Ensemble Mthods & Mixture Models

Rakhshanda Jabeen

2019-11-24

Summary

In this Project, spam data is used for the comparison between AdaBoost and Random forest as well as a comparison between the two mwthods that which method is more useful in this particular scenario. We will also calculate the EM algorithm for different components.

1. ENSEMBLE METHODS

Ensemble methods are meta algorithms that combine several machine learning techniques in one predictive model to decrease variance, bias and improve prediction. The ensemble method works better with the trees.

Ensemble method can be categorized into two groups:

- Sequential method where the base learner is generated sequentially for example **(AdaBoost)**
- Parallel method where the base learner is generated in parallel for example **(Random Forest)**

In this task we evaluate the performance of Adaboost and Random Forest on the spam data which has 2740 e-mails containing the frequency of various words and characters. It is manually classified whether the e-mail is spam or not.Then, data is randomly shuffled and divided into two parts as train and test data.

1.1. ADAboost Classification

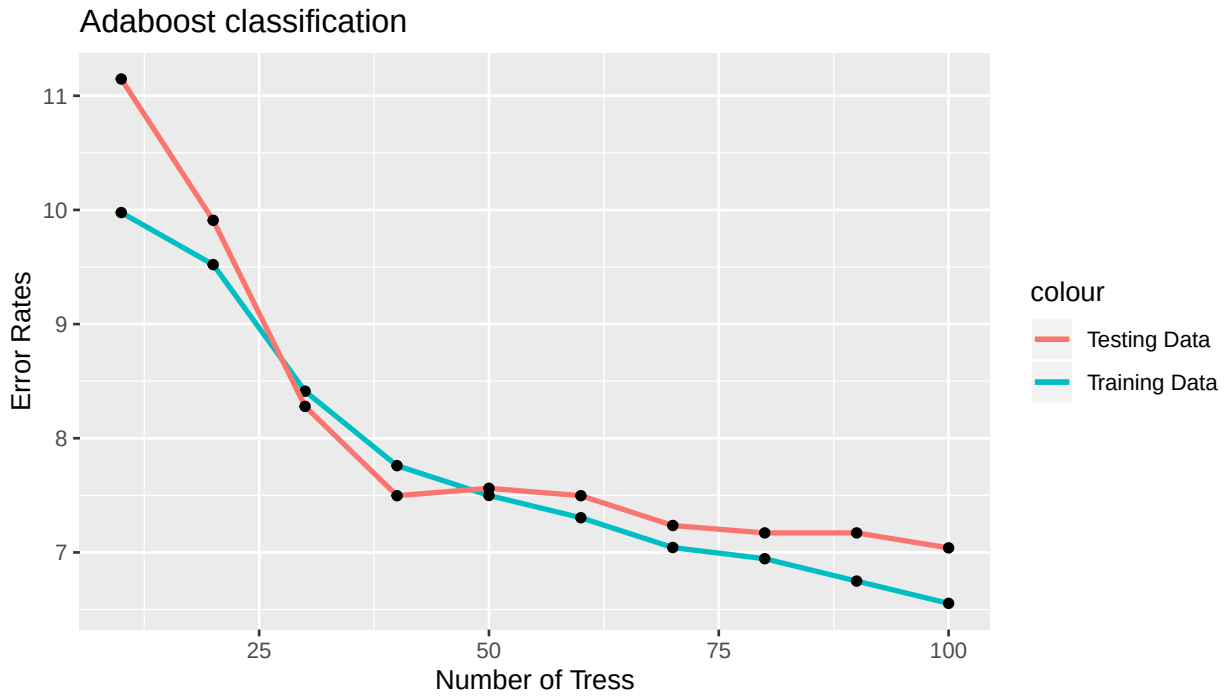
Using the Adaboost classification, we created a model on training data to describe the response variable Spam using all the other variables.

Missclassification rates

The following table and graph represents misclassification rates of the data when ADAbosst classification model is applied to train and test data.

Table 1: Error Rates

No.of.Trees	Training.data	Testing.data
10	9.977	11.147
20	9.521	9.909
30	8.412	8.279
40	7.760	7.497
50	7.499	7.562
60	7.304	7.497
70	7.043	7.236
80	6.945	7.171
90	6.749	7.171
100	6.554	7.040



Adaboost convert weak learner to strong learner. The main principle of AdaBoost is to fit weak learner sequentially and by putting more weight on misclassified by earlier rounds. From the above graph, we can easily see that the error rate keep on decreasing as we increase the interaction because Adaboost keeps on adding more weight on the misclassified instance.

1.2. Random Forest Classification

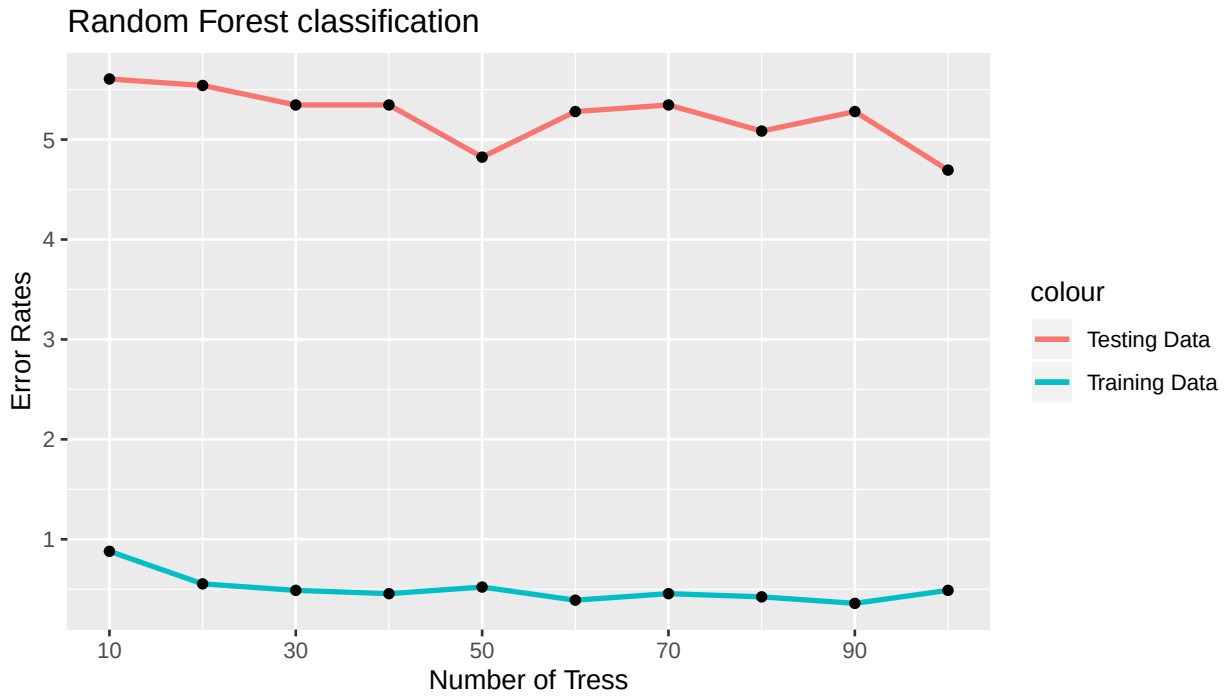
Using the Random Forest classification, we created a model on training data to describe the response variable Spam using all the other variables.

Missclassification rates

The following table and graph represents misclassification rates of the data when ADAbosst classification model is applied to train and test data.

Table 2: Error Rates

No.of.Trees	Trainind.Data	Testing.Data
10	0.880	5.606
20	0.554	5.541
30	0.489	5.346
40	0.456	5.346
50	0.522	4.824
60	0.391	5.280
70	0.456	5.346
80	0.424	5.085
90	0.359	5.280
100	0.489	4.694



Random Forest consists of a plethora of individual trees, it then assigns the votes to each tree and expectorates a prediction class. Prediction class with high votes becomes our desired prediction class. Correlation between the trees matters. Uncorrelated models can predict accurate values as compare to individual predictions. Predictions by the individual trees should have the lowest correlation if it has a high correlation then errorm increases. As we can see in the above graph, increasing the trees in each iteration error rate is almost the same which means every tree has the same correlation after a specific point. Which means with random forest data can never be overfitted.

Conclusion

Adaboost converts the data into stem and two leaves, then it assigns weights to it while random forest works by making multiple trees and then take their average. Random Forest learn from various overgrow tree and the final decision is made based on the majority. This process works best on model which have high variance and low bias because we can minimize the variance by taking the average of all the overfit tree.

On the other hand, Boosting manipulate the training set by each iteration. Form this method works well if we worry about model high business. From the above graph, we can easily see that Random forest is better than Adaboost, It means the model has high variance and low bias.

2. MIXTURE MODELS

The mixture model comes under unsupervised learning and it is the combination of different probabilistic models. Below given is the mixture model's generalized formula.

$$p(x) = \sum_{k=1}^{K} p(k)p(x|k)p(k) = \pi_k$$

Bernoulli Distribution

$$p(x) = \sum_{k} Bernoulli(x|\mu_k)$$

Where we assuem that

$$Bernoullu(x|\mu_k) = \prod_{i} \mu_{k_i}^{x_i}(1-\mu_{k_i})^{1-x_i}$$

E-Step

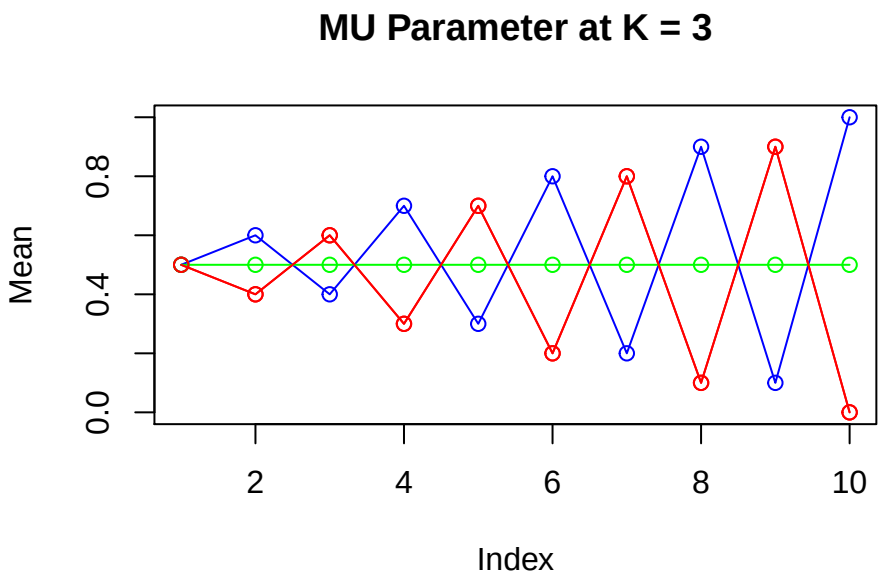
$$p(z_{n_k}|x_n, \mu, \pi) = z_n = \frac{\pi_k p(x_n|\mu_k)}{\sum_k \pi_k p(x_n|\mu_k)}$$

M-Step

$$\pi_k^{ML} = \frac{\sum_n p(z_{n_k}|x_n, \mu, \pi)}{N} \pi_k^{ML} = \frac{\sum_n x_{n_i} p(z_{n_k}|x_n, \mu, \pi)}{\sum_n p(z_{n_k}|x_n, \mu, \pi)}$$

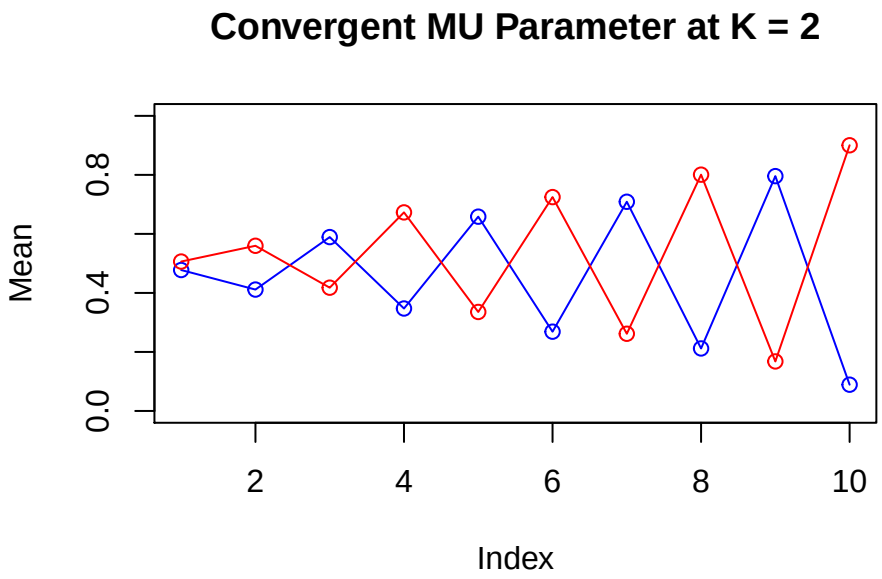
By combining the latent variables, we calculate the maximum likelihood. We compare the current maximum likelihood with the previous maximum likelihood until the difference between both is not less than or equal to our defined minimum change. When it satisfies this condition, then it converges to its local maximum.

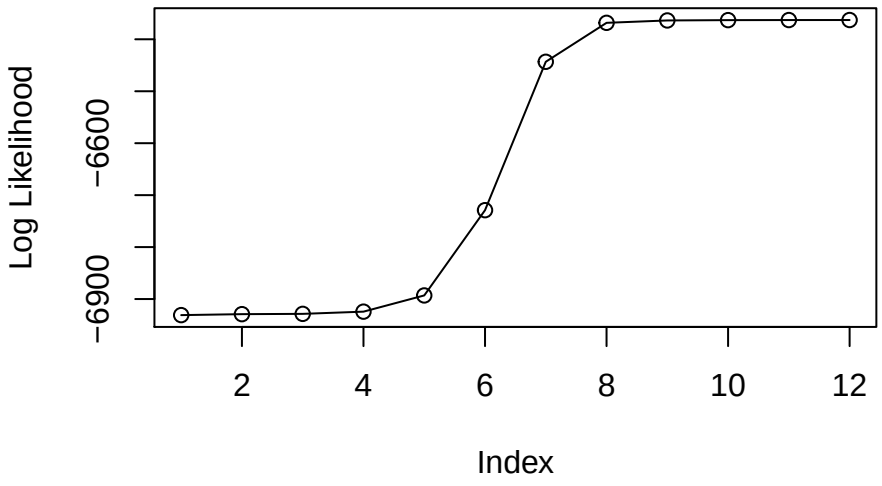
Here we are attempting to model the mixed dataset with 3 components.



Mixing Component, K = 2

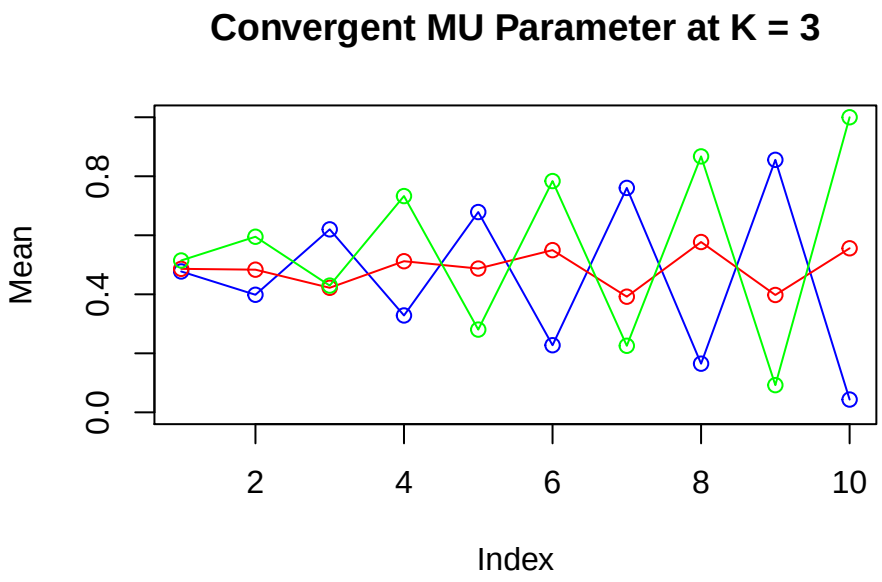
```
## Converge at iteration: 11 with log likelihood: -6362.897
```

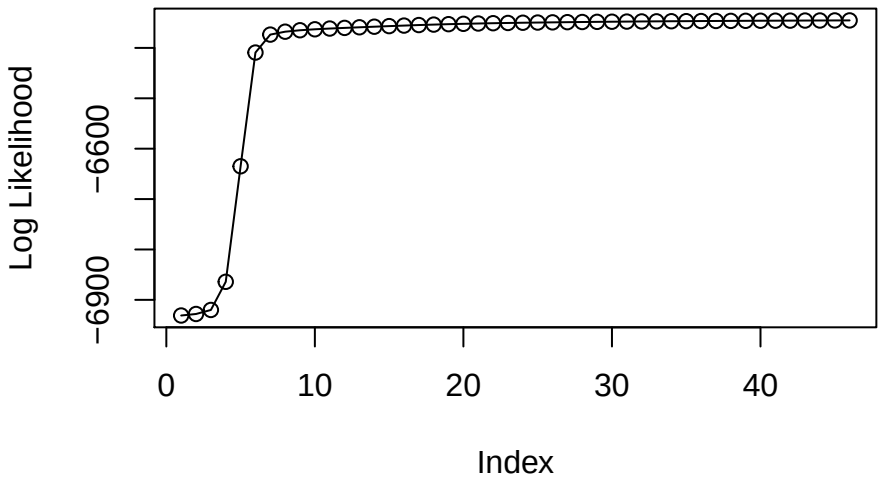




Here we are attempting to model the mixed dataset with a very few components, the above plot represents the change of the log-likelihood function of the mixture model with respect to the number of iterations.

```
## Converge at iteration: 45 with log likelihood: -6345.431
```

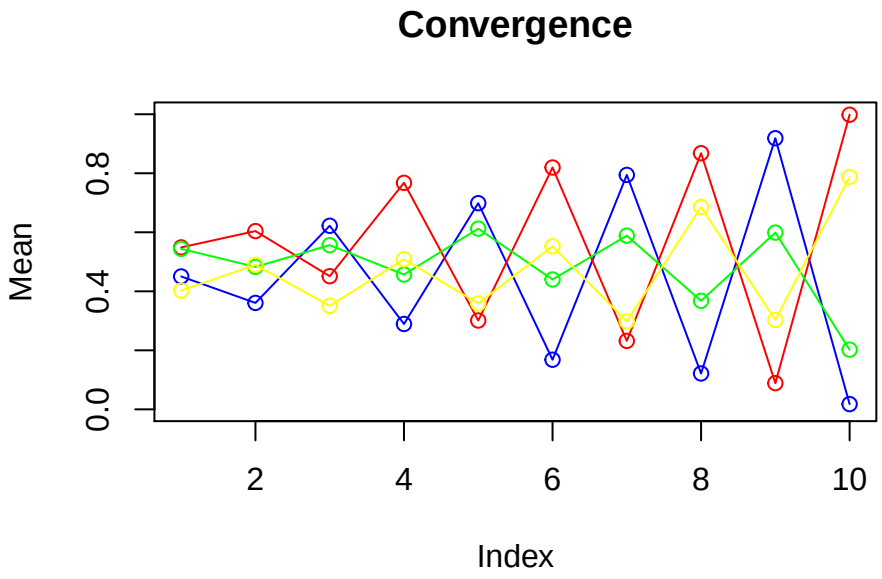


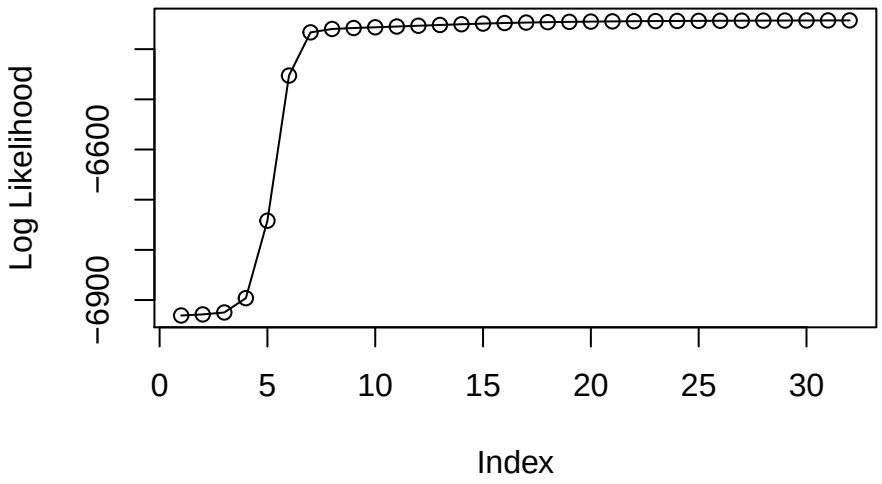


The above plot represents the change of the log-likelihood function of the mixture model with respect to the number of iterations.

K = 4

```
## Converge at iteration: 31 with log likelihood: -6342.668
```





Here we are attempting to model the mixed dataset with 4 components, the above plot represents the change of the log-likelihood function of the mixture model with respect to the number of iterations.

Conclusion

We can clearly see that as number of components are incresing EM algorithm performs too many iterations to maximize the log liklihood.

Appendix

```
knitr::opts_chunk$set(echo = FALSE)
setwd("E:/Machine Learning/Block 2/project 1")
library(knitr)
library(mboost)
library(ggplot2)
library(randomForest)
library(readxl)
library(kableExtra)
sp = read.csv2(file = "spambase.csv", header = TRUE, sep = ";")
sp$Spam = as.factor(sp$Spam)
n=dim(sp)[1]
RNGkind(sample.kind = "Rounding")
set.seed(12345)
id=sample(1:n, floor(n*2/3))
train=sp[id,]
test=sp[-id,]
 n_tree = seq(10,100, by = 10)
adaboost = function(data,test)
{
  count = 1
  error_rates = 0
  for (i in n_tree)
  {
    model = blackboost(formula = Spam ~ ., data = data,
                       family = Binomial(type = c("adaboost")),
                       control = boost_control(mstop = i))
    fit = predict(model, test, type = 'class')
    conf_mat = table(test$Spam, fit)
    n = nrow(test)
    error_rates[count] = (1-sum(diag(conf_mat))/n)*100
    count = count + 1
  }
  return(error_rates)
}
tr_ada = round(adaboost(data = train,train),3)
tes_ada = round(adaboost(data = train,test),3)
df = data.frame("No of Trees"= n_tree,
                "Training data" = tr_ada,
                "Testing data" = tes_ada)
kable(df, "latex", caption = "Error Rates", booktabs = T) %>%
kable_styling(latex_options = "hold_position")
ggplot(data = df)+
geom_line(aes(x = n_tree , y = tr_ada, col = "Training Data"), size = 1)+
  geom_point( aes(x = n_tree , y = tr_ada))+
  geom_line(aes(x = n_tree , y = tes_ada, col = "Testing Data"), size = 1)+
  geom_point( aes(x = n_tree , y = tes_ada))+
  xlab("Number of Tress") + ylab("Error Rates")+ ggtitle("Adaboost classification")
random_forest = function(data)
{
  count = 1
  error_rates = c()
```

```
  for (j in n_tree)
  {
    model = randomForest(Spam ~., data = train, ntree = j)
    fit = predict(model, data, type = 'class')
    conf_mat = table(data$Spam, fit)
    n = nrow(data)
    error_rates[count] = (1-sum(diag(conf_mat))/n)*100
    count = count+1
  }
  error_rates
}
tr_rf = round(random_forest(train), 3)
tes_rf = round(random_forest(test), 3)
df1 = data.frame("No of Trees" = n_tree, "Trainind Data" = tr_rf,
                 "Testing Data" = tes_rf)
kable(df1, "latex", caption = "Error Rates", booktabs = T) %>%
kable_styling(latex_options = "hold_position")
ggplot(data = df1)+
  geom_line(aes(x = n_tree , y = tr_rf, col = "Training Data"), size = 1)+
  geom_point( aes(x = n_tree , y = tr_rf))+
  geom_line(aes(x = n_tree , y = tes_rf, col = "Testing Data"),size=1)+
  geom_point( aes(x = n_tree , y = tes_rf))+
  scale_x_continuous(breaks = seq(10, 100,20))+
  xlab("Number of Tress") + ylab("Error Rates")+
  ggtitle("Random Forest classification")
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 distributionst
mix_model = function(K)
{
  # K is  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
  oldlikeli = 0
  count = 0
  for(it in 1:max_it) {
```

```
    ll = vector(length =N)
    for(n in 1:N)
    {
      sum = 0
      for(comp in 1:K)
      {
        z[n,comp] = prod(mu[comp,]^x[n,]*(1-mu[comp,])^(1-x[n,])) * pi[comp]
        sum = sum + z[n,comp]
      }
      norms = z[n,]
      z[n,] = z[n,]/sum
      ll[n] = log(sum(norms))
    }

    llik[it] = sum(ll)

    #Log likelihood computation.
    flush.console()
    # Stop if the lok likelihood has not changed significantly
    # Your code here

    if(abs(oldlikeli - llik[it]) < min_change)
    {
      cat("Converge at iteration:", count, "with log likelihood:", llik[it])
      break
    }
    else
    {
      count = count+1
      oldlikeli = llik[it]
    }
    #M-step: ML parameter estimation from the data and fractional component assignments
    # Your code here
    nomi = colSums(z)
    pi = nomi/N

    for (comp in 1:K) {
      mu[comp,] = colSums(x*z[,comp])/nomi[comp]
    }
  }
   log_lik = llik[1:it]
  lst = list(mu, log_lik)
  return(lst)
}
true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=10) # 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),
     ylab = "Mean", main = "MU Parameter at K = 3")
points(true_mu[2,], type="o", col="red")
```

```
points(true_mu[3,], type="o", col="green")
points(true_mu[2,], type="o", col="red")
# 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])
    }
  }
m = mix_model(2)
mu = m[[1]]
plot(mu[1,], type="o", col="blue", ylim=c(0,1),
     ylab = "Mean", main = "Convergent MU Parameter at K = 2")
points(mu[2,], type="o", col="red")

plot(m[[2]], type="o", ylab = "Log Likelihood")
t = mix_model(3)
mu1 = t[[1]]
plot(mu1[1,], type="o", col="blue", ylim=c(0,1),
     ylab = "Mean", main = "Convergent MU Parameter at K = 3")
points(mu1[2,], type="o", col="red")
points(mu1[3,], type="o", col="green")
plot(t[[2]], type="o", ylab = "Log Likelihood")

s = mix_model(K = 4)
mu2 = s[[1]]
plot(mu2[1,], type="o", col="blue", ylim=c(0,1),
     ylab = "Mean", main = "Convergence")
points(mu2[2,], type="o", col="red")
points(mu2[3,], type="o", col="green")
points(mu2[4,], type="o", col="yellow")
plot(s[[2]], type="o", ylab = "Log Likelihood")
```