Computer Lab Block 2

Group A7

2022-12-09

Contents

Group members:Hamza (hammu144), Dinesh (dinsu875) and Umamaheswarababu (umama339)

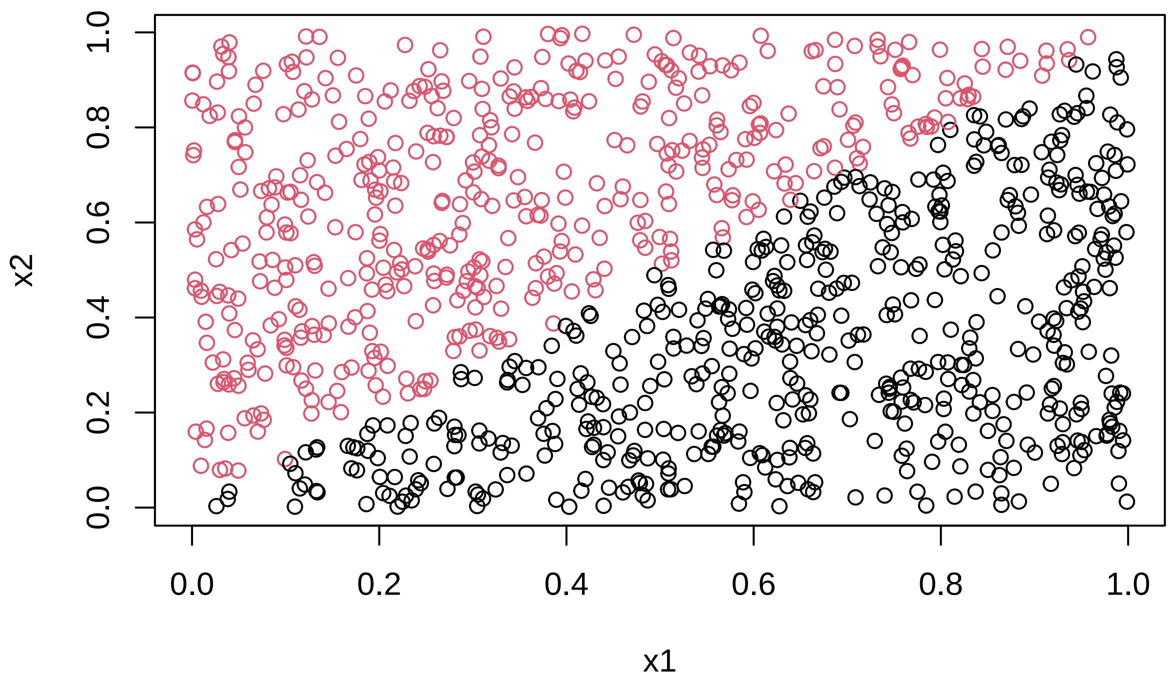
Statement of Contribution : Assignment 1 was mostly done by Umamaheswarababu, Assignment 2 was mostly done by Dinesh and Hamza has contributed to both the assignments

1 Assignment-1 : ENSEMBLE METHODS

Creating training data and test data

```r
#training data
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)
```

```r
#test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y1<-as.numeric(x1<x2)
telabels<-as.factor(y1)
y<-telabels
tedata<-data.frame(x1,x2,y)
plot(x1,x2,col=(y1+1))
```



Q1. Repeat the procedure above for 1000 training datasets of size 100 and report the mean and variance of the misclassification errors. In other words, create 1000 training datasets of size 100, learn a random forest from each dataset, and compute the misclassification error in the same test dataset of size 1000. Report results for when the random forest has 1, 10 and 100 trees.

```r
#Creating 1000 training datasets and storing them in a list
train_sets_1<-list()
for (i in 1:1000){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<x2)
  trlabels<-as.factor(y)
  y<-trlabels
  trdata<-data.frame(x1,x2,y)
  train_sets_1[[i]]<-trdata
}

#test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y1<-as.numeric(x1<x2)
telabels<-as.factor(y1)
y<-telabels
```

```r
tedata<-data.frame(x1,x2,y)


#A function to calculate misclassification error
missclass<- function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

#importing "randomForest" package
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.
```

```r
#three vectors to store misclassification errors for different trees
misclass_1<-c()
misclass_10<-c()
misclass_100<-c()

#learning a random forest with 1, 10 and 100 trees
for (i in train_sets_1){
  random_forest_1<-randomForest(y~., data=i, ntree=1, nodesize=25, keep.forest=TRUE)
  random_forest_10<-randomForest(y~., data=i, ntree=10, nodesize=25, keep.forest=TRUE)
  random_forest_100<-randomForest(y~., data=i, ntree=100, nodesize=25, keep.forest=TRUE)

  #computing misclassification errors with test data
  prediction_1<-predict(random_forest_1, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_1)
  misclass_1<-c(misclass_1,misclass) #misclass errors for tree_1

  prediction_10<-predict(random_forest_10, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_10)
  misclass_10<-c(misclass_10,misclass) #misclass errors for tree_10

  prediction_100<-predict(random_forest_100, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_100)
  misclass_100<-c(misclass_100,misclass) #misclass errors for tree_100
}

#computing mean of misclassification errors
Mean<-c(mean(misclass_1),mean(misclass_10),mean(misclass_100))

#computing variance of misclassification errors
Variance<-c(var(misclass_1),var(misclass_10),var(misclass_100))

#storing mean and variance in a dataframe
df<-data.frame(Mean, Variance)
rownames(df)<-c("tree_1","tree_10","tree_100")
```

The mean and variance of the misclassification errors is given by

```r
df
```

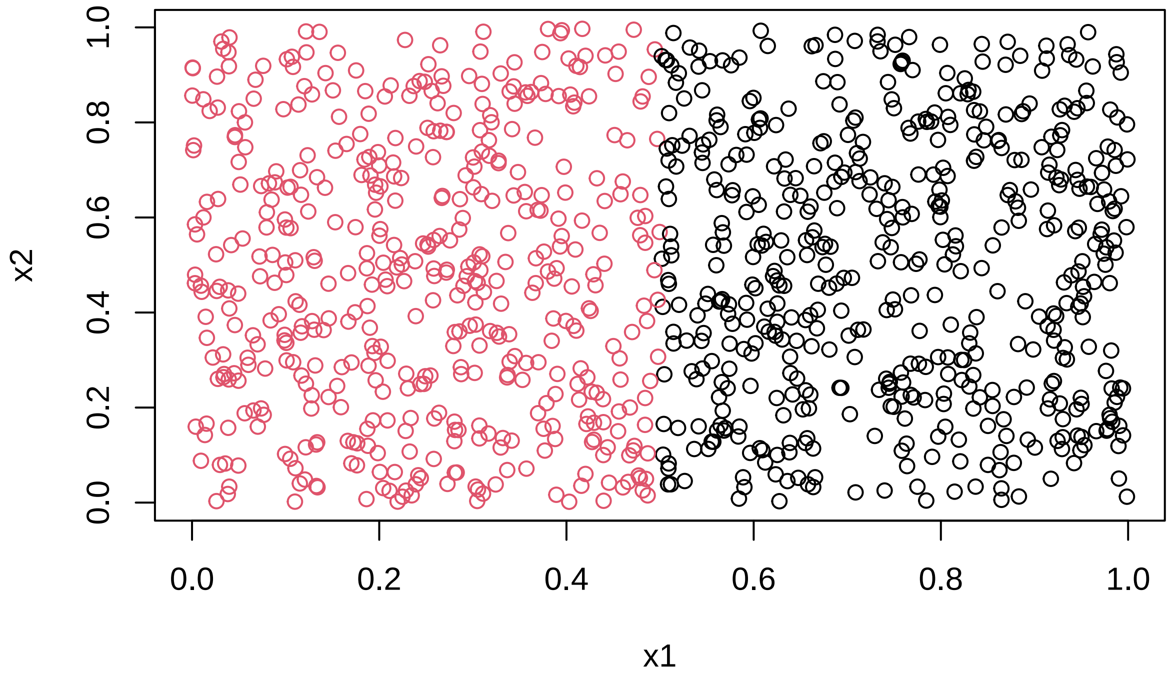
```
##                Mean     Variance
## tree_1   0.210170 0.0034808580
## tree_10  0.134469 0.0009964535
## tree_100 0.113275 0.0008700835
```

Q2. Repeat the exercise above but this time use the condition (x1<0.5) instead of (x1<x2) when producing the training and test datasets.

Creating training and test data

```
#Creating 1000 training datasets and storing them in a list
train_sets_1<-list()
for (i in 1:1000){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<0.5)
  trlabels<-as.factor(y)
  y<-trlabels
  trdata<-data.frame(x1,x2,y)
  train_sets_1[[i]]<-trdata
}

#test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y1<-as.numeric(x1<0.5)
telabels<-as.factor(y1)
y<-telabels
tedata<-data.frame(x1,x2,y)
plot(x1,x2,col=(y1+1))
```



Random forest models and computation of misclassification errors

```r
#three vectors to store misclassification errors for different trees
misclass_1<-c()
misclass_10<-c()
misclass_100<-c()

#learning a random forest with 1, 10 and 100 trees
for (i in train_sets_1){
  random_forest_1<-randomForest(y~., data=i, ntree=1, nodesize=25, keep.forest=TRUE)
  random_forest_10<-randomForest(y~., data=i, ntree=10, nodesize=25, keep.forest=TRUE)
  random_forest_100<-randomForest(y~., data=i, ntree=100, nodesize=25, keep.forest=TRUE)

  #computing misclassification errors with test data
  prediction_1<-predict(random_forest_1, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_1)
  misclass_1<-c(misclass_1,misclass) #misclass errors for tree_1

  prediction_10<-predict(random_forest_10, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_10)
  misclass_10<-c(misclass_10,misclass) #misclass errors for tree_10

  prediction_100<-predict(random_forest_100, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_100)
  misclass_100<-c(misclass_100,misclass) #misclass errors for tree_100
}
```

```
#computing mean of misclassification errors
Mean<-c(mean(misclass_1),mean(misclass_10),mean(misclass_100))

#computing variance of misclassification errors
Variance<-c(var(misclass_1),var(misclass_10),var(misclass_100))

#storing mean and variance in a dataframe
df<-data.frame(Mean, Variance)
rownames(df)<-c("tree_1","tree_10","tree_100")
```

The mean and variance of the misclassification errors is given by

```
df
```

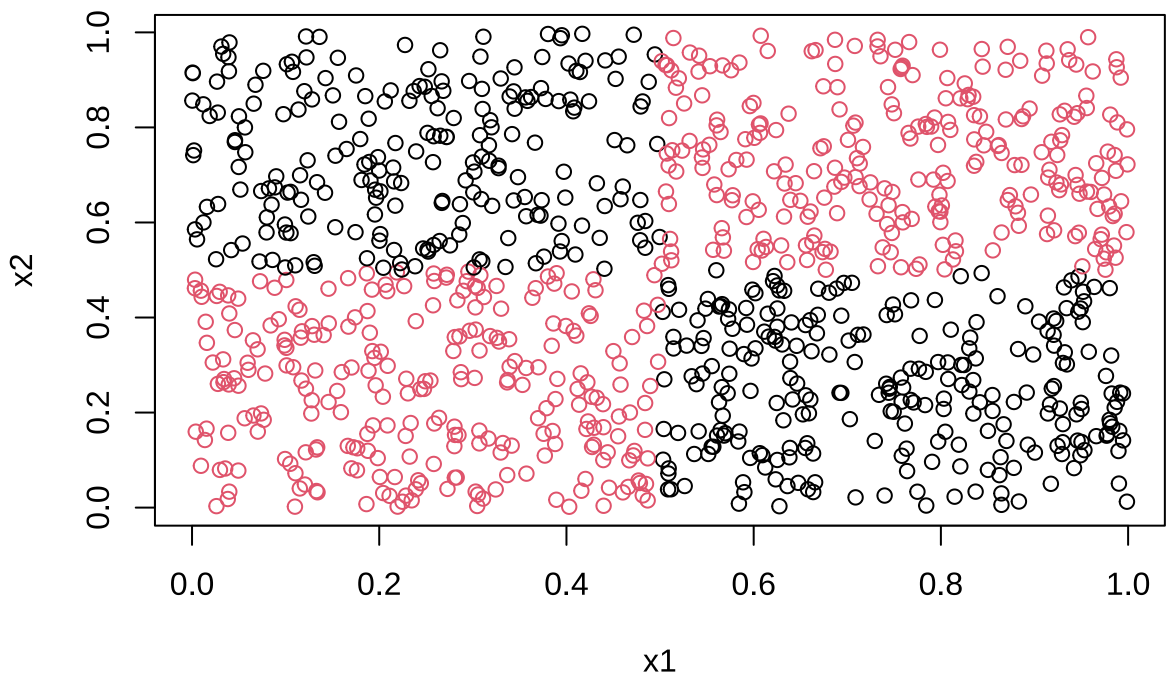
```
##              Mean     Variance
## tree_1   0.094772 1.758756e-02
## tree_10  0.017423 7.304866e-04
## tree_100 0.006934 8.044008e-05
```

Q3. Repeat the exercise above but this time use the condition ((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5)) instead of (x1<x2) when producing the training and test datasets. Unlike above, use nodesize = 12 for this exercise.

Creating training and test data

```
#Creating 1000 training datasets and storing them in a list
train_sets_1<-list()
for (i in 1:1000){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric((x1<0.5 & x2<0.5)|(x1>0.5 & x2>0.5))
  trlabels<-as.factor(y)
  y<-trlabels
  trdata<-data.frame(x1,x2,y)
  train_sets_1[[i]]<-trdata
}

#test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y1<-as.numeric((x1<0.5 & x2<0.5)|(x1>0.5 & x2>0.5))
telabels<-as.factor(y1)
y<-telabels
tedata<-data.frame(x1,x2,y)
plot(x1,x2,col=(y1+1))
```



Random forest models and computation of misclassification errors

```
#three vectors to store misclassification errors for different trees
misclass_1<-c()
misclass_10<-c()
misclass_100<-c()

#learning a random forest with 1, 10 and 100 trees
for (i in train_sets_1){
  random_forest_1<-randomForest(y~., data=i, ntree=1, nodesize=12, keep.forest=TRUE)
  random_forest_10<-randomForest(y~., data=i, ntree=10, nodesize=12, keep.forest=TRUE)
  random_forest_100<-randomForest(y~., data=i, ntree=100, nodesize=12, keep.forest=TRUE)

  #computing misclassification errors with test data
  prediction_1<-predict(random_forest_1, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_1)
  misclass_1<-c(misclass_1,misclass) #misclass errors for tree_1

  prediction_10<-predict(random_forest_10, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_10)
  misclass_10<-c(misclass_10,misclass) #misclass errors for tree_10

  prediction_100<-predict(random_forest_100, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_100)
  misclass_100<-c(misclass_100,misclass) #misclass errors for tree_100
}
```

```
#computing mean of misclassification errors
Mean<-c(mean(misclass_1),mean(misclass_10),mean(misclass_100))

#computing variance of misclassification errors
Variance<-c(var(misclass_1),var(misclass_10),var(misclass_100))

#storing mean and variance in a dataframe
df<-data.frame(Mean, Variance)
rownames(df)<-c("tree_1","tree_10","tree_100")
```

The mean and variance of the misclassification errors is given by

```
df
```

```
##              Mean    Variance
## tree_1   0.252983 0.013294409
## tree_10  0.123215 0.003219532
## tree_100 0.077395 0.001356680
```

Q4. Answer the following questions:

– What happens with the mean error rate when the number of trees in the random forest grows? Why?

The mean error rate is decreasing with increase in number of trees in random forest. When ensemble members are less, the misclassification rate is high and as the number of ensemble members increases, the model becomes better. More ensemble members does not make the model more flexible but it reduces only the variance.

– The third dataset represents a slightly more complicated classification problem than the first one. Still, you should get better performance for it when using sufficient trees in the random forest. Explain why you get better performance

The idea behind Random Forests is to inject additional randomness when construction each tree. We pick random subset of features for constructing each ensemble tree. If the number of features is more and number of ensembles is less then there is a chance that some of the features has less contribution to the construction of model which leads to reduction of prediction power of the model. But for the third data set in our case, it has only two features (x1 and x2) which is very smaller than number of ensemble trees(12). As we know the variance decreases with number of ensembles increases, this model has great predictive power which leads to better performance.

2 Assignment 2 : MIXTURE MODELS

Our Task is to implement the EM algorithm for Bernoulli mixture model. Please use the R template below to solve the assignment. Then, use your implementation to show what happens when your mixture model has too few and too many clusters, i.e. set M = 2,3,4 and compare results.

A Bernoulli mixture model is

$$p(x) = \sum_{m=1}^{M} \pi_m Bern(x|\mu_m)$$

where $x=(x_1, . . . ,x_D) $ is a D-dimensional binary random vector, π_m = p(y = m) and

$$Bernoulli(x|\mu_m) = \prod_{d=1}^{D} \mu_{m,d}^{xd}(1-\mu_{m,d})^{(1-x_d)}$$

E-step : Compute p(w|X) for w matrix

$$p(w_{nm}|x_n, \mu, \pi) = \frac{\pi_m p(x_n|\mu_m)}{\sum_m \pi_m p(x_n|\mu_m)}$$

Log likelihood computation The log likelihood function is

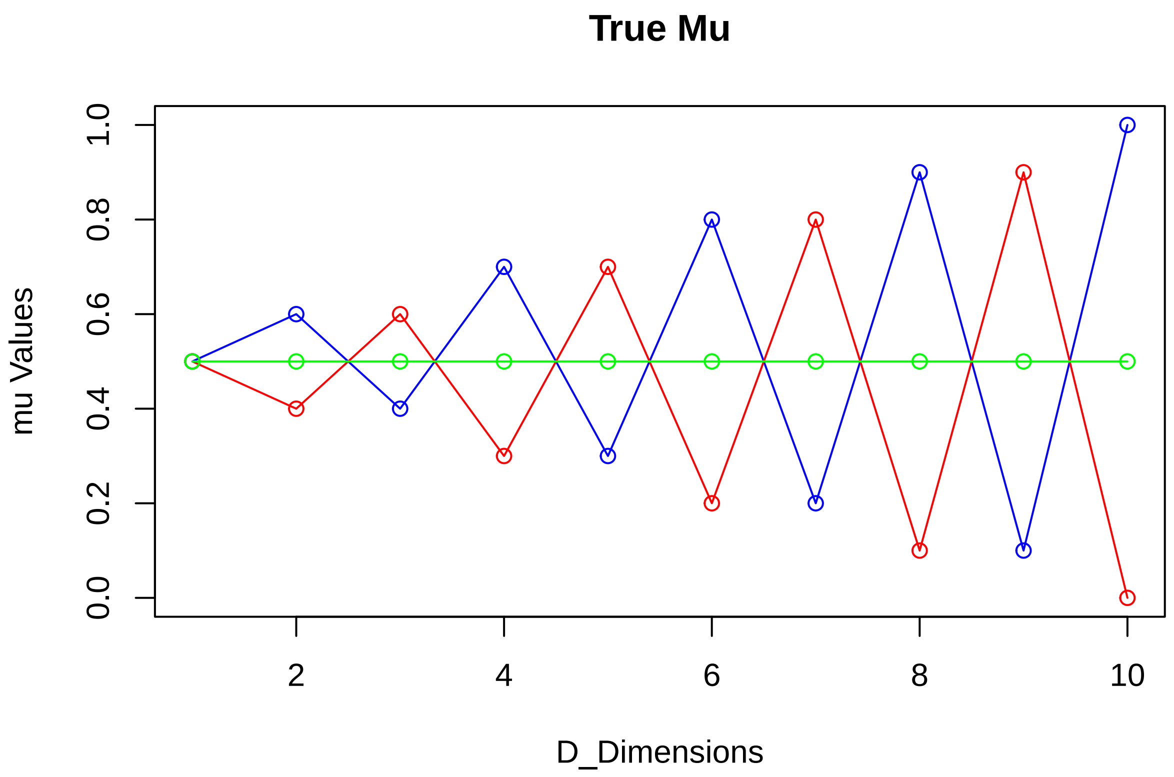
$$logp(\{x_n, w_n\}|\mu, \pi) = \sum_n logp(x_n, w_n|\mu, \pi) = \sum_n log \prod_m [\pi_m \prod_d \mu_{md}^{x_{nd}}(1-\mu_{md}^{(1-x_{nd})})]^{z_{nm}}$$

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

$$\pi_m^{ML} = \frac{\sum_n p(w_{nm}|x_n, \mu, \pi)}{n}$$

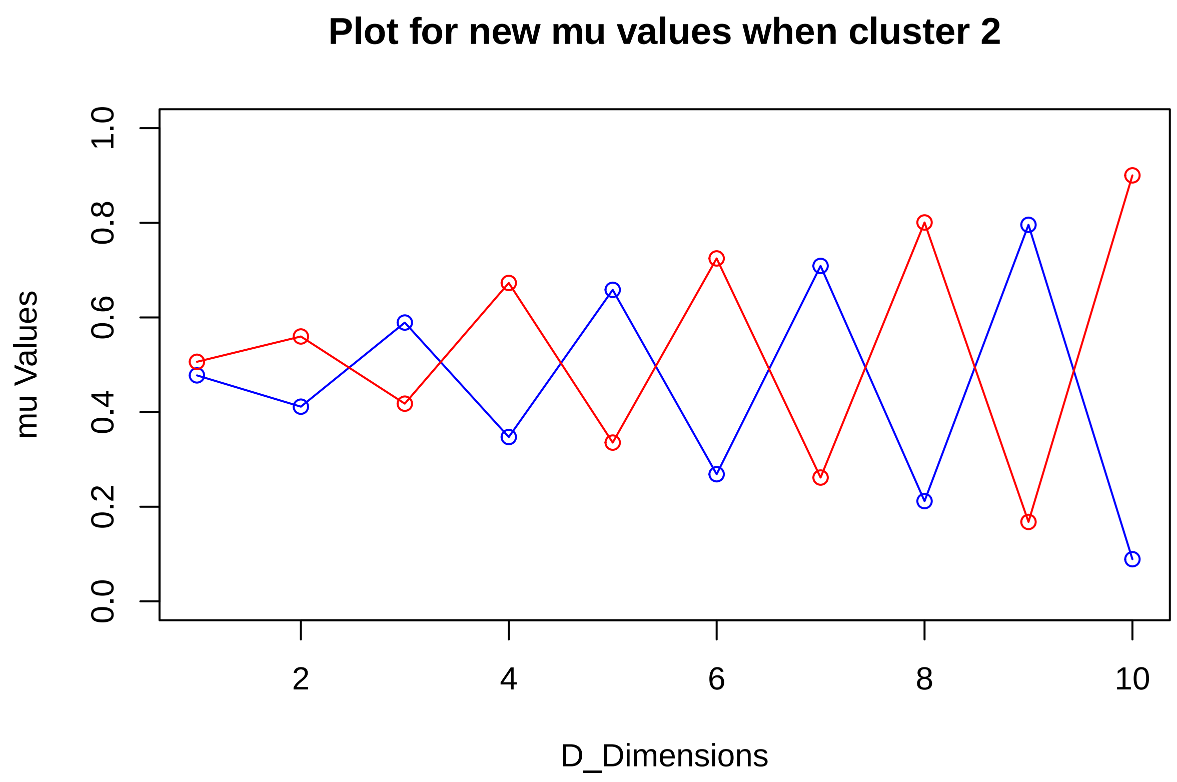
$$\mu_{md}^{ML} = \frac{\sum_n x_{ni} p(w_{nm}|x_n, \mu, \pi)}{\sum_n p(w_{nm}|x_n, \mu, \pi)}$$

Finally, the EM algorithm generates the plotting of observed data log-likelihood as a function of iteration number.EM algorithm has the property to increase the likelihood at each step.

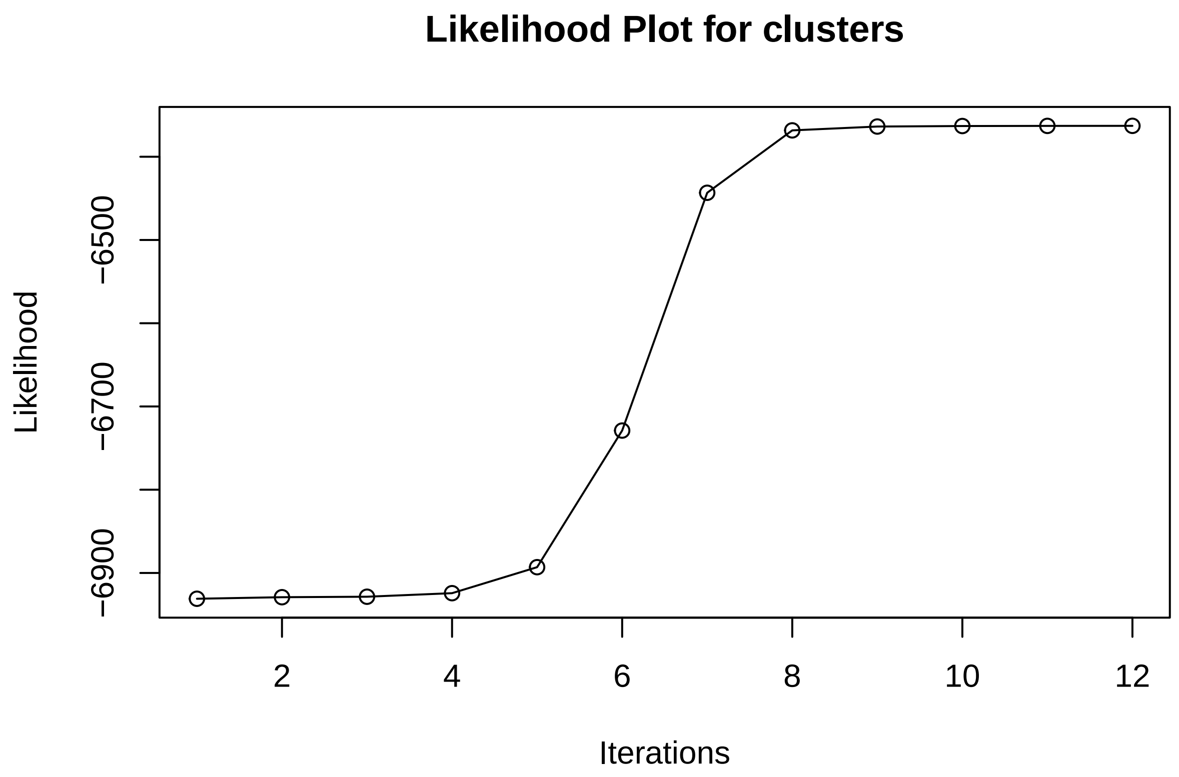


2.1 For M=2 :

```
## 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
```



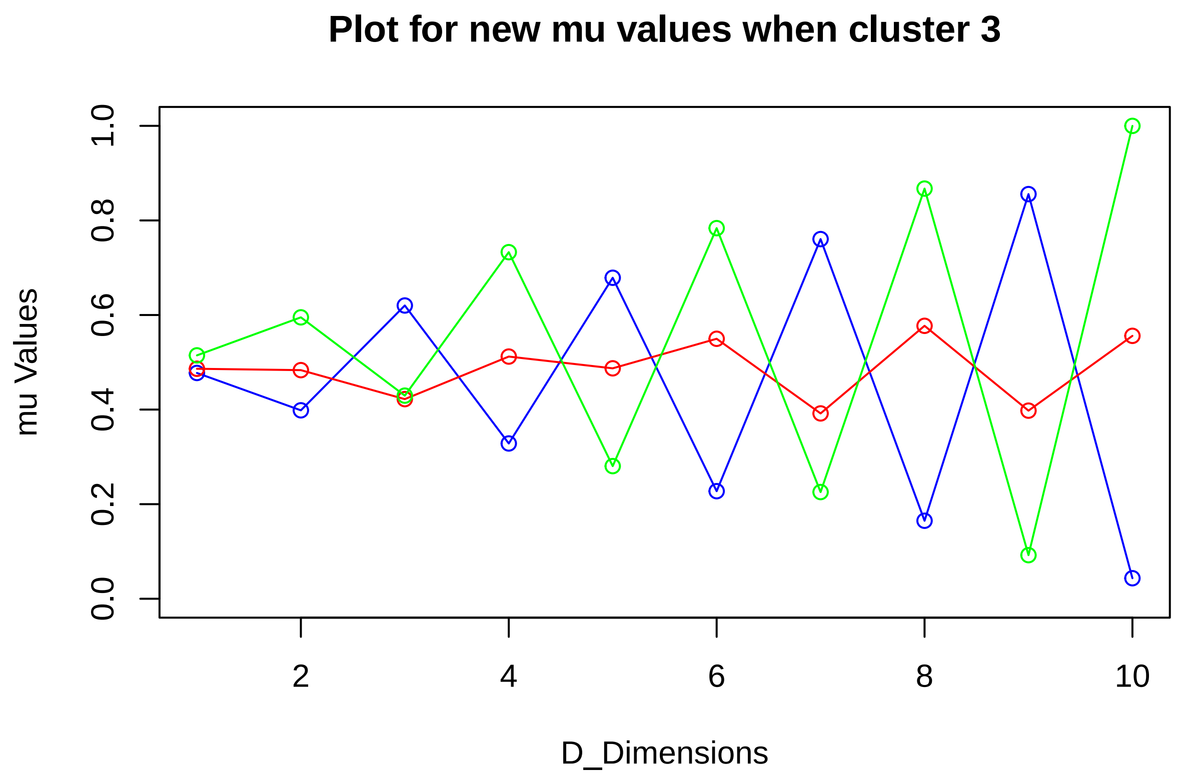
```
## PI :  0.497125 0.502875
## MU :          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4775488 0.4113939 0.5892308 0.3472420 0.6583712 0.2686589 0.7089490
## [2,] 0.5062860 0.5597531 0.4177551 0.6728856 0.3354854 0.7247188 0.2616231
##           [,8]      [,9]      [,10]
## [1,] 0.2118629 0.7957549 0.08905747
## [2,] 0.8007511 0.1678555 0.90027808
```



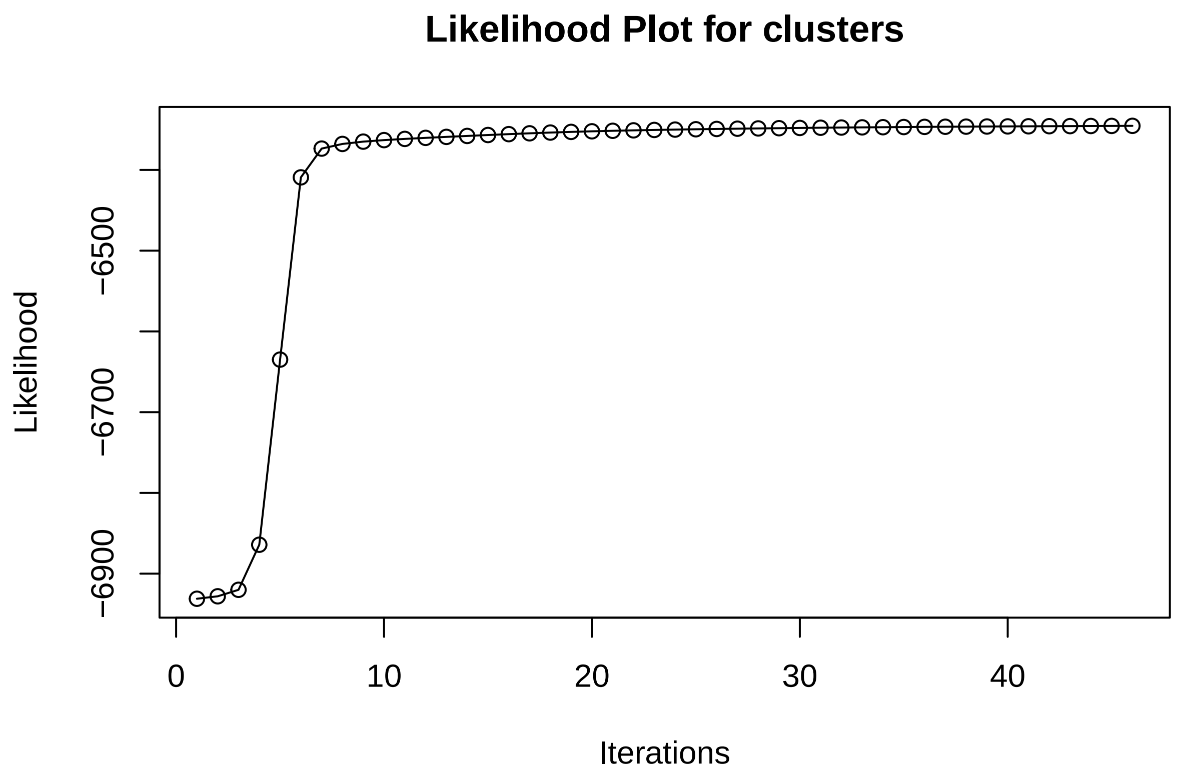
The plot above is generated using M = 2,so the number of clusters are assumed to be 2.After 12 iterations, the EM algorithm stops with a log likelihood of -6362.897, which considerably increases until it meets the stopping criterion condition.Thus showing that there hasn't been a significant change in the log likelihood.When the clusters are too less, this may lead to the under fitting. Therefore, we expect the model to be under fitted.

2.2 For M=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
```



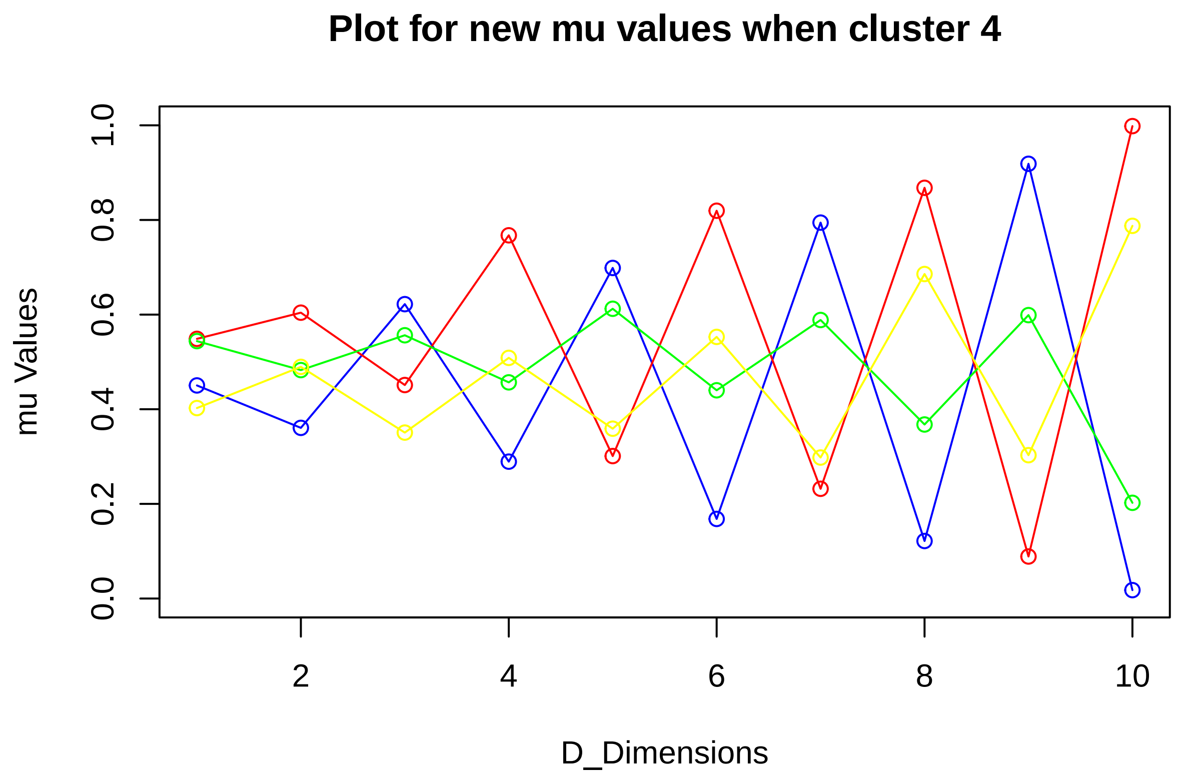
```
## PI :  0.3964172 0.278708 0.3248749
## MU :           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4774399 0.3984407 0.6200854 0.3283412 0.6787726 0.2274644 0.7605397
## [2,] 0.4863053 0.4834283 0.4221438 0.5121400 0.4872729 0.5497432 0.3918968
## [3,] 0.5146518 0.5950474 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
```



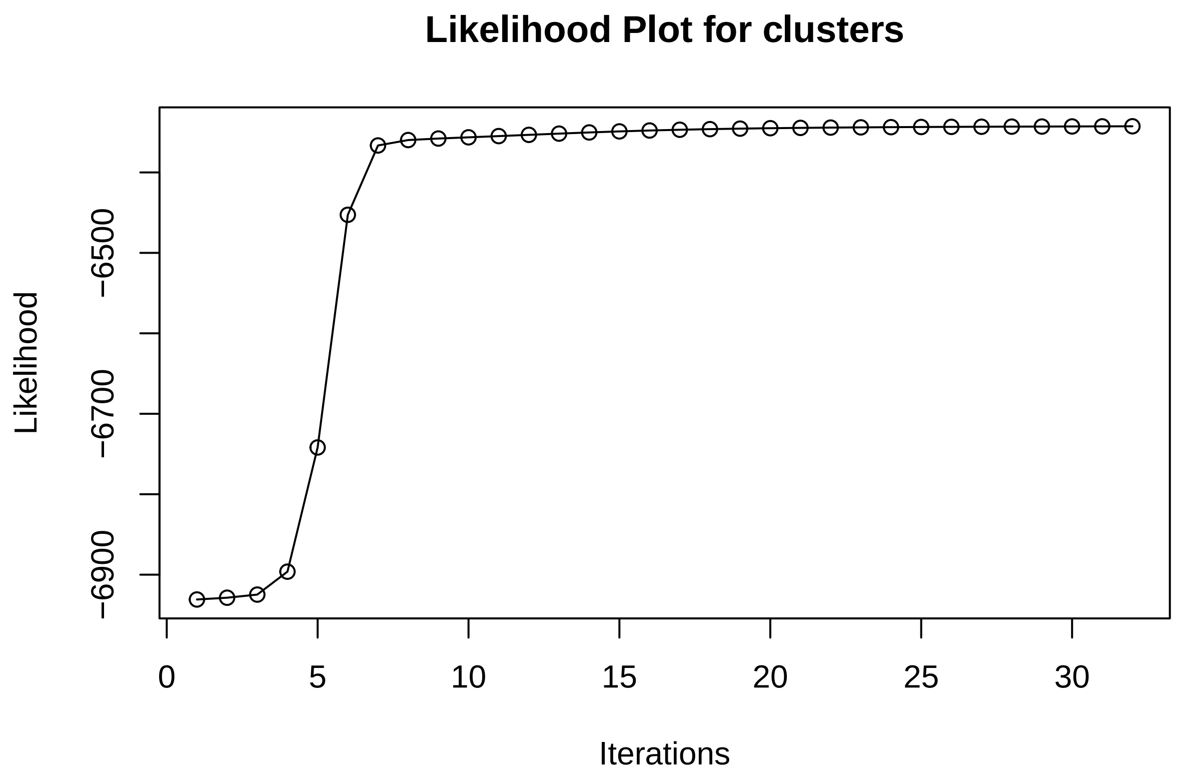
The plot above is generated using M = 3,so the number of clusters are assumed to be 3. It has been observed that, it is pretty similar to the true ones where the uniform one which has been influenced by the other two distributions.After 46 iterations, the EM algorithm stops with a log likelihood of -6345.431, which considerably increases until it meets the stopping criterion condition.The values of pi and mu coverges the true values and it is close to value of true pi i.e,(true pi = 1/3).

2.3 For M=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
```



The plot above is generated using M = 4,so the number of clusters are assumed to be 4.The green and yellow curves are highly irregular and do not resemble the true distributions. However, taking the average of these two curves would approximate the Bernoulli distribution with uniform parameters quite well. This suggests that while the individual curves may be chaotic, they can still be useful in estimating the underlying distribution.

After 32 iterations, the EM algorithm stops with a log likelihood of -6342.668 , which considerably increases until it meets the stopping criterion condition.When there are too many factors taken into account, it can cause the model to be overly specific to the training data and not generalize well to new examples. This is known as overfitting. In the case of overfitting, the model may capture random noise in the data rather than the underlying patterns. The estimated values produced by the model may be very different from the true values, particularly when using a large number of clusters. In such cases, the EM algorithm may have difficulty distinguishing between the different distributions in the data.

3 Appendix

```
knitr::opts_chunk$set(echo = TRUE)
# Assignment 1
#training data
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
```

```r
trlabels<-as.factor(y)
#test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y1<-as.numeric(x1<x2)
telabels<-as.factor(y1)
y<-telabels
tedata<-data.frame(x1,x2,y)
plot(x1,x2,col=(y1+1))
#Creating 1000 training datasets and storing them in a list
train_sets_1<-list()
for (i in 1:1000){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<x2)
  trlabels<-as.factor(y)
  y<-trlabels
  trdata<-data.frame(x1,x2,y)
  train_sets_1[[i]]<-trdata
}

#test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y1<-as.numeric(x1<x2)
telabels<-as.factor(y1)
y<-telabels
tedata<-data.frame(x1,x2,y)


#A function to calculate misclassification error
missclass<- function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

#importing "randomForest" package
library(randomForest)

#three vectors to store misclassification errors for different trees
misclass_1<-c()
misclass_10<-c()
misclass_100<-c()

#learning a random forest with 1, 10 and 100 trees
for (i in train_sets_1){
  random_forest_1<-randomForest(y~., data=i, ntree=1, nodesize=25, keep.forest=TRUE)
  random_forest_10<-randomForest(y~., data=i, ntree=10, nodesize=25, keep.forest=TRUE)
  random_forest_100<-randomForest(y~., data=i, ntree=100, nodesize=25, keep.forest=TRUE)

  #computing misclassification errors with test data
  prediction_1<-predict(random_forest_1, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_1)
  misclass_1<-c(misclass_1,misclass) #misclass errors for tree_1

  prediction_10<-predict(random_forest_10, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_10)
  misclass_10<-c(misclass_10,misclass) #misclass errors for tree_10

  prediction_100<-predict(random_forest_100, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_100)
  misclass_100<-c(misclass_100,misclass) #misclass errors for tree_100
}

#computing mean of misclassification errors
Mean<-c(mean(misclass_1),mean(misclass_10),mean(misclass_100))

#computing variance of misclassification errors
Variance<-c(var(misclass_1),var(misclass_10),var(misclass_100))

#storing mean and variance in a dataframe
df<-data.frame(Mean, Variance)
rownames(df)<-c("tree_1","tree_10","tree_100")
df
#Creating 1000 training datasets and storing them in a list
train_sets_1<-list()
for (i in 1:1000){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<0.5)
  trlabels<-as.factor(y)
  y<-trlabels
  trdata<-data.frame(x1,x2,y)
  train_sets_1[[i]]<-trdata
}

#test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y1<-as.numeric(x1<0.5)
telabels<-as.factor(y1)
y<-telabels
tedata<-data.frame(x1,x2,y)
plot(x1,x2,col=(y1+1))
#three vectors to store misclassification errors for different trees
misclass_1<-c()
misclass_10<-c()
misclass_100<-c()
```

```
#learning a random forest with 1, 10 and 100 trees
for (i in train_sets_1){
  random_forest_1<-randomForest(y~., data=i, ntree=1, nodesize=25, keep.forest=TRUE)
  random_forest_10<-randomForest(y~., data=i, ntree=10, nodesize=25, keep.forest=TRUE)
  random_forest_100<-randomForest(y~., data=i, ntree=100, nodesize=25, keep.forest=TRUE)

  #computing misclassification errors with test data
  prediction_1<-predict(random_forest_1, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_1)
  misclass_1<-c(misclass_1,misclass) #misclass errors for tree_1

  prediction_10<-predict(random_forest_10, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_10)
  misclass_10<-c(misclass_10,misclass) #misclass errors for tree_10

  prediction_100<-predict(random_forest_100, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_100)
  misclass_100<-c(misclass_100,misclass) #misclass errors for tree_100
}

#computing mean of misclassification errors
Mean<-c(mean(misclass_1),mean(misclass_10),mean(misclass_100))

#computing variance of misclassification errors
Variance<-c(var(misclass_1),var(misclass_10),var(misclass_100))

#storing mean and variance in a dataframe
df<-data.frame(Mean, Variance)
rownames(df)<-c("tree_1","tree_10","tree_100")
df
#Creating 1000 training datasets and storing them in a list
train_sets_1<-list()
for (i in 1:1000){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric((x1<0.5 & x2<0.5)|(x1>0.5 & x2>0.5))
  trlabels<-as.factor(y)
  y<-trlabels
  trdata<-data.frame(x1,x2,y)
  train_sets_1[[i]]<-trdata
}

#test data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y1<-as.numeric((x1<0.5 & x2<0.5)|(x1>0.5 & x2>0.5))
telabels<-as.factor(y1)
y<-telabels
tedata<-data.frame(x1,x2,y)
plot(x1,x2,col=(y1+1))
```

```r
#three vectors to store misclassification errors for different trees
misclass_1<-c()
misclass_10<-c()
misclass_100<-c()

#learning a random forest with 1, 10 and 100 trees
for (i in train_sets_1){
  random_forest_1<-randomForest(y~., data=i, ntree=1, nodesize=12, keep.forest=TRUE)
  random_forest_10<-randomForest(y~., data=i, ntree=10, nodesize=12, keep.forest=TRUE)
  random_forest_100<-randomForest(y~., data=i, ntree=100, nodesize=12, keep.forest=TRUE)

  #computing misclassification errors with test data
  prediction_1<-predict(random_forest_1, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_1)
  misclass_1<-c(misclass_1,misclass) #misclass errors for tree_1

  prediction_10<-predict(random_forest_10, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_10)
  misclass_10<-c(misclass_10,misclass) #misclass errors for tree_10

  prediction_100<-predict(random_forest_100, newdata=tedata)
  misclass<-missclass(tedata$y,prediction_100)
  misclass_100<-c(misclass_100,misclass) #misclass errors for tree_100
}

#computing mean of misclassification errors
Mean<-c(mean(misclass_1),mean(misclass_10),mean(misclass_100))

#computing variance of misclassification errors
Variance<-c(var(misclass_1),var(misclass_10),var(misclass_100))

#storing mean and variance in a dataframe
df<-data.frame(Mean, Variance)
rownames(df)<-c("tree_1","tree_10","tree_100")
df
# Assignment 2
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log lik between two consecutive 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 Mu", xlab ="D_Dimensions", ylab = "mu Values")
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
```

```r
# Producing the training data
for(i in 1:n) {
  m <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[i,d] <- rbinom(1,1,true_mu[m,d])
  }
}

Mixture_Model <- function(M_clusters){
  M=M_clusters # number of clusters
  w <- matrix(nrow=n, ncol=M) # weights
  pi <- vector(length = M) # mixing coefficients
  mu <- matrix(nrow=M, ncol=D) # conditional distributions
  llik <- vector(length = max_it) # log likelihood of the EM iterations

  # Random initialization of the parameters
  pi <- runif(M,0.49,0.51)
  pi <- pi / sum(pi)

  for(a in 1:M) {
    mu[a,] <- runif(D,0.49,0.51)
  }
  pi
  mu
  for(it in 1:max_it)
  {
    # E-step: Computation of the weights
    for (d in 1:n)
    {
      p_x <- 0
      for (m in 1:M)
      {
        #Bernouilli distributions
        p_x = p_x+ pi[m]*prod(((mu[m,]^x[d,])*((1-mu[m,])^(1-x[d,]))))
      }
      for (m in 1:M)
      {
        w[d,m] = pi[m]*prod(((mu[m,]^x[d,])*((1-mu[m,])^(1-x[d,])))) / p_x
      }
    }
    #Log likelihood computation.
    log_L <- matrix(0, nrow =1000, ncol = M)
    llik[it] <-0
    bern_MM <- 0
    for(d in 1:n)
    {
      for (m in 1:M)
      {
        bern_MM <- prod( ((mu[m,]^x[d,])*((1-mu[m,])^(1-x[d,]))))
        log_L[d,m] <- pi[m] * bern_MM
      }
      llik[it]<- sum(log(rowSums(log_L)))
    }
    cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    flush.console()

    # Stop if the log likelihood has not changed significantly
    if (it > 1)
    {
      if (llik[it]-llik[it-1] < min_change)
      {
        if(M==2)
        {
          plot(mu[1,], type="o", col="blue", ylim=c(0,1),
               main="Plot for new mu values when cluster 2",xlab ="D_Dimensions",ylab = "mu Values")
          points(mu[2,], type="o", col="red")
        }
        if(M==3)
        {
          plot(mu[1,], type="o", col="blue", ylim=c(0,1),
               main="Plot for new mu values when cluster 3",xlab ="D_Dimensions",ylab = "mu Values")
          points(mu[2,], type="o", col="red")
          points(mu[3,], type="o", col="green")
        }
        if(M==4)
        {
          plot(mu[1,], type="o", col="blue", ylim=c(0,1),
               main="Plot for new mu values when cluster 4",xlab ="D_Dimensions",ylab = "mu Values")
          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
    mu <- (t(w) %*% x) /colSums(w)
    pi <- colSums(w)/n
  }
  cat("PI : " , pi )
  cat("\n")
  cat("MU : ")
  print(mu)
  plot(llik[1:it], type="o", main="Likelihood Plot for clusters",xlab ="Iterations",ylab = "Likelihood"
}

# For M=2
Mixture_Model(2)
# For M=3
Mixture_Model(3)
# For M=4
Mixture_Model(4)
```