Block2_Lab1

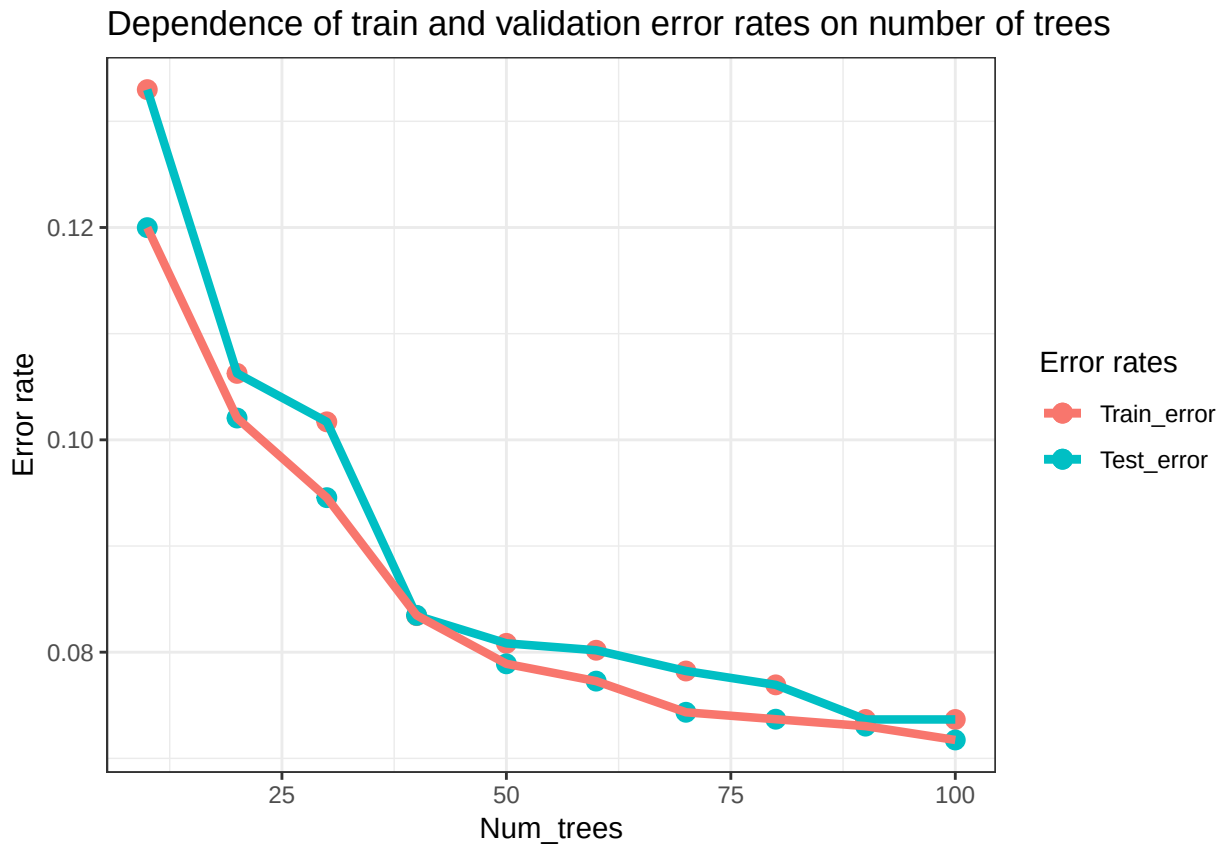
rossu809

3 December 2018

Assignment 1. ENSEMBLE METHODS

```
# data splitting
spam_data <- read.csv2("spambase.csv")
spam_data$Spam <- as.factor(spam_data$Spam)
n=dim(spam_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*2/3))
train=spam_data[id,]
test=spam_data[-id,]
```

1.1 Performance of Adaboost classification trees on spam data



Analysis:

With AdaBoost, we combine predictors by adaptively weighting the difficult-to-classify samples more heavily. So, for each iteration, classifiers concentrate on the training data values that are missed in previous iteration. From there, we make predictions based on the predictions of the various weak learners in the ensemble.

Here, feature *Spam* is the target and the model is trained with train data. Here, blackboost function is used which implements the 'classical' gradient boosting utilizing regression trees as base-learners. The arbitrary loss function (exponential) to be optimized is specified via the family. When we see the plot, the misclassification error rates for both test and train data seems to be decreasing with increase in number of trees. And also, the test error for 90 and 100 trees seems to be the same.

1.2 Performance of random forests on spam data

Analsis:

With random forests, we train however many decision trees using samples of both the data points and the features. From there, each decision should be de-correlated. We can then take an average of the prediction (regression) or majority vote (for classification).

Here, feature *Spam* is the target and the model is trained with train data. The misclassification rate of train and test data seems to be high with less number of trees (say 10). When we see the plot and dataframe (df1), its noted that the test error is same for the trees ranging from 50-90. And the error rates are randomly varying with no constant increase or decrease. But these values are quite closer to each other.

Conclusion

From both the plots, error values for random forest are quite lesser than the adaboost classifier for this specific spam dataset. So, we can infer that, Random forest performance is better than the Adaboost technique in this example.

Assignment 2. MIXTURE MODELS

In mixture models, data follows a mixture of known distributions such as where $p(x|k)$ are called mixture components and $p(k)$ are called mixing coefficients. EM algorithm for mixtures of multivariate Bernoullis distributions is implemented using the given template. The formulas we used to implement this are as follows:

E-step: Computation of the fractional component assignment

Mixture of multivariate Bernoulli distributions:

$$p(\boldsymbol{x}) = \sum_k \pi_k Bernoulli(\boldsymbol{x}|\boldsymbol{\mu}_k)$$

where we assume that

$$Bernoulli(\boldsymbol{x}|\boldsymbol{\mu}_k) = \prod_i Bernoulli(x_i|\mu_{ki}) = \prod_i \mu_{ki}^{x_i}(1-\mu_{ki})^{(1-x_i)}$$

```
  # E-step: Computation of the fractional component assignment
  # Bernoulli distribution
bernoulli_x_mu <- 0
px <- as.integer(N)
for(n in 1:1000){
  p_x <- 0
  for(k in 1:K){
    bernoulli_x_mu <- prod((mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,]))   # slide 7
    p_x <- sum(p_x , (pi[k]*bernoulli_x_mu))
  }
  px[n] <- p_x
}
```

For z matrix,

$$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}) = \frac{p(z_{nk}, \boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_k p(z_{nk}, \boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{\pi})} = \frac{\pi_k p(\boldsymbol{x}_n|\boldsymbol{\mu}_k)}{\sum_k \pi_k p(\boldsymbol{x}_n|\boldsymbol{\mu}_k)}$$

```
 # update z
 for(n in 1:N){
   for(k in 1:K){
     z[n,k]<- (pi[k]*prod((mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,])))/px[n]    # slide 9
   }
 }
```

Log likelihood computation

The log likelihood function is

$$\log p(\{\boldsymbol{x}_n, \boldsymbol{z}_n\}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_n \log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_n \log \prod_k [\pi_k \prod_i \mu_{ki}^{x_{ni}}(1-\mu_{ki})^{(1-x_{ni})}]^{z_{nk}}$$

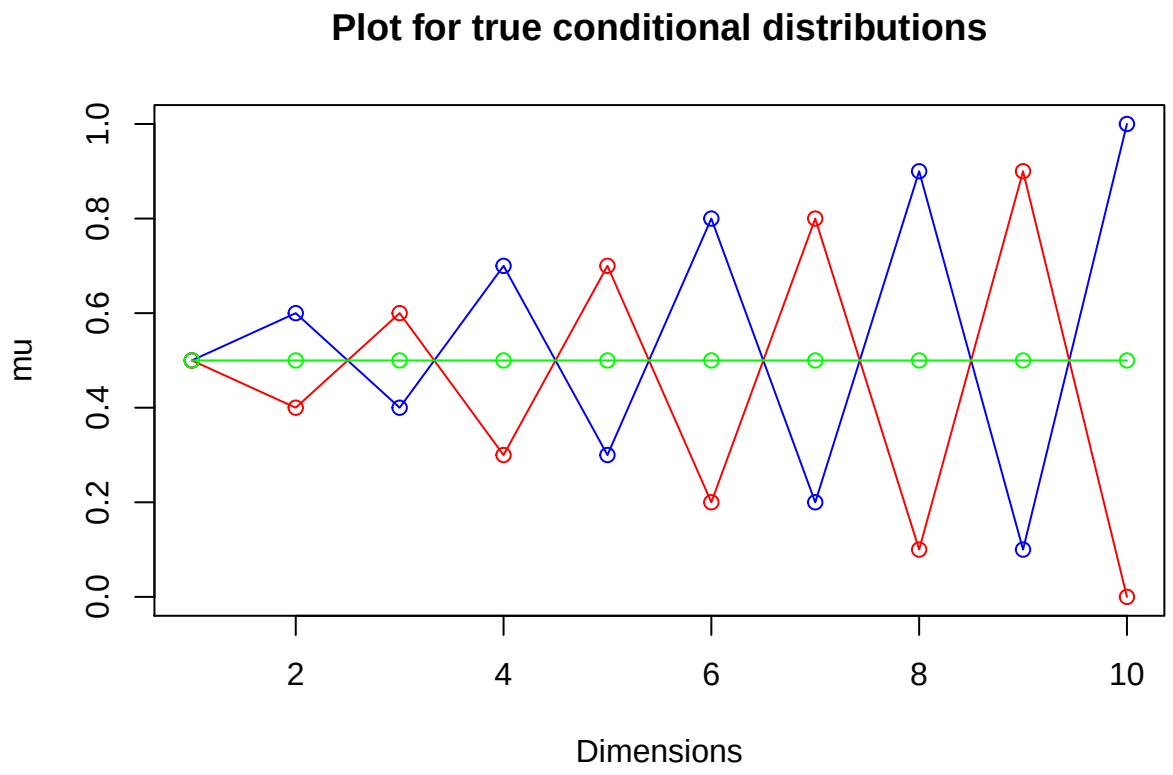
```r
#Log likelihood computation
bern <- 0
for(n in 1:N)
{
  for (k in 1:K)
  {
    bern <- prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))
    lik[n,k] <- pi[k] *  bern  # slide 8
  }
  llik[it]<- sum(log(rowSums(lik)))
}
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")

# Stop if the lok likelihood has not changed significantly

if (it > 1)
{
  if (llik[it]-llik[it-1] < min_change)
    break
}
```

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

$$\pi_k^{ML} = \frac{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{N}$$

$$\mu_{ki}^{ML} = \frac{\sum_n x_{ni} p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}$$

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



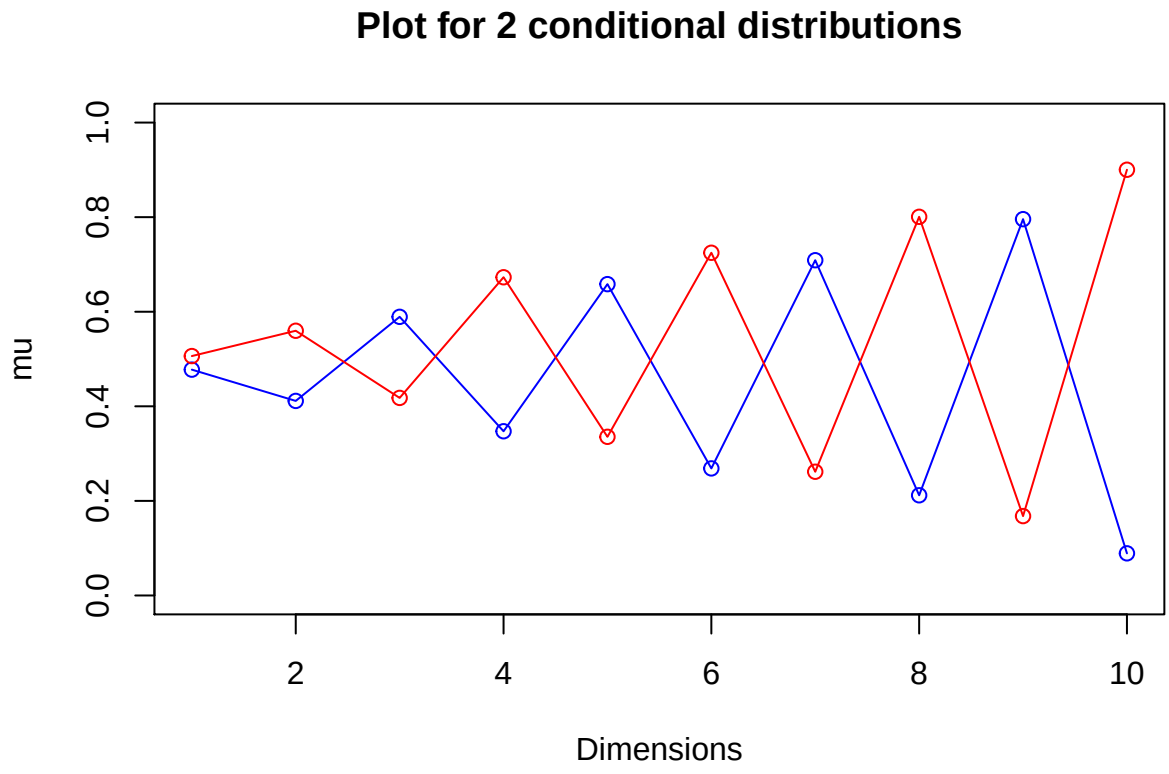
When K="2"

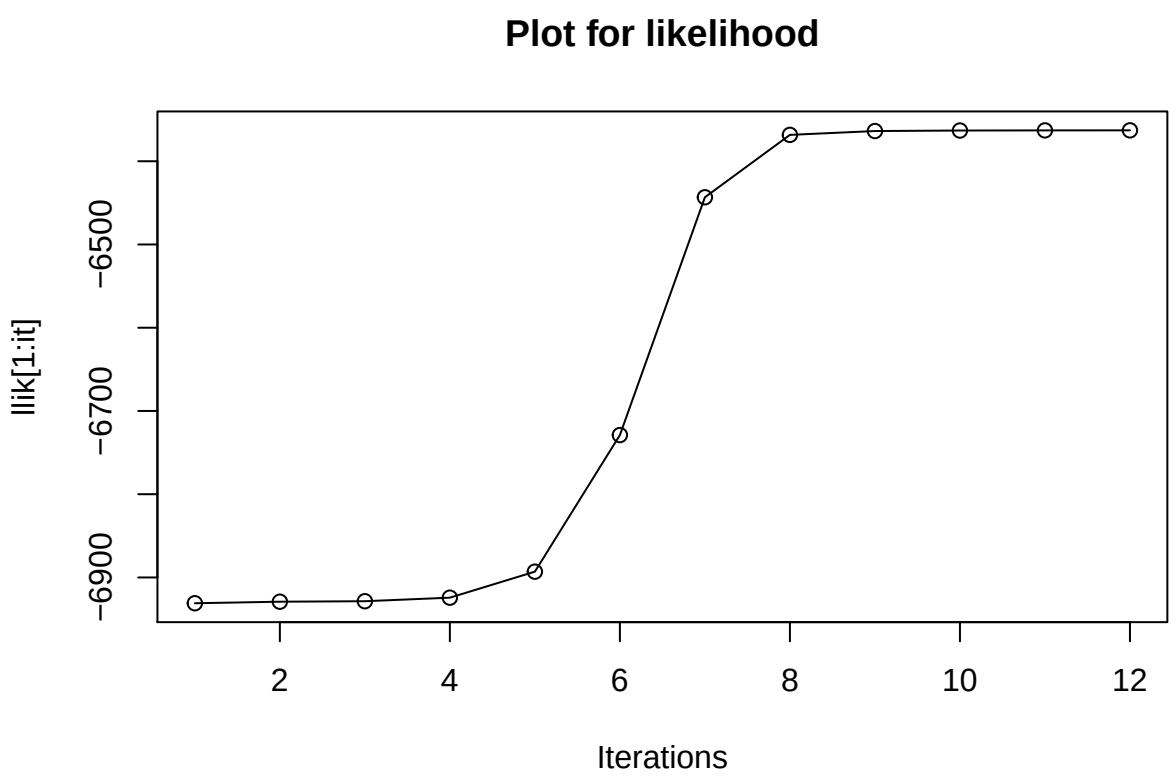
```
## iteration:  1 log likelihood:  -6930.975
## iteration:  2 log likelihood:  -6929.125
## iteration:  3 log likelihood:  -6928.562
## iteration:  4 log likelihood:  -6924.281
## iteration:  5 log likelihood:  -6893.055
## iteration:  6 log likelihood:  -6728.948
## iteration:  7 log likelihood:  -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
```

```
## Pi: 0.497125 0.502875
```

```
## Mu:
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4775488 0.4113939 0.5892308 0.3472420 0.6583712 0.2686589 0.7089490
## [2,] 0.5062860 0.5597531 0.4177551 0.6728856 0.3354854 0.7247188 0.2616231
##           [,8]      [,9]      [,10]
## [1,] 0.2118629 0.7957549 0.08905747
## [2,] 0.8007511 0.1678555 0.90027808
```





True mixing coefficients (π) : 0.333333, 0.333333

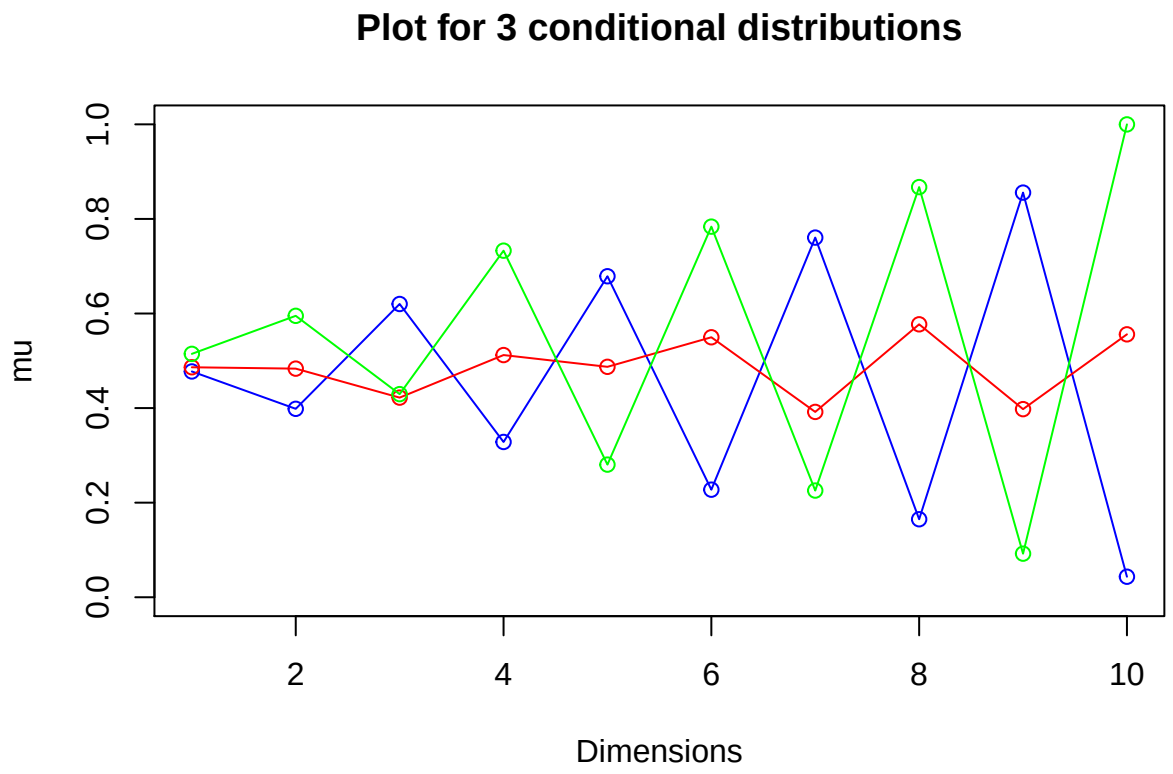
Analysis:

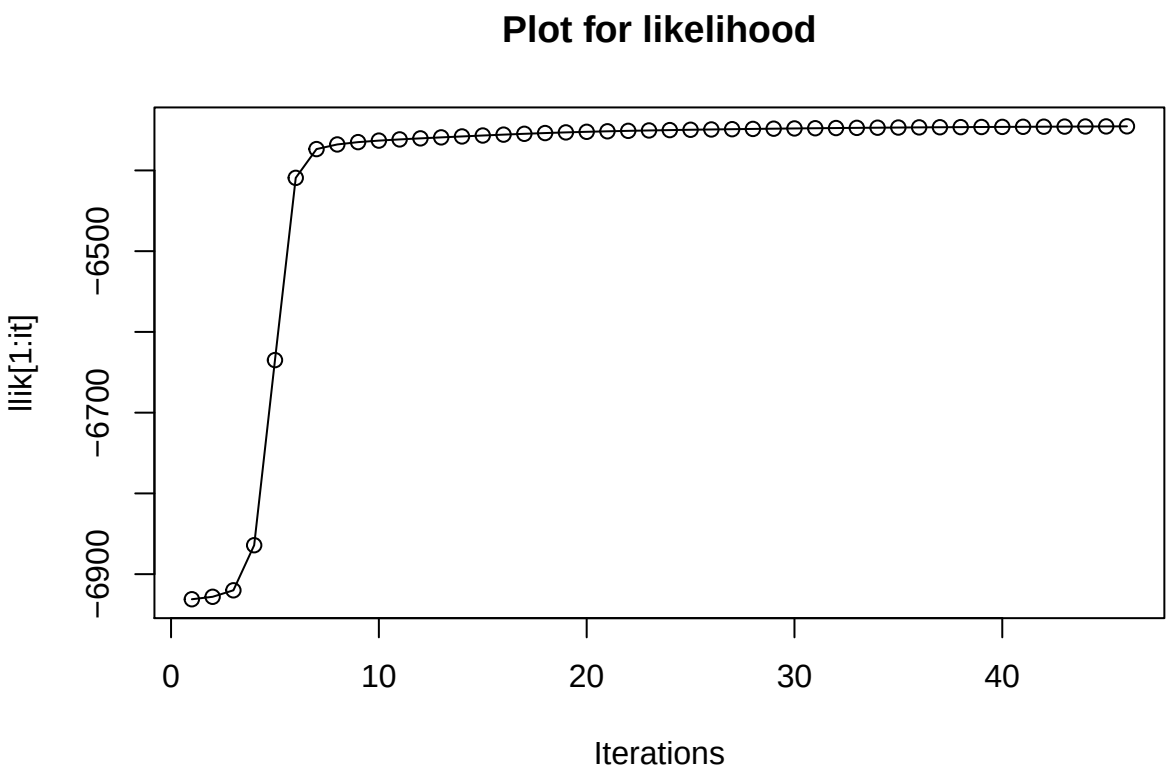
The number of guessed components considered here is 2. Two bernoulli distributions are used. The mixing coefficients are 0.497125, 0.502875 and the μ values (conditional distributions) seems to be varying for both components (but less than 1). We have 12 iterations for 2 components based on the condition stated. The 3rd distribution is merged into one of the either distribution. The algorithm estimated well, as we see both plots of true μ values and new μ values are quite similar. In likelihood plot, the values are negative and increasing consistently (in plot, from index value, 4-8). And then it remains constant. But when the components are too less, this may lead to underfitting (#slide 12).

When K="3"

```
## 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

## Pi: 0.3964172 0.278708 0.3248749

## Mu:

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4774399 0.3984407 0.6200854 0.3283412 0.6787726 0.2274644 0.7605397
## [2,] 0.4863053 0.4834283 0.4221438 0.5121400 0.4872729 0.5497432 0.3918968
## [3,] 0.5146518 0.5950474 0.4294967 0.7329049 0.2804651 0.7837214 0.2255769
##           [,8]       [,9]      [,10]
## [1,] 0.1649014 0.85568380 0.04340808
## [2,] 0.5771323 0.39774130 0.55592505
## [3,] 0.8673460 0.09215422 0.99992821
```


Plot for 3 conditional distributions
mu
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Dimensions



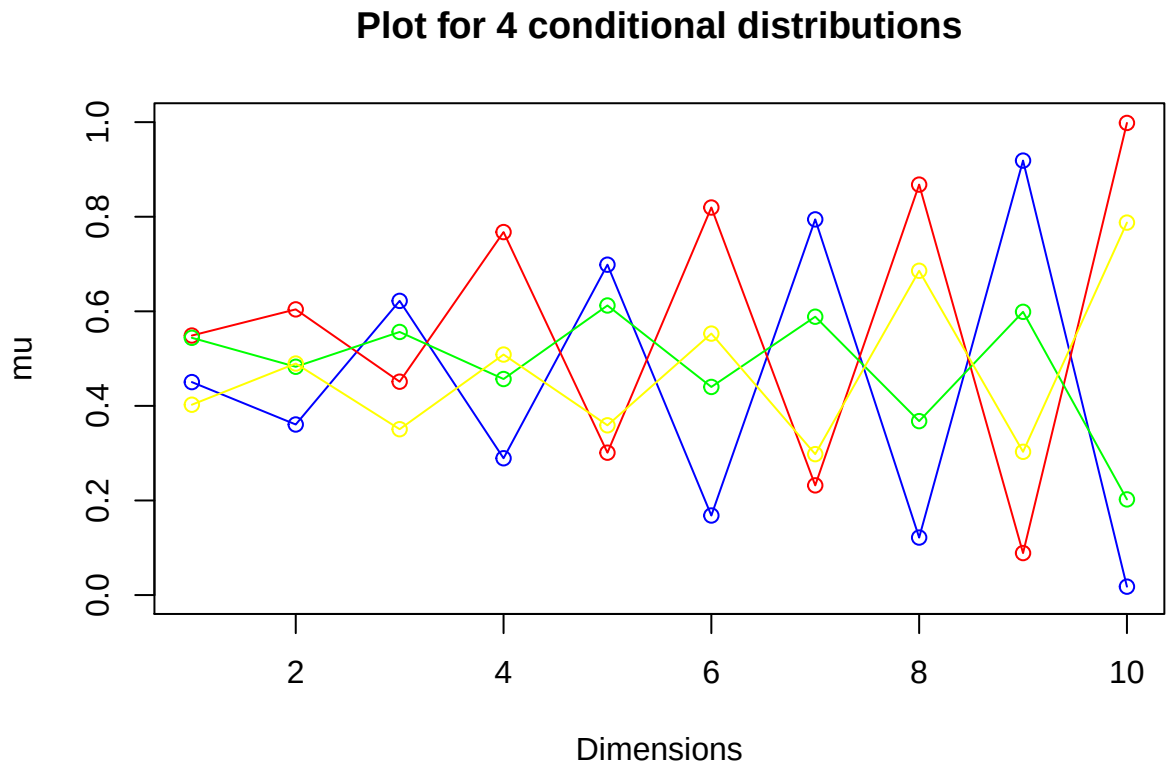
Analysis:

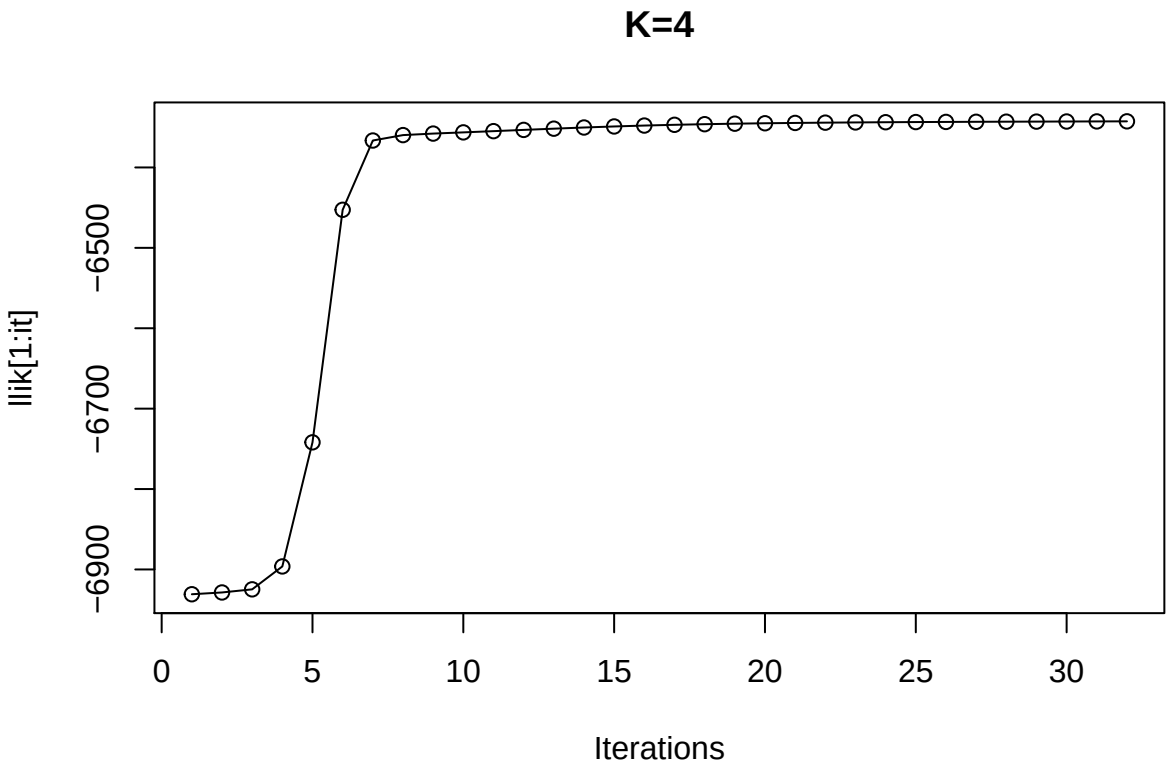
The number of guessed components considered here is 3. The mixing coefficients are 0.3964172, 0.278708, 0.3248749. We have 46 iterations for 3 components and converges. The plot with true distributions almost matches the updated one. And the likelihood remains consistent between 4 and 8 indices.

When K="4"

```
## 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

## Pi: 0.269019 0.286305 0.2457246 0.1989515

## Mu:

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4502917 0.3606587 0.6220817 0.2892407 0.6986320 0.1681768 0.7943990
## [2,] 0.5487864 0.6040921 0.4511711 0.7675478 0.3010522 0.8195305 0.2318913
## [3,] 0.5439579 0.4827437 0.5563603 0.4568300 0.6123061 0.4400351 0.5885625
## [4,] 0.4025047 0.4895637 0.3506597 0.5085745 0.3588983 0.5528693 0.2979403
##           [,8]       [,9]      [,10]
## [1,] 0.1215732 0.91870837 0.01774129
## [2,] 0.8679302 0.08877946 0.99833984
## [3,] 0.3677877 0.59890299 0.20227253
## [4,] 0.6857315 0.30292227 0.78760041
```





Analysis:

The number of guessed components considered here is 4. The mixing coefficients are 0.3964172, 0.278708, 0.3248749. We have 32 iterations for 4 components. The likelihood remains consistent between 4 and 8 indices but the plot with true μ doesn't match well with the updated one. The two of the distributions are estimated well and looks similar to the true distribution but the 3rd and 4th distributions would combinely (i.e taking average) provide good estimate of the true third distribution. Here the model becomes little complex and tend to be overfitting.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
# data splitting
spam_data <- read.csv2("spambase.csv")
spam_data$Spam <- as.factor(spam_data$Spam)
n=dim(spam_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*2/3))
train=spam_data[id,]
test=spam_data[-id,]
# predict function
predictionn <- function(fit,data){
  pred <- predict(fit,data,type = "class")
  mat <- table(data$Spam,pred)
  err <- 1 - sum(diag(mat))/sum(mat)
```

```
  return(err)
}

# plot function
library(ggplot2)
err_plot <- function(data){
  ggplot(data)+
  geom_point(aes(test_error,x=num_trees,color="blue"),size=3)+
  geom_point(aes(train_error,x=num_trees,color="green"),size=3)+
  geom_line(aes(test_error,x=num_trees,color="green"),size=1.5)+
  geom_line(aes(train_error,x=num_trees,color="blue"),size=1.5)+
  ggtitle("Dependence of train and validation error rates on number of trees")+
    xlab("Num_trees")+ ylab("Error rate")+
    scale_color_discrete(name = "Error rates", labels = c("Train_error", "Test_error"))+theme_bw()
}
#fit the model
library(mboost)

df <- data.frame()
for (i in seq(10,100,10)) {
  set.seed(12345)
  spam_ada_fit <- blackboost(Spam~., data=train, family = AdaExp(),control = boost_control(mstop = i))
  ran_test_err_a <- predictionn(spam_ada_fit,test)
  paste("Test error rate:",ran_test_err_a)
  ran_train_err_a <- predictionn(spam_ada_fit,train)
  paste("Train error rate:",ran_train_err_a)
  df <- rbind(df, data.frame(test_error=ran_test_err_a,train_error=ran_train_err_a,num_trees=i))
}
err_plot(df)
library(randomForest)

df1 <- data.frame()
for (i in seq(10,100,10)) {
  set.seed(12345)
  spam_random_fit <- randomForest(Spam~., data=train, ntree=i)
  ran_test_err <- predictionn(spam_random_fit,test)
  ran_train_err <- predictionn(spam_random_fit,train)
  df1 <- rbind(df1, data.frame(test_error=ran_test_err,train_error=ran_train_err,num_trees=i))
}
err_plot(df1)
knitr::include_graphics("1.PNG")
knitr::include_graphics("2.PNG")
knitr::include_graphics("e_step.PNG")
knitr::include_graphics("3.PNG")
knitr::include_graphics("z.PNG")
knitr::include_graphics("4.PNG")
knitr::include_graphics("llik.PNG")
knitr::include_graphics("5.PNG")
knitr::include_graphics("m_step.PNG")
##############################################################################
#Mixture models

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="Plot for true conditional distributions",xlab ="Dimensions",ylab = "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])
  }}

#############################################################################################

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(j in 1:K) {
  mu[j,] <- runif(D,0.49,0.51)
}
#pi
#mu
lik <-matrix(0,nrow =1000,ncol = K)

for(it in 1:max_it)
{
  # E-step: Computation of the fractional component assignment
  # Bernoulli distribution
bernoulli_x_mu <- 0
px <- as.integer(N)
for(n in 1:1000){
  p_x <- 0
  for(k in 1:K){
    bernoulli_x_mu <- prod((mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,]))   # slide 7
    p_x <- sum(p_x , (pi[k]*bernoulli_x_mu))
  }
  px[n] <- p_x
}
```

```
# update z
for(n in 1:N){
  for(k in 1:K){
    z[n,k]<- (pi[k]*prod((mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,])))/px[n]    # slide 9
  }
}

 #Log likelihood computation
 bern <- 0
 for(n in 1:N)
 {
   for (k in 1:K)
   {
     bern <- prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))
     lik[n,k] <- pi[k] *  bern  # slide 8
   }
   llik[it]<- sum(log(rowSums(lik)))
 }
 cat("iteration: ", it, "log likelihood: ", llik[it], "\n")

 # Stop if the lok likelihood has not changed significantly

 if (it > 1)
 {
   if (llik[it]-llik[it-1] < min_change)
     break
 }

 #M-step: ML parameter estimation from the data and fractional component assignments
 pi <- colSums(z)/N
 mu<- (t(z) %*% x) /colSums(z)
}
cat("Pi:",pi, "\n")
cat("Mu:", "\n")
print(mu)
plot(mu[1,], type="o", col="blue", ylim=c(0,1),
     main="Plot for 2 conditional distributions",xlab ="Dimensions",ylab = "mu")
points(mu[2,], type="o", col="red")
plot(llik[1:it], type="o",xlab = "Iterations",main = "Plot for likelihood")

#############################################################################################

### K=3

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(j in 1:K) {
```

```
  mu[j,] <- runif(D,0.49,0.51)
}
#pi
#mu
lik <-matrix(0,nrow =1000,ncol = K)

for(it in 1:max_it)
{
  # E-step: Computation of the fractional component assignment
  # Bernoulli distribution
  bernoulli_x_mu <- 0
px <- as.integer(N)
for(n in 1:1000){
  p_x <- 0
  for(k in 1:K){
    bernoulli_x_mu <- prod((mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,]))   # slide 7
    p_x <- sum(p_x , (pi[k]*bernoulli_x_mu))
  }
  px[n] <- p_x
}
for(n in 1:N){
  for(k in 1:K){
    z[n,k]<- (pi[k]*prod((mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,])))/px[n]    # slide 9
  }
}


  #Log likelihood computation
  bern <- 0
  for(n in 1:N)
  {
    for (k in 1:K)
    {
      bern <- prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))
      lik[n,k] <- pi[k] *  bern  # slide 8
    }
    llik[it]<- sum(log(rowSums(lik)))
  }
  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")

  # Stop if the lok likelihood has not changed significantly

  if (it > 1)
  {
    if (llik[it]-llik[it-1] < min_change)
      break
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  pi <- colSums(z)/N
  mu<- (t(z) %*% x) /colSums(z)
}
cat("Pi:",pi, "\n")
```

```
cat("Mu:", "\n")
print(mu)
plot(mu[1,], type="o", col="blue", ylim=c(0,1),
     main="Plot for 3 conditional distributions",xlab ="Dimensions",ylab = "mu")
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
plot(llik[1:it], type="o",xlab = "Iterations",main = "Plot for likelihood")

#############################################################################################

### K=4

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(j in 1:K) {
  mu[j,] <- runif(D,0.49,0.51)
}
#pi
#mu
lik <-matrix(0,nrow =1000,ncol = K)

for(it in 1:max_it)
{
  # E-step: Computation of the fractional component assignment
  # Bernoulli distribution
bernoulli_x_mu <- 0
px <- as.integer(N)
for(n in 1:1000){
  p_x <- 0
  for(k in 1:K){
    bernoulli_x_mu <- prod((mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,]))   # slide 7
    p_x <- sum(p_x , (pi[k]*bernoulli_x_mu))
  }
  px[n] <- p_x
}
for(n in 1:N){
  for(k in 1:K){
    z[n,k]<- (pi[k]*prod((mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,])))/px[n]    # slide 9
  }
}

  #Log likelihood computation
  bern <- 0
  for(n in 1:N)
  {
    for (k in 1:K)
    {
```

```
      bern <- prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))
      lik[n,k] <- pi[k] *  bern  # slide 8
    }
    llik[it]<- sum(log(rowSums(lik)))
  }
  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")

  # Stop if the lok likelihood has not changed significantly

  if (it > 1)
  {
    if (llik[it]-llik[it-1] < min_change)
      break
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  pi <- colSums(z)/N
  mu<- (t(z) %*% x) /colSums(z)
}
cat("Pi:",pi, "\n")
cat("Mu:", "\n")
print(mu)
plot(mu[1,], type="o", col="blue", ylim=c(0,1),
      main="Plot for 4 conditional distributions",xlab ="Dimensions",ylab = "mu")
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",xlab = "Iterations",main = "K=4")
```