732A99/732A68/ TDDE01 Machine Learning

Computer lab 1 block 2

Johannes Hedström, Mikael Montén & Siddhesh Sreedar

STIMA
Department of Computer and Information Science
Linköpings universitet

2023-12-11

Contents

1 Contribution of work

Mikael, Siddhesh and Johannes collaborated all the assignments and discussed our results.

2 Ensemble methods

Your task is to learn some random forests using the function randomForest from the R package randomForest. The training data is produced by running the following R code:

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

The task is therefore classifying Y from X1 and X2, where Y is binary and X1 and X2 continuous. You should learn a random forest with 1, 10 and 100 trees, which you can do by setting the argument ntree to the appropriate value. Use nodesize = 25 and keep.forest = TRUE. The latter saves the random forest learned. You need it because you should also compute the misclassification error in the following test dataset (use the function predict for this purpose):

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

You can see that thee is a clear boundary between the 2 classes and it looks to be linear.

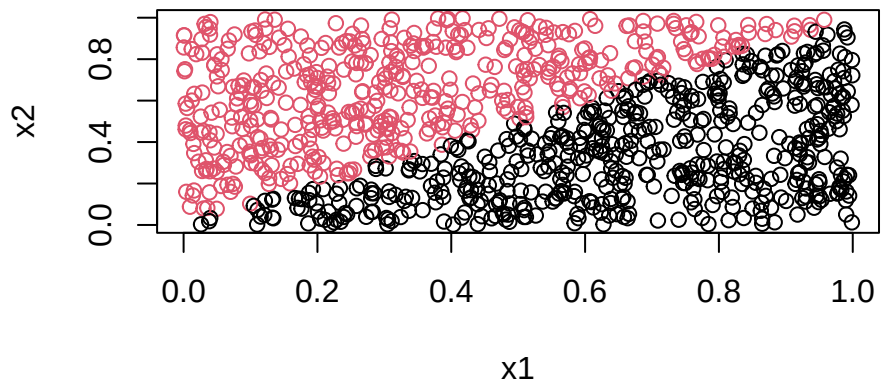

Figure 1: Generated points colored by class

2.1 1.

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.

Table 1: Misclassification error for the first 10 samples

1	10	100
0.171	0.113	0.084
0.253	0.135	0.105
0.185	0.113	0.091
0.158	0.100	0.083
0.146	0.176	0.159
0.186	0.174	0.129
0.135	0.104	0.084
0.145	0.121	0.094
0.289	0.093	0.080
0.187	0.158	0.078

Table 2: Mean of each random forest size for the 1000 samples

	x
1	0.209946

	x
10	0.135049
100	0.111160

Table 3: Variance of each random forest size for the 1000 samples

x
0.0034127
0.0009558
0.0008628

The random forest models with 100 trees have the lowest mean in misclassification error, so the more trees the better, we can also se that the variance is decreasing! However, there seems to be a marginal improvement between 10 and 100 trees.

2.2 2.

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

Table 4: Misclassification error for the first 10 samples

1	10	100
0.005	0.005	0.005
0.003	0.011	0.008
0.003	0.053	0.003
0.004	0.005	0.005
0.032	0.010	0.012
0.000	0.000	0.000
0.174	0.011	0.005
0.011	0.001	0.003
0.144	0.022	0.018
0.012	0.003	0.002

Table 5: Mean of each random forest size for the 1000 samples

	x
1	0.092951
10	0.013772
100	0.006414

Table 6: Variance of each random forest size for the 1000 samples

x
0.0172945
0.0004568
0.0000668

Means are lower than for the first iteration of the exercise, however variances has increased for all three.

2.3 3.

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.

Table 7: Misclassification error for the first 10 samples

	1	10	100
2	0.253	0.135	0.105
3	0.185	0.113	0.091
4	0.158	0.100	0.083
5	0.146	0.176	0.159
6	0.186	0.174	0.129
7	0.135	0.104	0.084
8	0.145	0.121	0.094
9	0.289	0.093	0.080
10	0.187	0.158	0.078
11	0.131	0.110	0.106

Table 8: Mean of each random forest size for the 1000 samples

	x
1	0.2305943
10	0.1289460
100	0.0940745

Table 9: Variance of each random forest size for the 1000 samples

x
0.0087581
0.0021155
0.0014303

Means and variances are situated somewhere in between the general range for exercise 1 and exercise 2 for all amounts of trees.

2.4 4

Answer the following questions:

2.4.1 a)

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

When increasing the number of trees we reduce the variance(different observation used) which is an effect from bagging and the correlations(different splits between the trees), and this will result in better predictions and lower mean error rate. The increasing of number of trees therefore makes the predictions more robust.

2.4.2 b)

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.

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

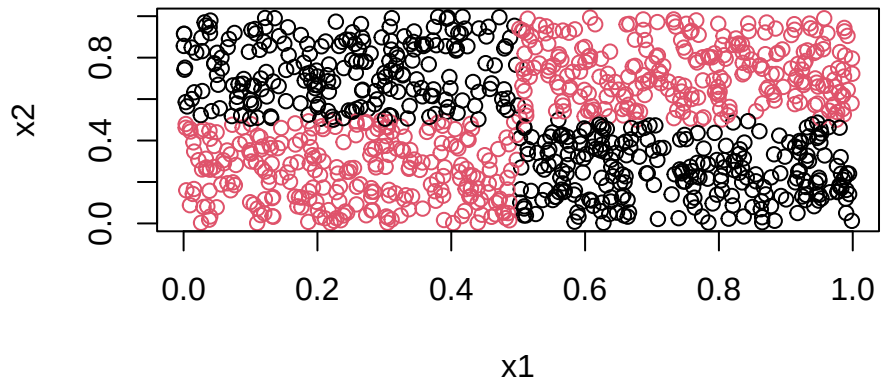

Figure 2: Generated points colored by class

We get a better performance as the boundary in the first one is linear and to make tree models to fit this problem is hard, we´ll need to have a really deep tree or several deep trees as in random forest and it will

probably never be really good. This complicated problem have more constant boundary for x2 which then switches after a certain value of x1, which makes it non-linear. When making several trees in random forest we will get lots of trees with different splits and all together

The decrease of nodesize might also improve the model for the more complex problem as it can grow larger trees, less observations needed in a node to create a split(this could also create overfit is nodesize is too small).

3 Mixture models

Your 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. Please provide a short explanation as well. A Bernoulli mixture model is

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

where $x = (x1, ..., x_D)$ is a D-dimensional binary random vector, $\pi m = p(y = m)$ and

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

where $\mu_m = (\mu_{m,1}, ..., \mu m, D)$ is a D-dimensional vector of probabilities. As usual, the log likelihood of the dataset $\{x_i\}_{i=1}^n$ is

$$\sum_{i=1}^{n} logp(x_i)$$

Finally, in the EM algorithm, the parameter updates for the Bernoulli mixture model are the same as for the Gaussian mixture model (see Equations 10.16a,b in the lecture slides).

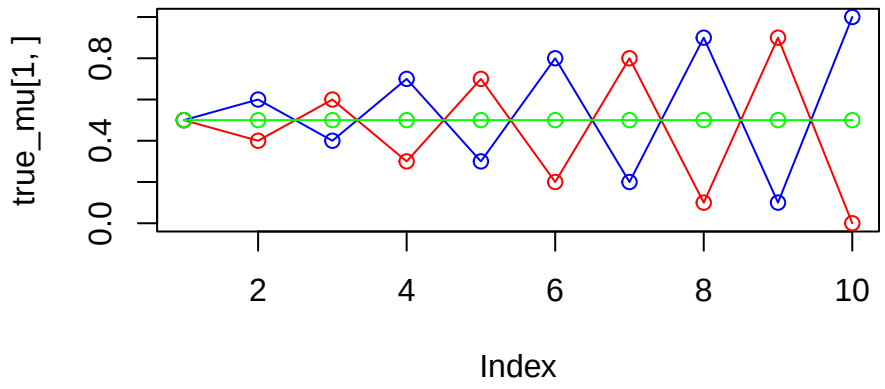

Figure 3: True Mu values

EM algorithm:

Learn the GMM

Data: Unlabeled training data $\mathcal{T} = \{\mathbf{x}_i\}_{i=1}^n$, number of clusters M.

Result: Gaussian mixture model

Initialize $\hat{\theta} = \left\{\hat{\pi}_m, \hat{\mu}_m, \widehat{\Sigma}_m\right\}_{m=1}^M$

repeat

For each $\mathbf{x}_i$ in $\{\mathbf{x}_i\}_{i=1}^n$, compute the prediction $p\left(y \mid \mathbf{x}_i, \hat{\theta}\right)$ according to (10.5) using the current parameter estimates $\hat{\theta}$.

Update the parameter estimates $\hat{\theta} \leftarrow \left\{\hat{\pi}_m, \hat{\mu}_m, \widehat{\Sigma}_m\right\}_{m=1}^M$ according to (10.16)

until convergence

Predict as QDA, Method 10.1

10.5 (Expectation):

$$w_{i,m}(x_i) = \frac{\pi_m Bern(x_i|\mu_m)}{\sum_{m=1}^M \pi_m Bern(x_i|\mu_m)}$$

10.16(a,b)(Maximization):

10.16a:

$$\hat{\pi}_m = \frac{1}{n}\sum_{i=1}^n w_i(m)$$

10.16b:

$$\hat{\mu}_m = \frac{1}{\sum_{i=1}^n w_i(m)}\sum_{i=1}^n w_i(m)\mathbf{x}_i$$

$$\text{where } w_i(m) = p\left(y_i = m \mid \mathbf{x}_i, \hat{\theta}\right)$$

First step is to create a Bernoulli function that is used to update the weights.

```
# Function for the Bernoulli product
bernoulli <- function(x,mu,M){

  mat <- matrix(0,ncol = M,nrow = nrow(x))
  for (i in 1:nrow(x)) { # for every x and every cluster calculate the prod over the dimensions

  for(m in 1:M){
  mat[i,m]<- prod(mu[m,]^x[i,] * ( 1 - mu[m,])^(1-x[i,]))

    }
  }
  mat
}

w <- bernoulli(x,mu,M)
```

Here we implement the Bernoulli function in the code as the E-step and the M-step is computed by log likelihood.

```
for(it in 1:max_it) {

#points(mu[4,], type="o", col="yellow")
Sys.sleep(0.1)

# E-step: Computation of the weights(Expectation)
w <- bernoulli(x,mu,M)

p_x <- (w*pi)/rowSums(w*pi) # calculate probabilities


#Log likelihood computation.(Maximization)
llik[it] <- sum(log(rowSums(w*pi)))


cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()


# Stop if the log likelihood has not changed significantly
if(it>=2 && abs((llik[it] - llik[it-1])) < min_change){
  break()
}
#M-step: ML parameter estimation from the data and weights

#10.16.a
pi <- (1/n) * colSums(p_x)


# 10.16b
for (i in 1:M) {
  for(j in 1:ncol(mu)){
  mu_hat <-  sum(p_x[,i] * x[,j])

  mu[i,j] <- mu_hat / colSums(p_x)[i]
    }
  }
}
```

```
## iteration:  1 log likelihood:  -6931.483
## iteration:  2 log likelihood:  -6929.074
## iteration:  3 log likelihood:  -6928.091
## iteration:  4 log likelihood:  -6920.654
## iteration:  5 log likelihood:  -6869.145
## iteration:  6 log likelihood:  -6655.294
## iteration:  7 log likelihood:  -6431.753
```

```
## iteration:  8 log likelihood:  -6378.968
## iteration:  9 log likelihood:  -6362.26
## iteration:  10 log likelihood:  -6355.272
## iteration:  11 log likelihood:  -6351.729
## iteration:  12 log likelihood:  -6349.692
## iteration:  13 log likelihood:  -6348.405
## iteration:  14 log likelihood:  -6347.539
## iteration:  15 log likelihood:  -6346.929
## iteration:  16 log likelihood:  -6346.488
## iteration:  17 log likelihood:  -6346.163
## iteration:  18 log likelihood:  -6345.918
## iteration:  19 log likelihood:  -6345.732
## iteration:  20 log likelihood:  -6345.59
## iteration:  21 log likelihood:  -6345.479
## iteration:  22 log likelihood:  -6345.393
```

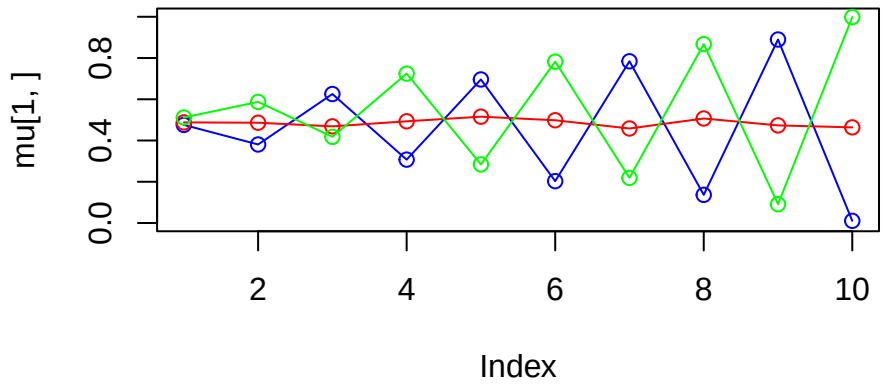

Figure 4: Estimated Mu values for M = 3

Comparing this figure with Figure 3 shows that each of the 3 μ values seems to have converged to a good estimate with respect to the true values.

```
## [1] 0.3301220 0.3276797 0.3421983
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4756868 0.3807664 0.6254036 0.3073594 0.6960711 0.2029757 0.7841952
## [2,] 0.4878853 0.4863472 0.4685707 0.4932018 0.5156753 0.4981670 0.4583882
## [3,] 0.5116776 0.5871875 0.4178847 0.7244971 0.2841490 0.7824529 0.2189240
##           [,8]       [,9]      [,10]
## [1,] 0.1365602 0.88976292 0.01039218
```

```
## [2,] 0.5067311 0.47335207 0.46369297
## [3,] 0.8675466 0.09106362 0.99832918
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  0.5  0.6  0.4  0.7  0.3  0.8  0.2  0.9  0.1   1.0
## [2,]  0.5  0.4  0.6  0.3  0.7  0.2  0.8  0.1  0.9   0.0
## [3,]  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5   0.5
```

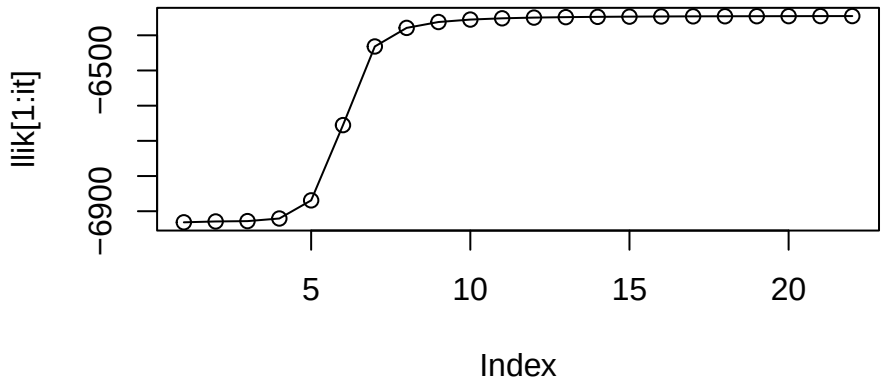

Figure 5: Likelihood value for each iteration

The tables produce confirm what the figure says, that all three true μ values are found when looking for three clusters of mu.

Next step is to do the same computations but for $M = 2$ and $M = 4$.

```
## iteration:  1 log likelihood:  -6931.374
## iteration:  2 log likelihood:  -6929.087
## iteration:  3 log likelihood:  -6928.06
## iteration:  4 log likelihood:  -6920.368
## iteration:  5 log likelihood:  -6867.049
## iteration:  6 log likelihood:  -6654.234
## iteration:  7 log likelihood:  -6462.067
## iteration:  8 log likelihood:  -6416.521
## iteration:  9 log likelihood:  -6394.191
## iteration:  10 log likelihood:  -6382.034
## iteration:  11 log likelihood:  -6374.614
## iteration:  12 log likelihood:  -6369.492
## iteration:  13 log likelihood:  -6365.42
## iteration:  14 log likelihood:  -6361.752
## iteration:  15 log likelihood:  -6358.227
```

```
## iteration:  16 log likelihood:  -6354.854
## iteration:  17 log likelihood:  -6351.8
## iteration:  18 log likelihood:  -6349.253
## iteration:  19 log likelihood:  -6347.314
## iteration:  20 log likelihood:  -6345.953
## iteration:  21 log likelihood:  -6345.057
## iteration:  22 log likelihood:  -6344.488
## iteration:  23 log likelihood:  -6344.131
## iteration:  24 log likelihood:  -6343.903
## iteration:  25 log likelihood:  -6343.751
## iteration:  26 log likelihood:  -6343.646
## iteration:  27 log likelihood:  -6343.567
```

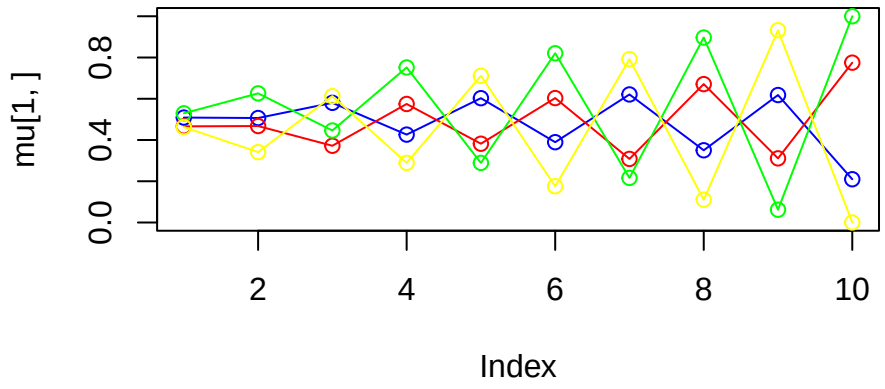

Figure 6: Estimated Mu values for M = 4

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.5087419 0.5066354 0.5801461 0.4261089 0.6028506 0.3893984 0.6208431
## [2,] 0.4662542 0.4677062 0.3721016 0.5747524 0.3819012 0.6033428 0.3081692
## [3,] 0.5294206 0.6262544 0.4460304 0.7517667 0.2889553 0.8206528 0.2162520
## [4,] 0.4627628 0.3412873 0.6128723 0.2889906 0.7116117 0.1756391 0.7921151
##           [,8]       [,9]        [,10]
## [1,] 0.3500757 0.61805851 0.2096617802
## [2,] 0.6710697 0.31098481 0.7750684454
## [3,] 0.8970426 0.06200737 0.9999910780
## [4,] 0.1106859 0.93241912 0.0001195492

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  0.5  0.6  0.4  0.7  0.3  0.8  0.2  0.9  0.1   1.0
## [2,]  0.5  0.4  0.6  0.3  0.7  0.2  0.8  0.1  0.9   0.0
## [3,]  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5   0.5
```

Two of the three true mu values are found when looking for four clusters of μ, the other two of the clusters seems to be a combination of the third true mu.

```
## iteration:  1 log likelihood:  -6930.976
## iteration:  2 log likelihood:  -6929.125
## iteration:  3 log likelihood:  -6928.562
## iteration:  4 log likelihood:  -6924.281
## iteration:  5 log likelihood:  -6893.056
## iteration:  6 log likelihood:  -6728.956
## iteration:  7 log likelihood:  -6443.309
## iteration:  8 log likelihood:  -6368.352
## iteration:  9 log likelihood:  -6363.755
## iteration:  10 log likelihood:  -6363.12
## iteration:  11 log likelihood:  -6362.951
## iteration:  12 log likelihood:  -6362.898
## iteration:  13 log likelihood:  -6362.881
## iteration:  14 log likelihood:  -6362.875
## iteration:  15 log likelihood:  -6362.872
```

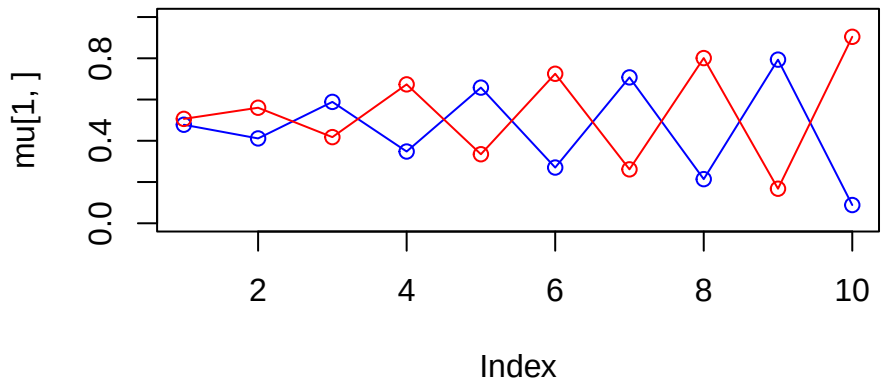

Figure 7: Estimated Mu values for M = 2

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4777359 0.4114562 0.5885945 0.3479409 0.6578324 0.2702388 0.7073550
## [2,] 0.5062149 0.5602865 0.4177010 0.6734963 0.3347262 0.7249750 0.2614159
##           [,8]      [,9]      [,10]
## [1,] 0.2139200 0.7935428 0.08834236
## [2,] 0.8010649 0.1675395 0.90424706
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  0.5  0.6  0.4  0.7  0.3  0.8  0.2  0.9  0.1   1.0
## [2,]  0.5  0.4  0.6  0.3  0.7  0.2  0.8  0.1  0.9   0.0
## [3,]  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5   0.5
```

Two of the three true mu values are found when looking for 2 clusters of μ.

Table 10: Log-likelihood for the different M's

	M=3	M=4	M=2
Log-likelihood	-6345	-6344	-6363
Iterations	22	27	15

We get different number of iterations for M's, the one with the lowest amount of iterations is for M=2 and the one with the most is for M=4. The log likelihood is lowest for M=4 and this is expected as increasing the value of M will lead to a better fit of the data, but this is only for the training data so it will not return a generalized model.

3.0.1 Comparing the pi values for the different M's

```
## $`M=2`
## [1] 0.4991356 0.5008644
##
## $`M=3`
## [1] 0.3301220 0.3276797 0.3421983
##
## $`M=4`
## [1] 0.2501201 0.2469146 0.2531561 0.2498092
```

The probability π sum to 1 for all models as expected and the probability looks to be divided equally between each group for all models.

4 Appendix

```
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning=FALSE, fig.width = 5, fig.height = 3, fig.a
set.seed(12345)
# packages
library(glmnet)
library(caret)
library(dplyr)
library(ggplot2)
library(randomForest)



x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
telabels<-as.factor(y)
plot(x1,x2,col=(y+1))

# creating 1000 datasets and making 3 random forest model for each dataset

df <- data.frame('1'=c(0),'10'=c(0),'100'=c(0))
df <- df[-1,]

for(i in 1:1000){

x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)


 vec <- c()
 k <- 1
# creating the models
for (j in c(1,10,100)) {

 mod <-  randomForest(x=trdata, y=trlabels, nodesize = 25,keep.forest =TRUE, ntree = j)

 vec[k] <- 1- (sum(diag(table(predict(mod,tedata ),telabels )))/1000)
```

```
 k <- k + 1
}

df <- rbind(df,vec)

}


colnames(df) <- c('1','10','100')

knitr::kable(head(df,10),caption = 'Misclassification error for the first 10 samples' )


knitr::kable(colMeans(df), caption = 'Mean of each random forest size for the 1000 samples')
knitr::kable(c(var(df[,1]),var(df[,2]),var(df[,3])), caption = 'Variance of each random forest size for



# creating 1000 datasets and making 3 random forest model for each dataset

df2 <- data.frame('1'=c(0),'10'=c(0),'100'=c(0))
df2 <- df2[-1,]

set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(x1<0.5)
telabels<-as.factor(y)

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)


 vec <- c()
 k <- 1
# creating the models
for (j in c(1,10,100)) {

 mod <-  randomForest(x=trdata, y=trlabels, nodesize = 25,keep.forest =TRUE, ntree = j)

 vec[k] <- 1- (sum(diag(table(predict(mod,tedata ),telabels )))/1000)
```

```
 k <- k + 1
}

df2 <- rbind(df2,vec)

}


colnames(df2) <- c('1','10','100')

knitr::kable(head(df2,10),caption = 'Misclassification error for the first 10 samples' )


knitr::kable(colMeans(df2), caption = 'Mean of each random forest size for the 1000 samples')

knitr::kable(c(var(df2[,1]),var(df2[,2]),var(df2[,3])), caption = 'Variance of each random forest size f
# creating 1000 datasets and making 3 random forest model for each dataset

df3 <- data.frame('1'=c(0),'10'=c(0),'100'=c(0))
df3 <- df[-1,]

set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
tedata<-cbind(x1,x2)
y <- as.numeric(((x1<0.5 & x2<0.5)| (x1>0.5 & x2>0.5)))
telabels<-as.factor(y)

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)


 vec <- c()
 k <- 1
# creating the models
for (j in c(1,10,100)) {

 mod <-  randomForest(x=trdata, y=trlabels, nodesize = 12,keep.forest =TRUE, ntree = j)

 vec[k] <- 1- (sum(diag(table(predict(mod,tedata ),telabels )))/1000)

 k <- k + 1
}
```

```
df3 <- rbind(df3,vec)

}


colnames(df3) <- c('1','10','100')

knitr::kable(head(df3,10),caption = 'Misclassification error for the first 10 samples' )


knitr::kable(colMeans(df3), caption = 'Mean of each random forest size for the 1000 samples')
knitr::kable(c(var(df3[,1]),var(df3[,2]),var(df3[,3])), caption = 'Variance of each random forest size f
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5)))
telabels<-as.factor(y)
plot(x1,x2,col=(y+1))

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))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
set.seed(1234567890)
# 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])
  }
}

M=3 # 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(m in 1:M) {
mu[m,] <- runif(D,0.49,0.51)
  }
# Function for the Bernoulli product
bernoulli <- function(x,mu,M){

  mat <- matrix(0,ncol = M,nrow = nrow(x))
  for (i in 1:nrow(x)) { # for every x and every cluster calculate the prod over the dimensions

  for(m in 1:M){
  mat[i,m]<- prod(mu[m,]^x[i,] * ( 1 - mu[m,])^(1-x[i,]))

    }
  }
  mat
}

w <- bernoulli(x,mu,M)


set.seed(1234567890)
# 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])
  }
}

M=3 # 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(m in 1:M) {
mu[m,] <- runif(D,0.49,0.51)
}
for(it in 1:max_it) {

#points(mu[4,], type="o", col="yellow")
Sys.sleep(0.1)

# E-step: Computation of the weights(Expectation)
w <- bernoulli(x,mu,M)

p_x <- (w*pi)/rowSums(w*pi) # calculate probabilities


#Log likelihood computation.(Maximization)
llik[it] <- sum(log(rowSums(w*pi)))


cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()


# Stop if the log likelihood has not changed significantly
if(it>=2 && abs((llik[it] - llik[it-1])) < min_change){
  break()
}
#M-step: ML parameter estimation from the data and weights

#10.16.a
pi <- (1/n) * colSums(p_x)


# 10.16b
for (i in 1:M) {
  for(j in 1:ncol(mu)){
  mu_hat <-  sum(p_x[,i] * x[,j])

  mu[i,j] <- mu_hat / colSums(p_x)[i]
    }
  }
}
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")

pi_1 <- pi
```

```
l1 <- llik
pi
mu
true_mu
plot(llik[1:it], type="o")
set.seed(1234567890)
# 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])
  }
}

M=4 # 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(m in 1:M) {
mu[m,] <- runif(D,0.49,0.51)
}

for(it in 1:max_it) {

Sys.sleep(0.1)

# E-step: Computation of the weights(Expectation)
# Your code here

w <- bernoulli(x,mu,M)

#p_x <- matrix(0,ncol = M,nrow=nrow(x))


p_x <- (w*pi)/rowSums(w*pi)


#Log likelihood computation.(Maximization)
# Your code here
```

```r
# log-likelihood



llik[it] <- sum(log(rowSums(w*pi)))

cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()


# Stop if the log likelihood has not changed significantly
# Your code here

if(it>=15 && abs((llik[it] - llik[it-1])) < min_change){
  break()
}
#M-step: ML parameter estimation from the data and weights
# Your code here

#10.16.a
pi <- (1/n) * colSums(p_x)


# 10.16b

for (i in 1:M) {
  for(j in 1:ncol(mu)){
  mu_hat <-  sum(p_x[,i] * x[,j])

  mu[i,j] <- mu_hat / colSums(p_x)[i]
}
}
}
pi_2 <- pi
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")
l2 <- llik

mu
true_mu

set.seed(1234567890)
# 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])
```

```
  }
}

M=2 # 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(m in 1:M) {
mu[m,] <- runif(D,0.49,0.51)
}

for(it in 1:max_it) {

 # points(mu[3,], type="o", col="green")
  #points(mu[4,], type="o", col="yellow")
Sys.sleep(0.1)

# E-step: Computation of the weights(Expectation)
# Your code here

w <- bernoulli(x,mu,M)

#p_x <- matrix(0,ncol = M,nrow=nrow(x))


p_x <- (w*pi)/rowSums(w*pi)


#Log likelihood computation.(Maximization)
# Your code here


# log-likelihood



llik[it] <- sum(log(rowSums(w*pi)))

cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
```

```r
# Stop if the log likelihood has not changed significantly
# Your code here

if(it>=15 && abs((llik[it] - llik[it-1])) < min_change){
  break()
}
#M-step: ML parameter estimation from the data and weights
# Your code here

#10.16.a
pi <- (1/n) * colSums(p_x)


# 10.16b

for (i in 1:M) {
  for(j in 1:ncol(mu)){
  mu_hat <-  sum(p_x[,i] * x[,j])

  mu[i,j] <- mu_hat / colSums(p_x)[i]
}
}
}
pi_3 <- pi

plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  points(mu[2,], type="o", col="red")
l23<- llik

mu
true_mu

l_df <- data.frame(c(max(l1[l1 < 0]), 22),c(max(l2[l2 < 0]),27),c(max(l23[l23 < 0]),15))

rownames(l_df) <- c('Log-likelihood', 'Iterations')
colnames(l_df) <- c("M=3", "M=4", "M=2")

knitr::kable(l_df, caption="Log-likelihood for the different M's", digits=0)

pi_df <- list("M=2"=pi_3,'M=3'=pi_1,"M=4"=pi_2)
pi_df
```