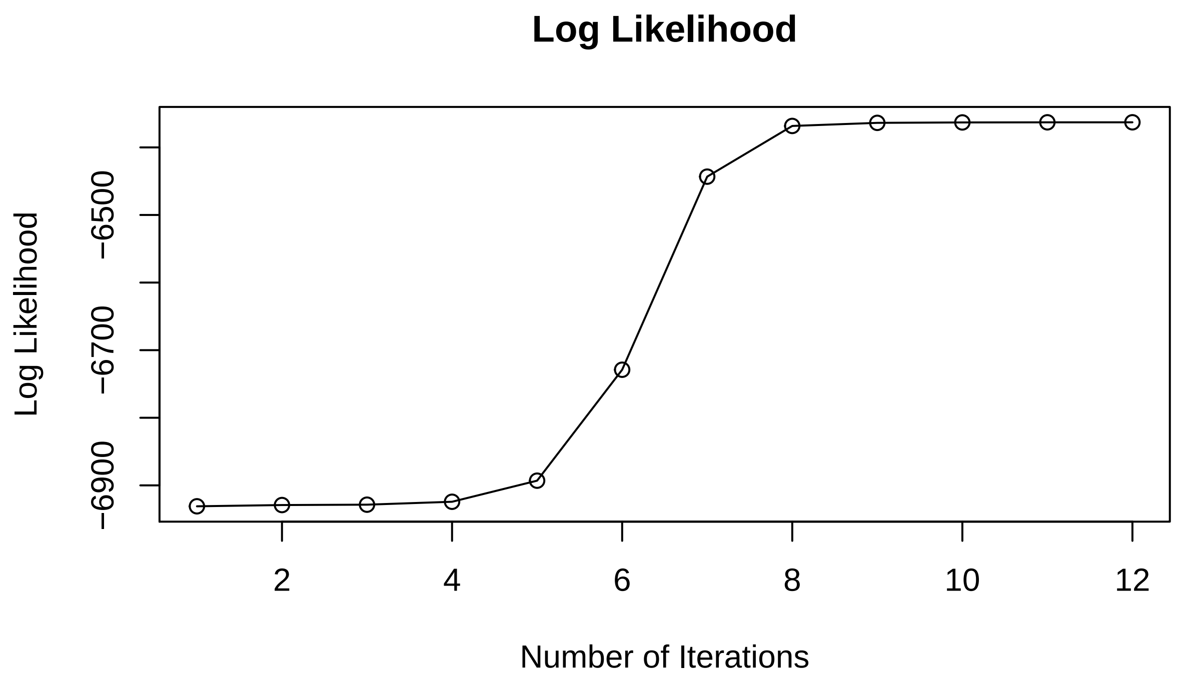
- Number of variables tried at each split is 7, with many relevant variables and relatively few noise variables, the probability of a relevant variable being selected at any split is is very high, that's why the error rate is better in the Random Forest model.
- Out of Bag error(OOB) is 5.05% stablizes at around 40 trees, which measures the misclassification rate of the OOB observations using the trees that they were not trained on. which correlates with missclassification rate of the whole random forest.
- The most significant variables that lead to the least impurity when selected are Char4, Char5, Word7, Capitalrun2 and Word16 using Gini index which measures the effect on prediction were this variable not available.

2. Mixture Models

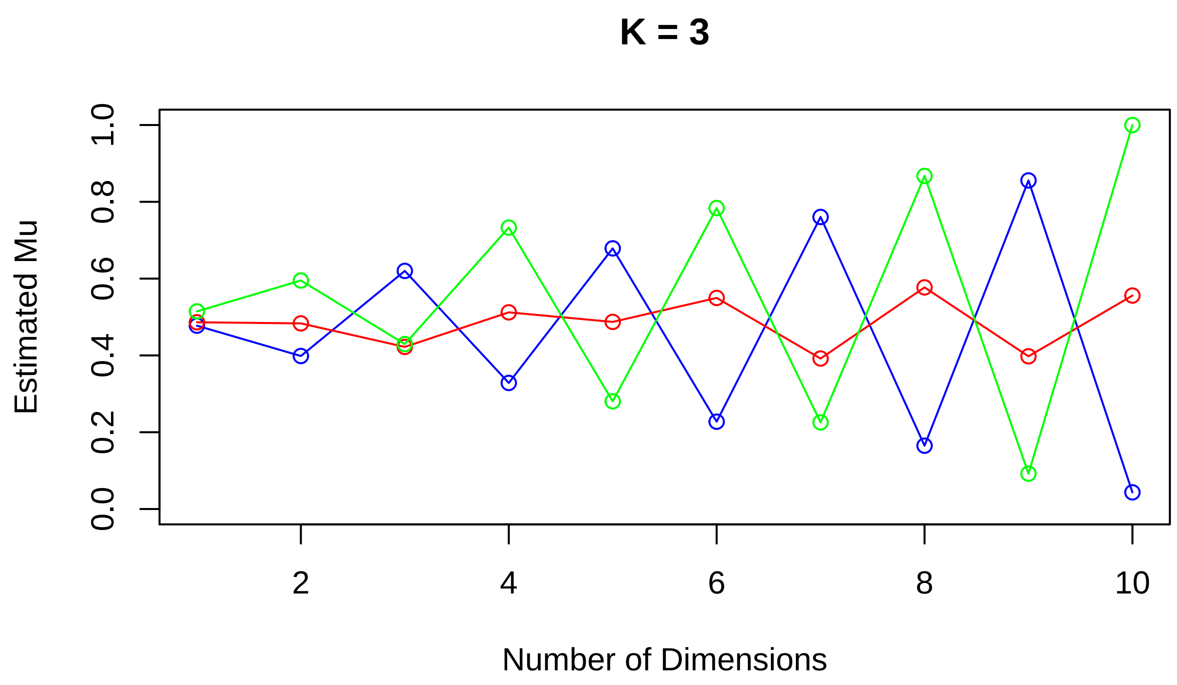


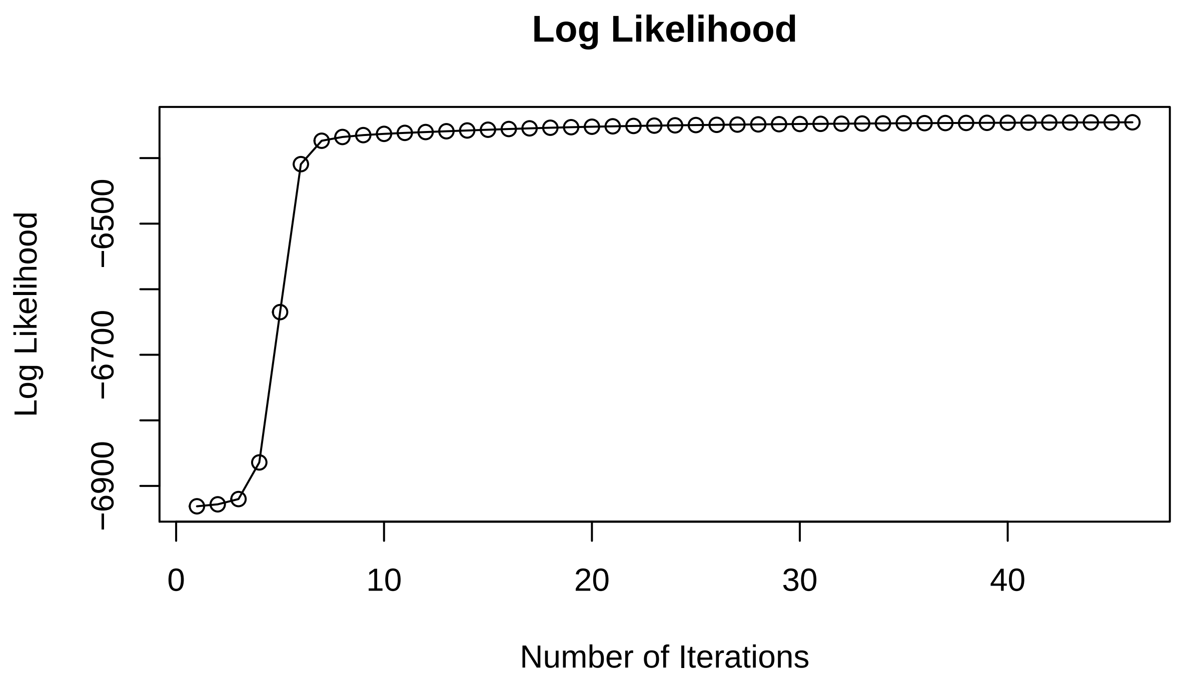
```
## iteration:  1 log likelihood:  -6930.975
## iteration:  2 log likelihood:  -6929.125
## iteration:  3 log likelihood:  -6928.562
## iteration:  4 log likelihood:  -6924.281
## iteration:  5 log likelihood:  -6893.055
## iteration:  6 log likelihood:  -6728.948
## iteration:  7 log likelihood:  -6443.28
## iteration:  8 log likelihood:  -6368.318
## iteration:  9 log likelihood:  -6363.734
## iteration:  10 log likelihood:  -6363.109
## iteration:  11 log likelihood:  -6362.947
## iteration:  12 log likelihood:  -6362.897
```



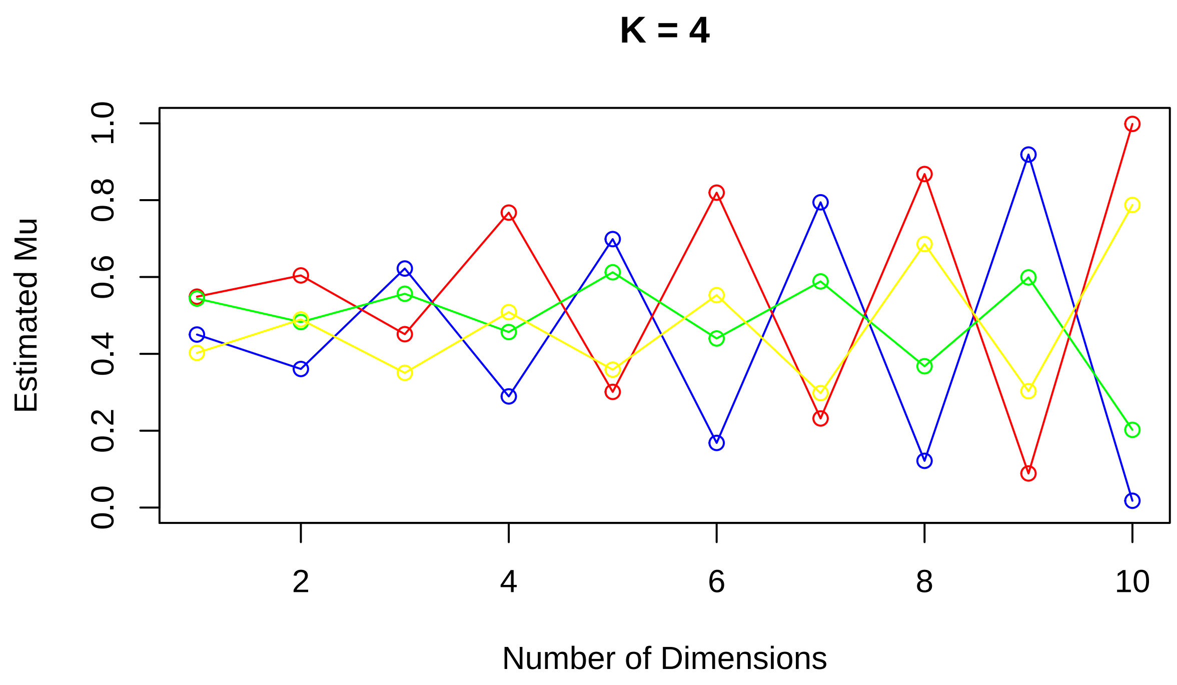


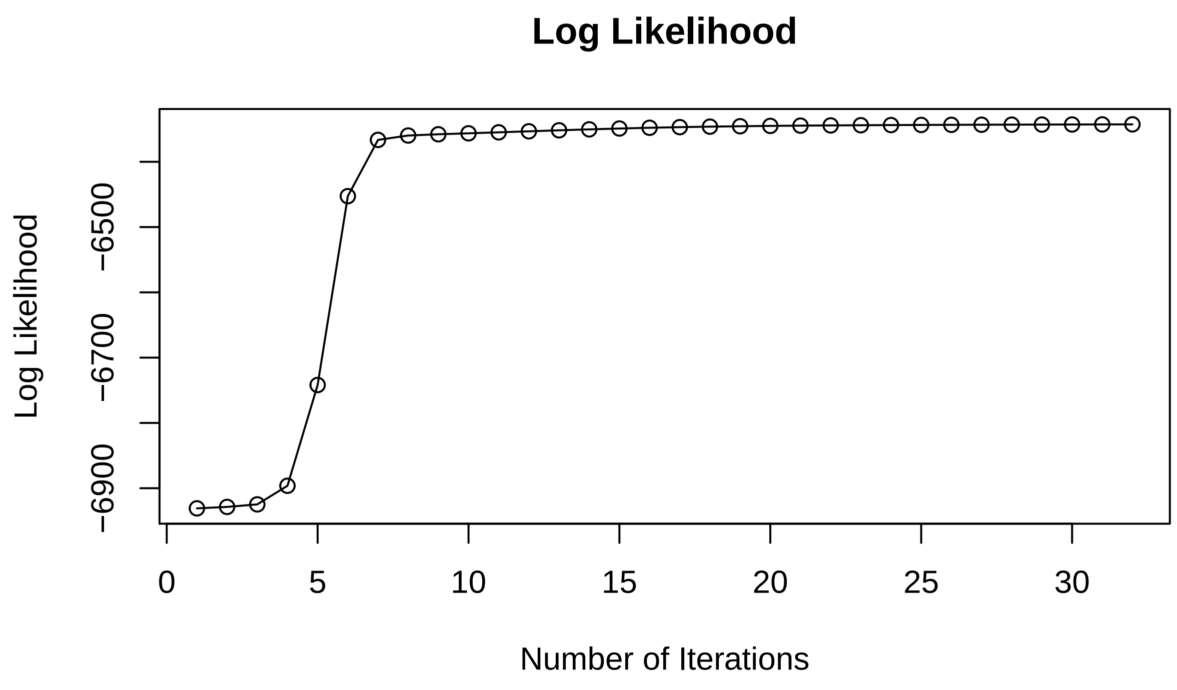
```
## iteration:  1 log likelihood:  -6931.064
## iteration:  2 log likelihood:  -6928.051
## iteration:  3 log likelihood:  -6920.026
## iteration:  4 log likelihood:  -6864.176
## iteration:  5 log likelihood:  -6634.916
## iteration:  6 log likelihood:  -6409.234
## iteration:  7 log likelihood:  -6373.593
## iteration:  8 log likelihood:  -6367.833
## iteration:  9 log likelihood:  -6364.983
## iteration:  10 log likelihood:  -6363.074
## iteration:  11 log likelihood:  -6361.594
## iteration:  12 log likelihood:  -6360.309
## iteration:  13 log likelihood:  -6359.103
## iteration:  14 log likelihood:  -6357.93
## iteration:  15 log likelihood:  -6356.786
## iteration:  16 log likelihood:  -6355.689
## iteration:  17 log likelihood:  -6354.668
## iteration:  18 log likelihood:  -6353.742
## iteration:  19 log likelihood:  -6352.92
## iteration:  20 log likelihood:  -6352.199
## iteration:  21 log likelihood:  -6351.567
## iteration:  22 log likelihood:  -6351.011
## iteration:  23 log likelihood:  -6350.515
## iteration:  24 log likelihood:  -6350.069
## iteration:  25 log likelihood:  -6349.661
## iteration:  26 log likelihood:  -6349.286
## iteration:  27 log likelihood:  -6348.938
## iteration:  28 log likelihood:  -6348.616
## iteration:  29 log likelihood:  -6348.315
## iteration:  30 log likelihood:  -6348.036
## iteration:  31 log likelihood:  -6347.776
## iteration:  32 log likelihood:  -6347.534
## iteration:  33 log likelihood:  -6347.308
## iteration:  34 log likelihood:  -6347.099
## iteration:  35 log likelihood:  -6346.904
## iteration:  36 log likelihood:  -6346.722
## iteration:  37 log likelihood:  -6346.553
## iteration:  38 log likelihood:  -6346.394
## iteration:  39 log likelihood:  -6346.246
## iteration:  40 log likelihood:  -6346.107
## iteration:  41 log likelihood:  -6345.977
## iteration:  42 log likelihood:  -6345.854
## iteration:  43 log likelihood:  -6345.739
## iteration:  44 log likelihood:  -6345.63
## iteration:  45 log likelihood:  -6345.528
## iteration:  46 log likelihood:  -6345.431
```





```
## iteration:  1 log likelihood:  -6930.838
## iteration:  2 log likelihood:  -6928.641
## iteration:  3 log likelihood:  -6924.748
## iteration:  4 log likelihood:  -6896.25
## iteration:  5 log likelihood:  -6741.896
## iteration:  6 log likelihood:  -6452.658
## iteration:  7 log likelihood:  -6366.493
## iteration:  8 log likelihood:  -6359.764
## iteration:  9 log likelihood:  -6357.876
## iteration:  10 log likelihood:  -6356.372
## iteration:  11 log likelihood:  -6354.86
## iteration:  12 log likelihood:  -6353.31
## iteration:  13 log likelihood:  -6351.776
## iteration:  14 log likelihood:  -6350.33
## iteration:  15 log likelihood:  -6349.03
## iteration:  16 log likelihood:  -6347.908
## iteration:  17 log likelihood:  -6346.968
## iteration:  18 log likelihood:  -6346.196
## iteration:  19 log likelihood:  -6345.566
## iteration:  20 log likelihood:  -6345.055
## iteration:  21 log likelihood:  -6344.637
## iteration:  22 log likelihood:  -6344.293
## iteration:  23 log likelihood:  -6344.008
## iteration:  24 log likelihood:  -6343.768
## iteration:  25 log likelihood:  -6343.563
## iteration:  26 log likelihood:  -6343.387
## iteration:  27 log likelihood:  -6343.233
## iteration:  28 log likelihood:  -6343.097
## iteration:  29 log likelihood:  -6342.975
## iteration:  30 log likelihood:  -6342.864
## iteration:  31 log likelihood:  -6342.762
## iteration:  32 log likelihood:  -6342.668
```





Analysis:

- After we created our data points using parameters mu and pi, the EM function takes only the data points and guess mu and pi(pi here is like a prior probablity based on our best guess that the responsiblities are equal but it can be set differently)
- In E-step we compute the posterior responsiblities for each observation based on Bayes Theorem, if K=2 we make the assumbtion that the points are in two clusters, anf if it is 3 the we assume 3 clustersand so on.

$$\gamma(z_{nk}) = \frac{\pi_k * p(x_n|\mu_k)}{\sum_{j=1}^{K} \pi_j * p(x_n|\mu_j)}, where\ i = 1, 2, ..., N$$

- In the M-step we updated our mu and pi based on the new responsiblities.

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n$$

$$\pi_k^{new} = \frac{N_k}{N}$$

- Log likelikelihood then is calculated (doesn't matter if it is done before M-step or after) to check if the updated values of mu and pi are converging to the true values.
- In the case of K=2 little convergence gained after the 8th iteration and stopped at 12th iteration when we started to get minimum change. the resulted values where close to the true μ_1 and μ_2, this because μ_3 value was 0.5 in the middle of the other clusters, so this third cluster splitted between the first two clusters without complications.
- When K=3 the convergence also start to blateau when we reached the 8th iteration and stopped at 46th iteration, this is because the values of first two true mus overlapping the the third mu, making it complicated for the algorithm to distinguish the third cluster from the other two (which are more distinct from each other).
- The same thing happened when K=4 also the convergence also start to blateau when we reached the 8th iteration and stopped at 32nd iteration this time, this is because the third cluster around true Mu3 got split into two cluster, one that is close to the first Mu and the other close to the second Mu.
- We can conclude from fact that the algorithm start to gain little convergence at the 8th iteration is because the first two original clusters are more distinguishable from the third cluster because their Mus overlapp the third mu, and once the estimated Mus for these two clusters are gained, the algorithm find it difficult to recognize the third cluster.

Appendix:

```r
library(mboost)
library(randomForest)

setwd('D:/Machine Learning/Workshop/Machine Learning/Block 2/Lab 1 Block 2')
RNGversion('3.5.1')
```

```
### 1. ENSEMBLE METHODS
sp      <- as.data.frame(read.csv2("Data/spambase.csv"))
sp$Spam <- as.factor(sp$Spam)

n=dim(sp)[1]
set.seed(12345)
id=sample(1:n, floor(n*2/3))
train=sp[id,]
test=sp[-id,]

k         <- seq(10,100,10)
miss_rate <- matrix(0,10,11)
colnames(miss_rate) <- c("L=0.1", "L=0.2", "L=0.3", "L=0.4", "L=0.5", "L=0.6", "L=0.7", "L=0.8", "L=0.9
rownames(miss_rate) <- c("10 Ts", "20 Ts", "30 Ts", "40 Ts", "50 Ts", "60 Ts", "70 Ts", "80 Ts", "90 Ts

# Adabtive Boosting
for(i in k){
  for(j in k){
    ada_model         <- blackboost(train$Spam ~ .,
                                    data    = train,
                                    family  = AdaExp(),
                                    control = boost_control(mstop = i, nu = j/100))

    ada_pred          <- predict(ada_model,
                                 newdata = test,
                                 type    ="class")

    ada_mat           <- table(Predected = ada_pred, Actual = test$Spam)

    miss_rate[i/10,j/10] <- round((1 - sum(diag(ada_mat))/sum(ada_mat)),4)
  }
}

Ada_Boost_Model    <- blackboost(train$Spam ~ .,
                            data    = train,
                            family  = AdaExp(),
                            control = boost_control(mstop = 100, nu = 0.5))

ada_boost_pred     <- predict(Ada_Boost_Model,
                             newdata = test,
                             type    ="class")

ada_boost_mat      <- table(Predected = ada_boost_pred, Actual = test$Spam)


# Random Forest
for(i in k){
  rf_model           <- randomForest(train$Spam ~ .,
                                     data  = train,
                                     ntree = i,
                                     importance = TRUE)

  rf_pred            <- predict(rf_model,
                              newdata = test,
                              type    ="class")

  rf_mat             <- table(Predected = rf_pred, Actual    = test$Spam)
  miss_rate[i/10,11] <- round((1 - sum(diag(rf_mat))/sum(rf_mat)),4)
}

list("Adaboost Least Error" = miss_rate[which.min(miss_rate[,5])+40],
     "Adaboost Confusion Matrix" = ada_boost_mat,
     "Random Forest Least Error" = miss_rate[which.min(miss_rate[,11])+100],
     "Random Forest Confusion Matrix" = rf_mat)

knitr::kable(miss_rate)

par(mfrow = c(2, 1))
plot(as.vector(t(miss_rate[,5])),
     main = "Adaboost Error Rate (nu set at 0.5)",
     xlab = "Number of Trees (in 10s)",
     ylab = "Error Rate",
     ylim = c(0.04,0.13),
     type = "l",
     pch  = 19,
     col  = "red")
plot(as.vector(t(miss_rate[,1:10])),
     main = "Error Rates with range of learning rates",
     xlab = "Number of Trees",
     ylab = "Error Rate",
     ylim = c(0.04,0.13),
     type = "l",
     pch  = 19,
     col  = "red")

par(mfrow = c(2, 1))
plot(miss_rate[,11],
     main = "Random Forest Error Rate",
     xlab = "Number of Trees (in 10s)",
     ylab = "Error Rate",
     ylim = c(0.04,0.13),
     type = "l",
     pch  = 19,
     col  = "black")
plot(rf_model, main = "OOB Error Rate")

knitr::kable(round(as.matrix(sort(rf_model$importance[,4], decreasing = TRUE),ncol = 4),2))

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
```

```r
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 = "Original Data",
     xlab = "Number of Dimensions",
     ylab = "True Mu")
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])
  }
}

EM <- function(c){
K=c # 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 (responsiblities)
  # Your code here
  for (n in 1:N) {
    phi = c()
    for (j in 1:K) {
      y1 = mu[j,]^x[n,]
      y2 = (1- mu[j,])^(1-x[n,])
      phi = c(phi, prod(y1,y2))
    }
    z[n,] = (pi*phi) / sum(pi*phi)
  }

  #Log likelihood computation.
  # Your code here

  likelihood <-matrix(0,1000,K)
  llik[it] <-0
  for(n in 1:N)
  {
    for (k in 1:K)
    {
      likelihood[n,k] <- pi[k]*prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))
    }
    llik[it]<- sum(log(rowSums(likelihood)))
  }

  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)
  {
    if (llik[it]-llik[it-1] < min_change)
    {
      if(K == 2)
        {
          plot(mu[1,],
               type="o",
               col="blue",
               ylim=c(0,1),
               main = "K = 2",
               xlab = "Number of Dimensions",
               ylab = "Estimated Mu")
          points(mu[2,], type="o", col="red")
        }
      else if(K==3)
        {
          plot(mu[1,],
               type="o",
               col="blue",
               ylim=c(0,1),
               main = "K = 3",
               xlab = "Number of Dimensions",
               ylab = "Estimated Mu")
          points(mu[2,], type="o", col="red")
          points(mu[3,], type="o", col="green")
        }

      else
        {
          plot(mu[1,],
               type="o",
```

```r
               col="blue",
               ylim=c(0,1),
               main = "K = 4",
               xlab = "Number of Dimensions",
               ylab = "Estimated Mu")
          points(mu[2,], type="o", col="red")
          points(mu[3,], type="o", col="green")
          points(mu[4,], type="o", col="yellow")
        }

      break()
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  mu<- (t(z) %*% x) /colSums(z)
  # N - Total no. of observations
  pi <- colSums(z)/N


}
pi
mu
plot(llik[1:it],
     type="o",
     main = "Log Likelihood",
     xlab = "Number of Iterations",
     ylab = "Log Likelihood")
}
EM(2)
EM(3)
EM(4)
```