Lab 1 Block 2 Report

Marijn Jaarsma & Simon Jorstedt & Hugo Morvan

2023-12-07

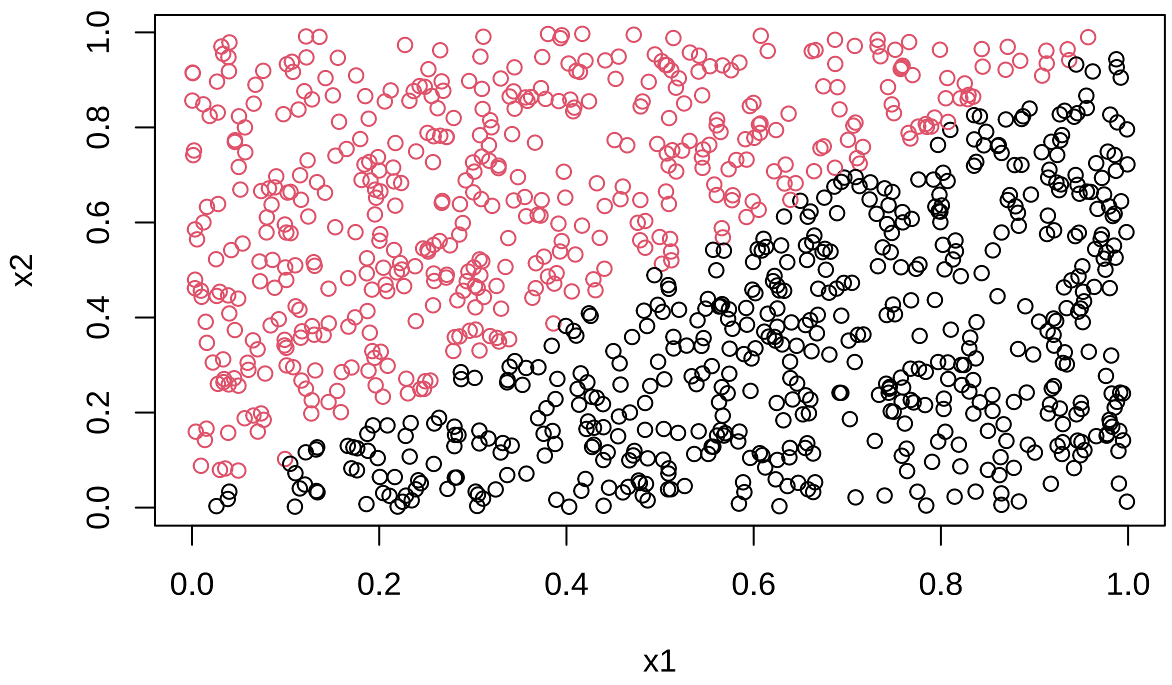
Statement of Contribution:

- Assignment 1 was coded and analysed by Simon and Hugo
- Assignment 2 was coded and analysed by Marijn All code and questions were analysed and discussed together.

1. Ensemble Methods

Your task is to learn some random forests using the function randomForest from the R package randomForest.

The task is therefore classifying Y from X_1 and X_2 , where Y is binary and X_1 and X_2 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):



```
## Mean misclassification error for 1 tree: 0.161
```

```
## Mean misclassification error for 10 trees: 0.173
```

```
## Mean misclassification error for 100 trees: 0.152
```

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.

```
## Overall mean for 1 tree: 0.209298
```

```
## Overall variance for 1 tree: 0.00299299
```

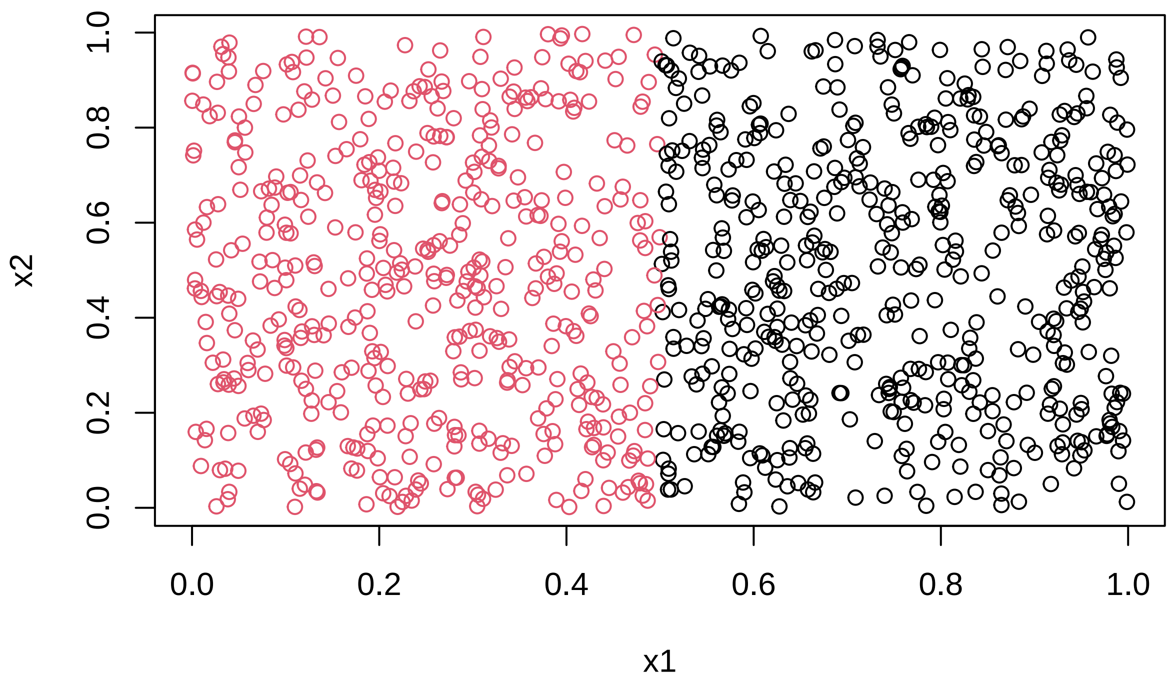
```
## Overall mean for 10 trees: 0.135861
```

```
## Overall variance for 10 trees: 0.0009381799
```

```
## Overall mean for 100 trees: 0.111803
```

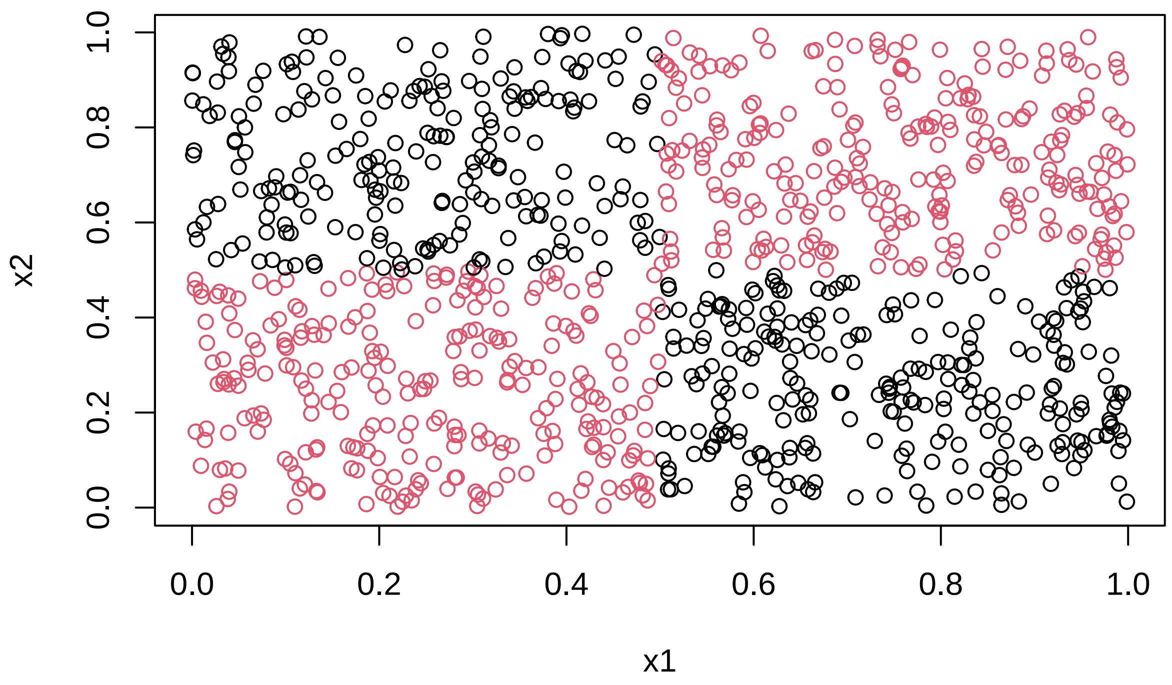
```
## Overall variance for 100 trees: 0.0009282444
```

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



```
## Overall mean for 1 tree: 0.092951

## Overall variance for 1 tree: 0.01729452

## Overall mean for 10 trees: 0.013772

## Overall variance for 10 trees: 0.0004568148

## Overall mean for 100 trees: 0.006414

## Overall variance for 100 trees: 6.676737e-05
```

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.



```
## Overall mean for 1 tree: 0.251183

## Overall variance for 1 tree: 0.01325594

## Overall mean for 10 trees: 0.122827

## Overall variance for 10 trees: 0.00320077

## Overall mean for 100 trees: 0.076979

## Overall variance for 100 trees: 0.001413212
```

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 decreases as the number of trees in the random forest grows. This is because as the number of tree increases, the “wisdom of crowd” effect intensifies

– 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.

We still get better performances because as the number of trees increases, we reduce the chances of overfitting by having many different trees that ignores the random variation in the training data and also reduces the variance of the ensemble model.

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

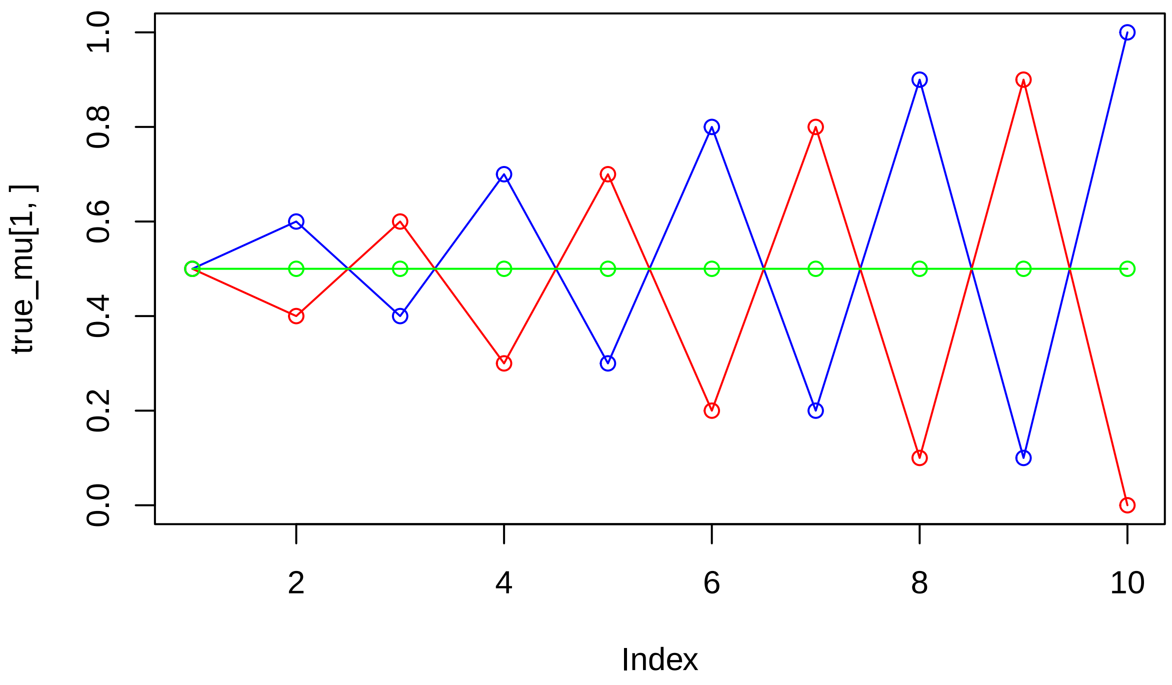
$$Bern(x|\mu_m) = \prod_{d=1}^{D} \mu_{m,d}^{x_d}(1 - \mu_{m,d})^{(1-x_d)}$$

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

$$\sum_{i=1}^{M} \log p(x_i)$$

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).

Defining true parameter values



Implementing EM algorithm

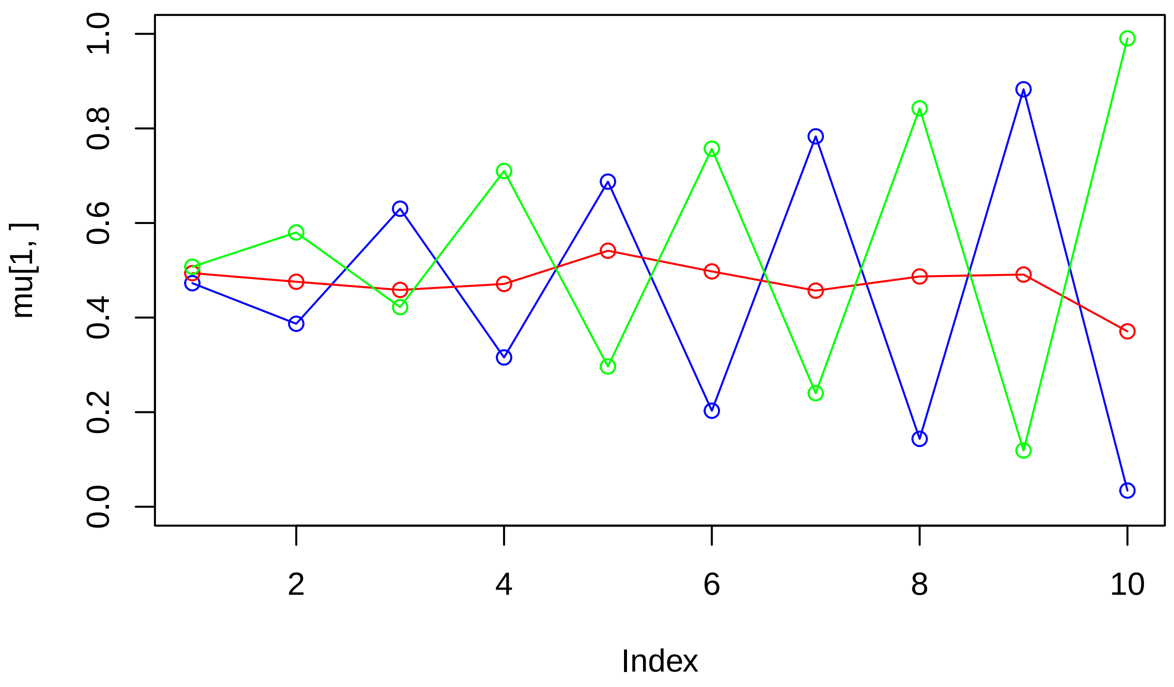
```
## [1] 0.3326090 0.3336558 0.3337352
```

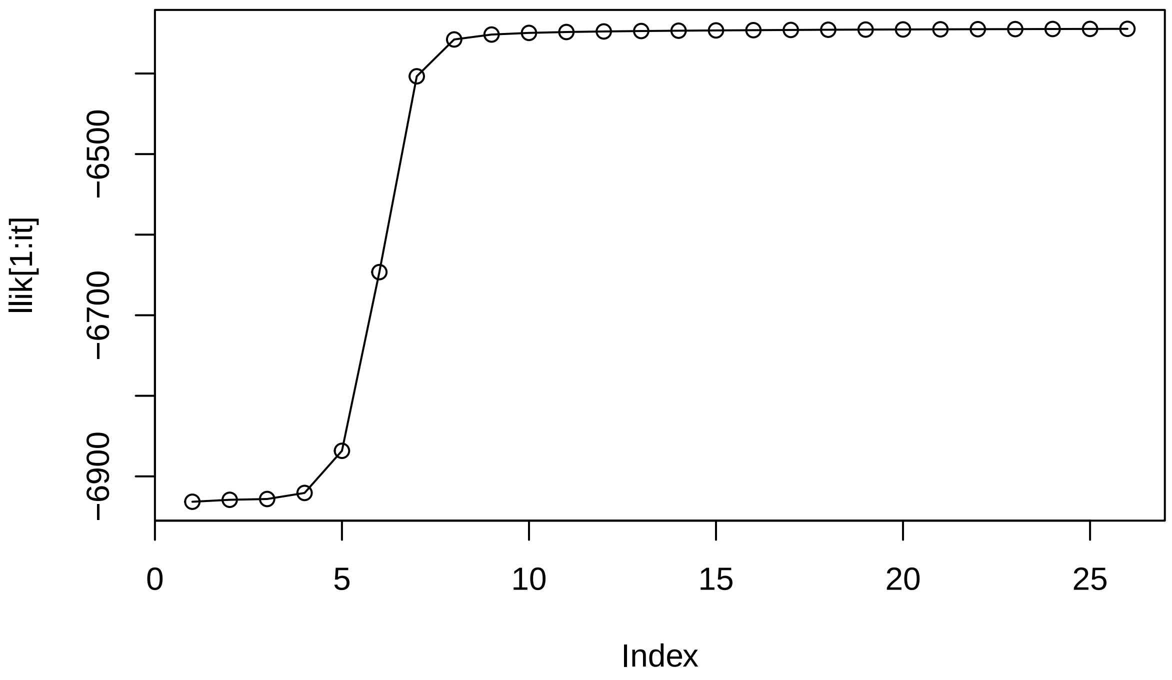
```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4939877 0.4935375 0.5042511 0.5040286 0.4987810 0.5012754 0.4971036
## [2,] 0.4993719 0.5088453 0.5068730 0.5016720 0.4929275 0.5077146 0.5095075
## [3,] 0.4975302 0.5077926 0.4939841 0.5059821 0.5063490 0.5041462 0.4929400
##           [,8]      [,9]     [,10]
## [1,] 0.4982144 0.4987654 0.4929075
## [2,] 0.4924574 0.4992470 0.5008651
## [3,] 0.4992362 0.4943482 0.4903974
```

```
## iteration:  1 log likelihood:  -6931.482
## iteration:  2 log likelihood:  -6929.074
## iteration:  3 log likelihood:  -6928.081
## iteration:  4 log likelihood:  -6920.57
## iteration:  5 log likelihood:  -6868.29
## iteration:  6 log likelihood:  -6646.505
## iteration:  7 log likelihood:  -6403.476
## iteration:  8 log likelihood:  -6357.743
## iteration:  9 log likelihood:  -6351.637
## iteration:  10 log likelihood:  -6349.59
## iteration:  11 log likelihood:  -6348.513
## iteration:  12 log likelihood:  -6347.809
## iteration:  13 log likelihood:  -6347.284
## iteration:  14 log likelihood:  -6346.861
## iteration:  15 log likelihood:  -6346.506
## iteration:  16 log likelihood:  -6346.2
## iteration:  17 log likelihood:  -6345.934
## iteration:  18 log likelihood:  -6345.699
## iteration:  19 log likelihood:  -6345.492
## iteration:  20 log likelihood:  -6345.309
## iteration:  21 log likelihood:  -6345.147
## iteration:  22 log likelihood:  -6345.003
## iteration:  23 log likelihood:  -6344.875
## iteration:  24 log likelihood:  -6344.762
## iteration:  25 log likelihood:  -6344.66
## iteration:  26 log likelihood:  -6344.57
```

```
## [1] 0.3416794 0.2690298 0.3892909
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4727544 0.3869396 0.6302224 0.3156325 0.6875038 0.2030173 0.7832090
## [2,] 0.4939501 0.4757687 0.4584644 0.4711358 0.5413928 0.4976325 0.4569664
## [3,] 0.5075441 0.5800156 0.4221148 0.7100227 0.2965478 0.7571593 0.2400675
##           [,8]      [,9]      [,10]
## [1,] 0.1435650 0.8827796 0.03422816
## [2,] 0.4869015 0.4909904 0.37087402
## [3,] 0.8424441 0.1188864 0.99033611
```


mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



Testing for different/'wrong' values of M

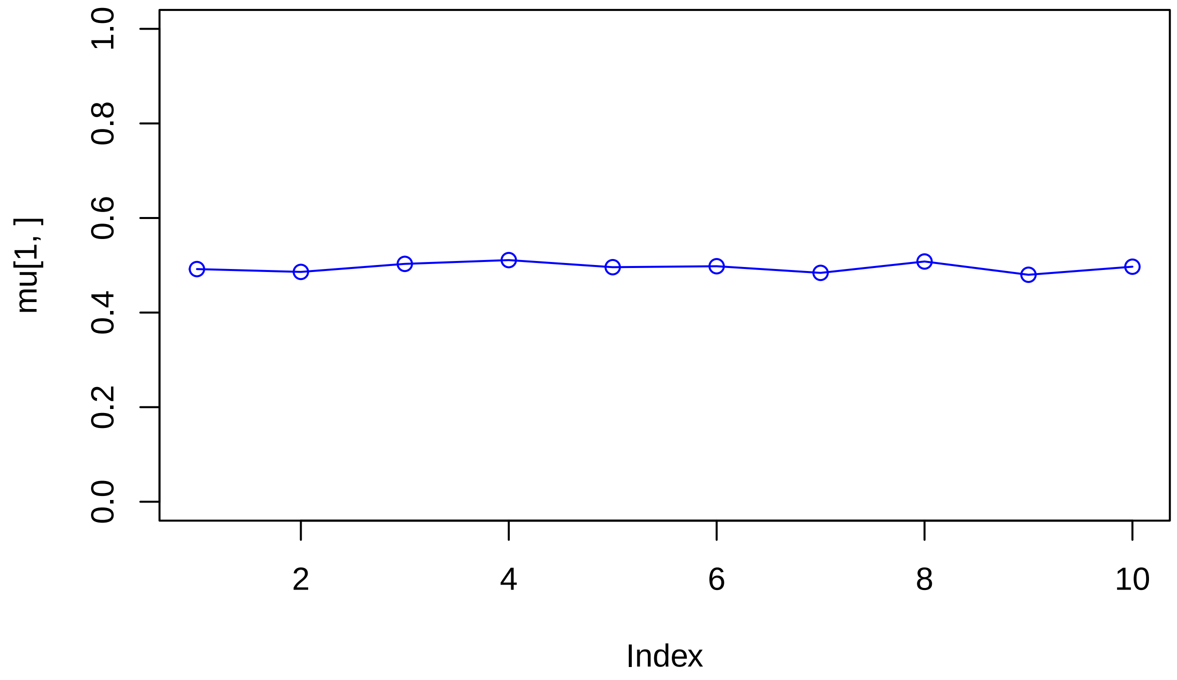
M = 1

```
## [1] 1
```

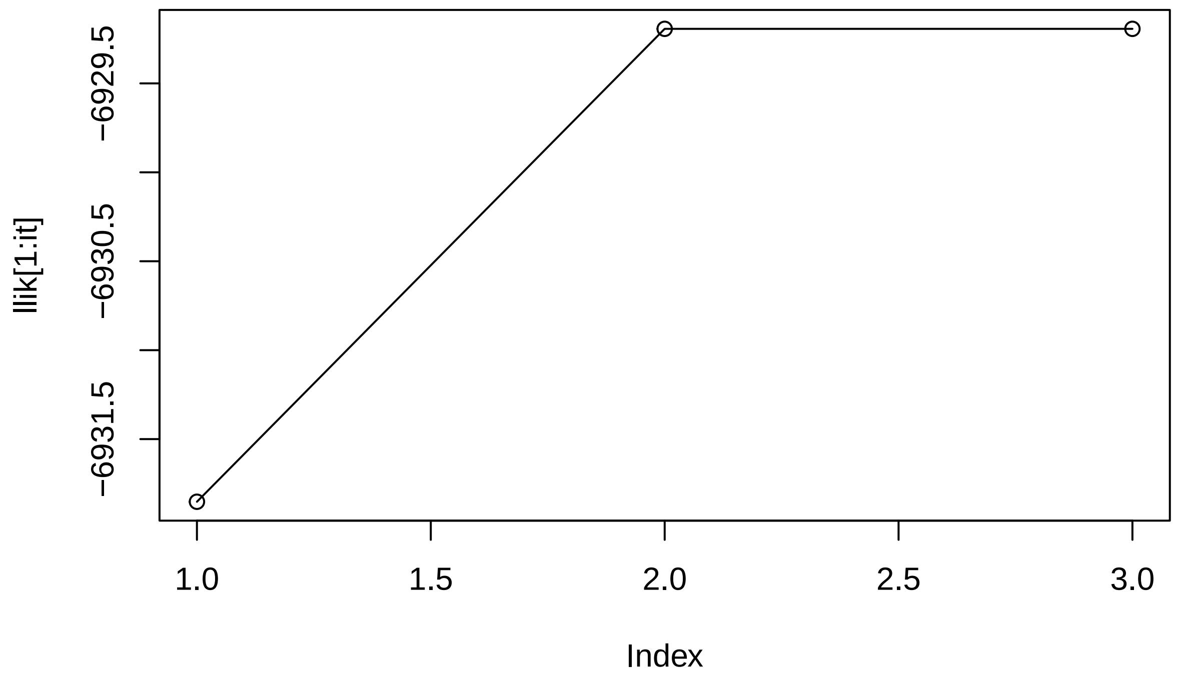
```
##          [,1]      [,2]      [,3]      [,4]     [,5]      [,6]      [,7]
## [1,] 0.490932 0.4982342 0.4961948 0.4967226 0.498422 0.4960055 0.4997165
##           [,8]      [,9]    [,10]
## [1,] 0.5090205 0.5057927 0.494766
```

```
## [1] 1
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8] [,9] [,10]
## [1,] 0.492 0.486 0.503 0.511 0.496 0.498 0.484 0.508 0.48 0.497
```



mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



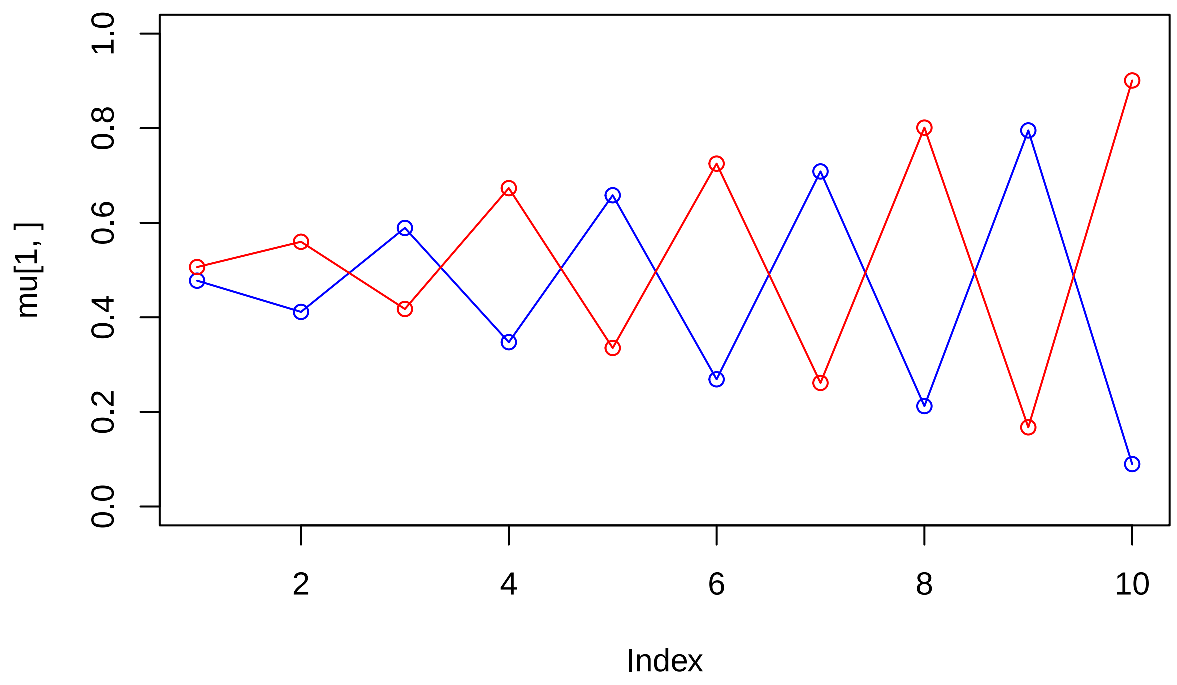
M = 2

```
## [1] 0.504519 0.495481
```

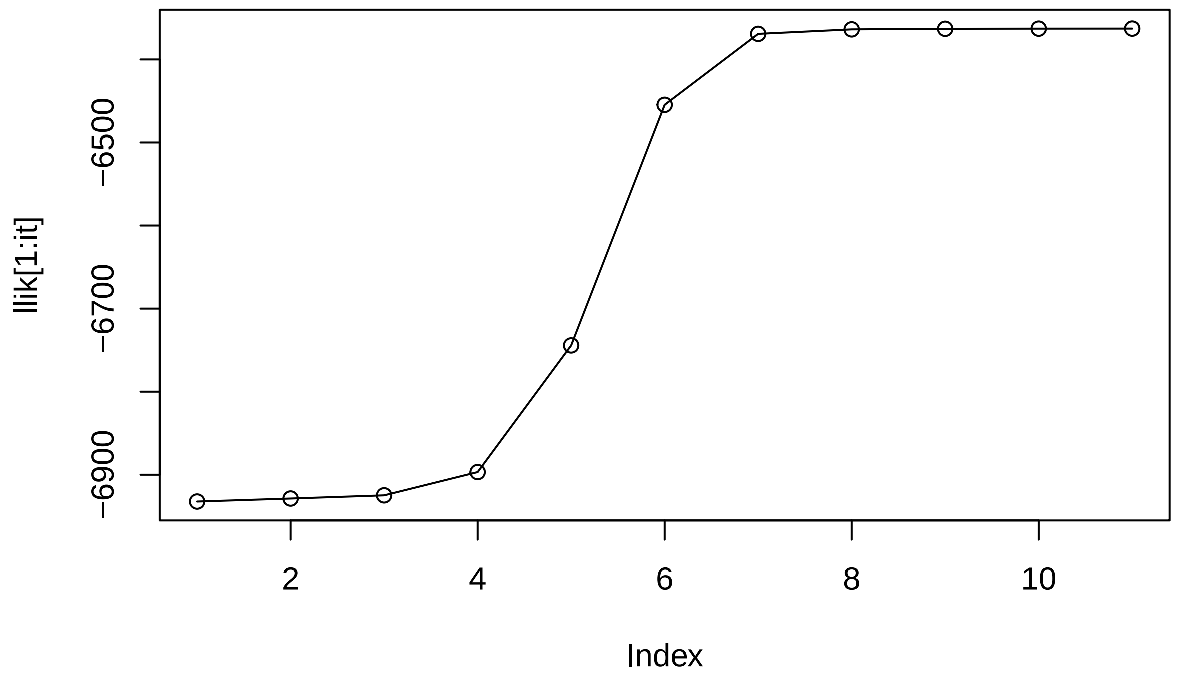
```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.5003773 0.5053941 0.4988233 0.4936057 0.4994058 0.4904128 0.5035921
## [2,] 0.4937989 0.5069180 0.5011052 0.4943551 0.5036051 0.4993455 0.5012054
##           [,8]      [,9]     [,10]
## [1,] 0.4910555 0.4947250 0.4904228
## [2,] 0.5004571 0.4987214 0.4989227
```

```
## [1] 0.4978461 0.5021539
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4775664 0.4115022 0.5890433 0.3474238 0.6581715 0.2689864 0.7085617
## [2,] 0.5063098 0.5598587 0.4176949 0.6731730 0.3352197 0.7250490 0.2613647
##           [,8]      [,9]      [,10]
## [1,] 0.2122779 0.7952892 0.08963517
## [2,] 0.8011852 0.1674155 0.90087025
```



mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



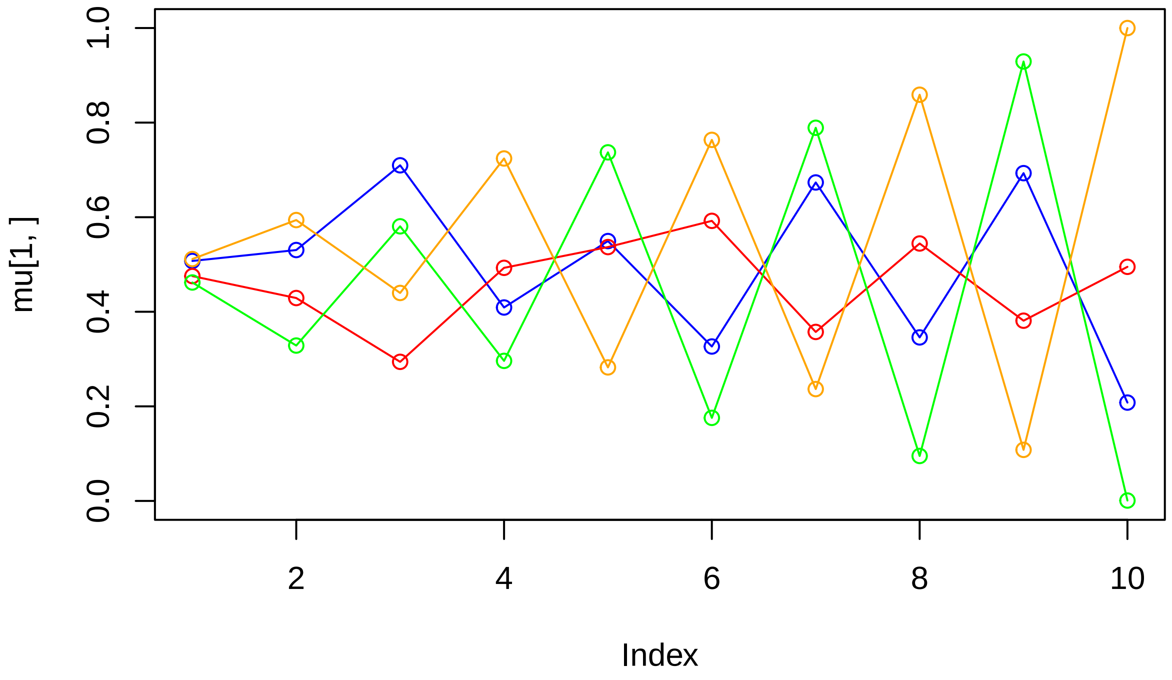
M = 4

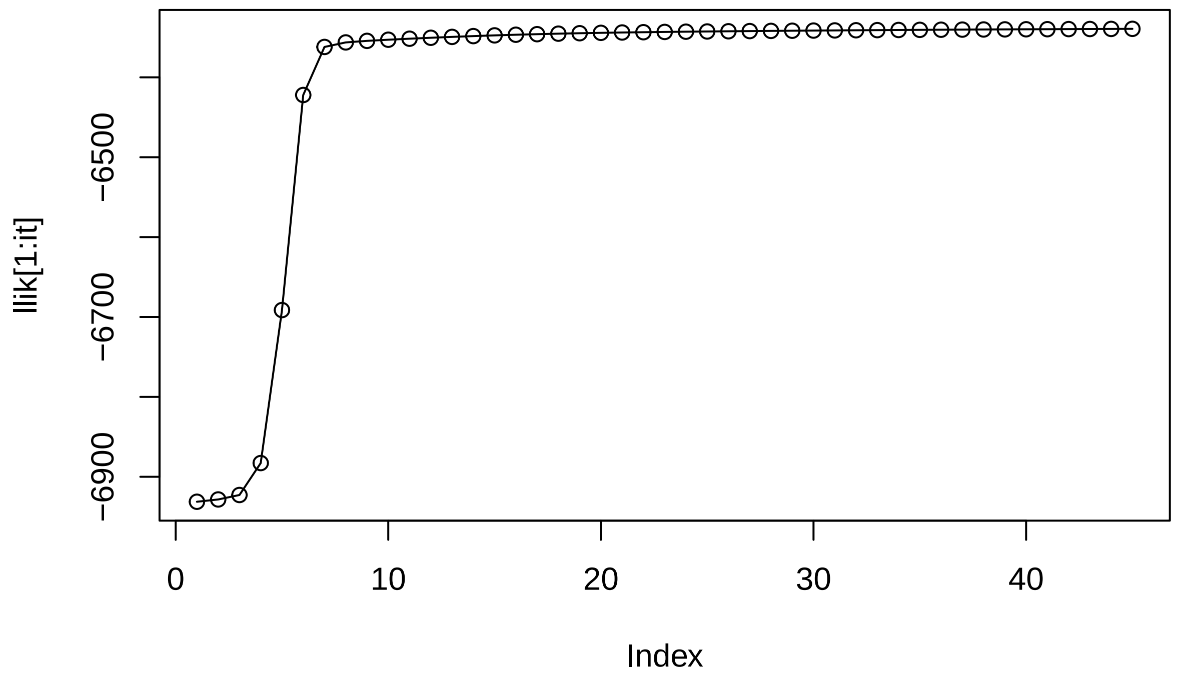
```
## [1] 0.2484501 0.2502980 0.2476268 0.2536251
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.5024204 0.5022187 0.4978065 0.5042903 0.5028750 0.4925821 0.5086689
## [2,] 0.4908125 0.4906579 0.4948314 0.4957019 0.4988231 0.4954166 0.4925546
## [3,] 0.4946235 0.5090379 0.5002896 0.5015360 0.4993958 0.4920966 0.4947109
## [4,] 0.4981094 0.4939862 0.4915595 0.5038859 0.4937181 0.5089298 0.5000765
##           [,8]      [,9]     [,10]
## [1,] 0.4947591 0.4942368 0.4925133
## [2,] 0.5016196 0.4978594 0.4970116
## [3,] 0.4902704 0.5019741 0.4904068
## [4,] 0.4943875 0.4979704 0.4997403
```

```
## [1] 0.2147179 0.1921974 0.2360570 0.3570278
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.5074175 0.5305526 0.7097004 0.4090309 0.5495300 0.3265539 0.6734651
## [2,] 0.4754629 0.4290590 0.2941442 0.4928757 0.5367959 0.5922686 0.3573383
## [3,] 0.4619640 0.3285627 0.5806695 0.2961395 0.7371512 0.1756462 0.7890675
## [4,] 0.5114892 0.5939519 0.4397692 0.7241412 0.2824028 0.7634926 0.2365379
##            [,8]      [,9]        [,10]
## [1,] 0.34577189 0.6931538 0.2079360909
## [2,] 0.54438008 0.3811067 0.4950318242
## [3,] 0.09506875 0.9292759 0.0008489173
## [4,] 0.85899911 0.1079964 0.9999451847
```





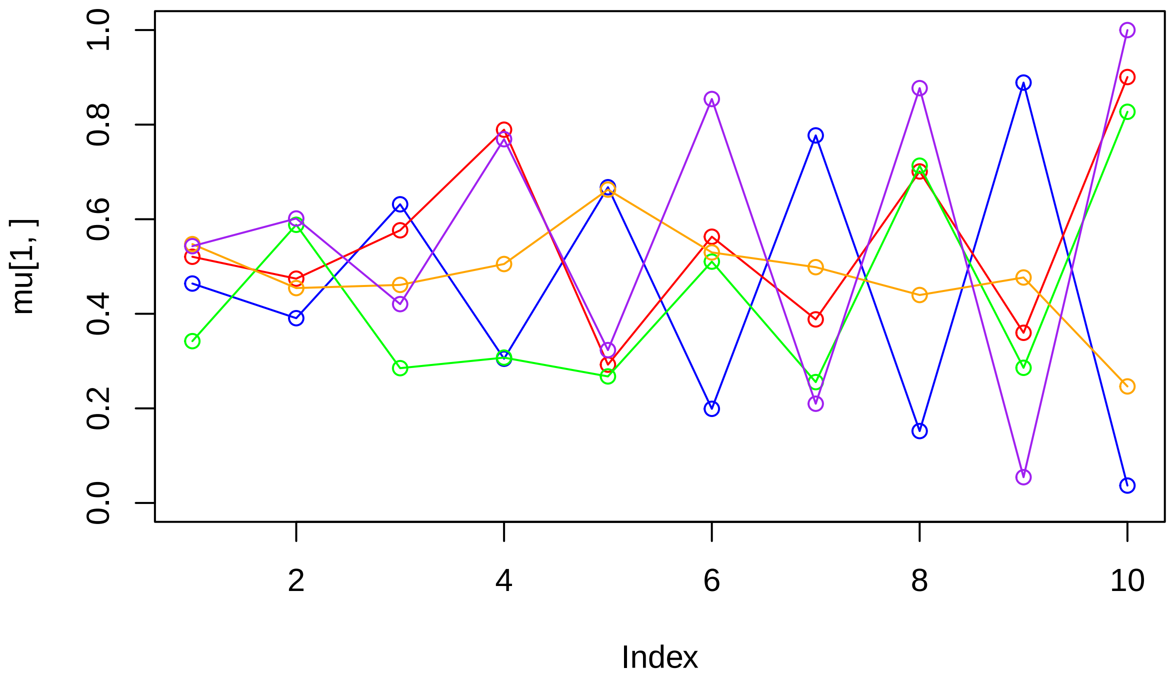
M = 5

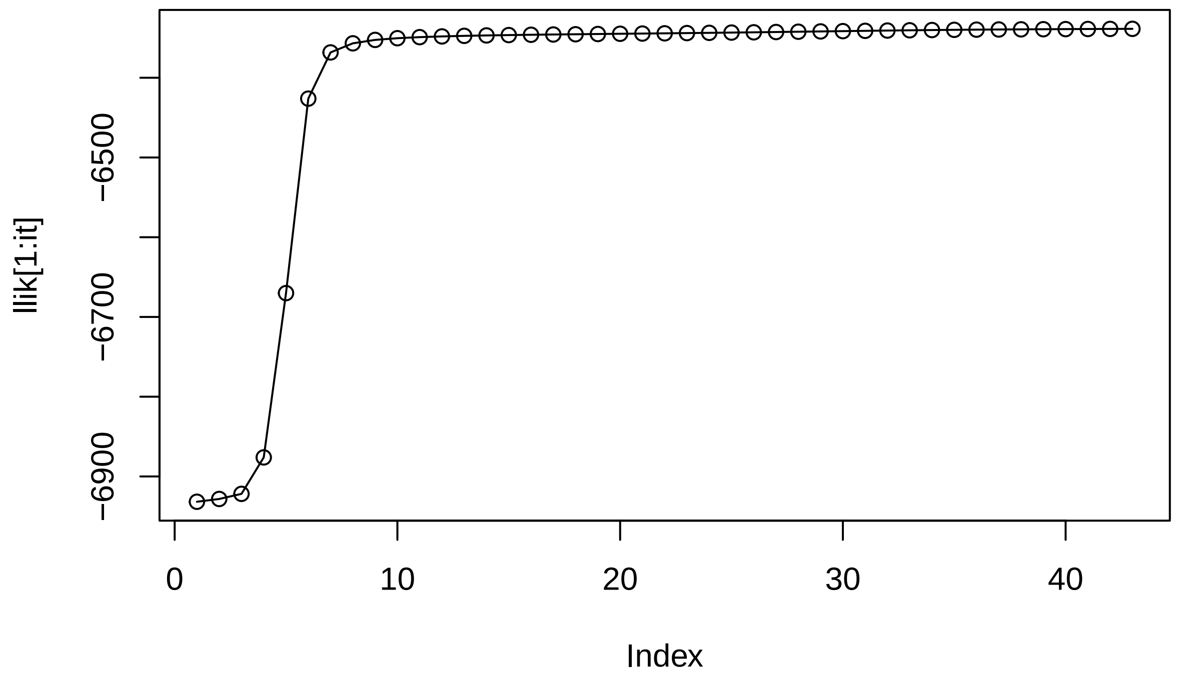
```
## [1] 0.2025189 0.2016880 0.1976318 0.1975042 0.2006570
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4901300 0.4928976 0.5071124 0.4936038 0.4989502 0.4912528 0.5007583
## [2,] 0.5089538 0.5055172 0.5006660 0.5083564 0.5003053 0.5020466 0.5057604
## [3,] 0.5019854 0.4950438 0.4920663 0.4951639 0.4969707 0.4964869 0.4966188
## [4,] 0.4903284 0.5041935 0.5002655 0.4921470 0.4951467 0.4979475 0.5003055
## [5,] 0.4960426 0.5089882 0.4956366 0.5031586 0.4966150 0.4908961 0.4958781
##           [,8]      [,9]     [,10]
## [1,] 0.4967445 0.5021727 0.4921613
## [2,] 0.4997745 0.5096218 0.5068372
## [3,] 0.5086394 0.5004841 0.4945962
## [4,] 0.5008179 0.5056923 0.4994151
## [5,] 0.5053859 0.5011826 0.5004403
```

```
## [1] 0.3453505 0.1085672 0.1100279 0.1864872 0.2495671
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4639799 0.3907129 0.6315776 0.3049526 0.6676564 0.1990694 0.7771917
## [2,] 0.5206037 0.4742815 0.5765919 0.7892672 0.2924454 0.5629881 0.3881788
## [3,] 0.3421747 0.5881709 0.2850274 0.3072872 0.2675629 0.5103402 0.2555409
## [4,] 0.5473384 0.4543585 0.4610819 0.5052458 0.6626068 0.5297721 0.4985891
## [5,] 0.5430340 0.6015552 0.4204822 0.7691873 0.3232295 0.8542064 0.2097865
##            [,8]       [,9]      [,10]
## [1,] 0.1520222 0.88892705 0.03673945
## [2,] 0.7009848 0.35981154 0.90063244
## [3,] 0.7132116 0.28574686 0.82720414
## [4,] 0.4397057 0.47677217 0.24641433
## [5,] 0.8772086 0.05446547 0.99998813
```





Analysis

- Having too few clusters will make it approach the mean of the true distributions for each dimension.
- Having too many clusters will do the same thing, but with redundancy (overlapping curves).
- Investigate M = 6 to confirm overlapping, and how pi and mu values change with this.

Appendix

```r
knitr::opts_chunk$set(echo = FALSE)
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)
```

```r
plot(x1,x2,col=(y+1))

get_misc_err <- function(conf_mat){
  #Given a confusion matrix, return the misclassification error
  (sum(conf_mat)-sum(diag(conf_mat)))/sum(conf_mat)
}

r1 <- randomForest(trdata, trlabels, ntree = 1, nodesize = 25, keep.forest = TRUE)
p1 <- predict(r1, tedata)
conf_mat1 <- table(p1, telabels)
#get_misc_err(conf_mat1)
cat("Mean misclassification error for 1 tree:", get_misc_err(conf_mat1), "\n")

r10 <- randomForest(trdata, trlabels, ntree = 10, nodesize = 25, keep.forest = TRUE)
p10 <- predict(r10, tedata)
conf_mat10 <- table(p10, telabels)
#get_misc_err(conf_mat10)
cat("Mean misclassification error for 10 trees:", get_misc_err(conf_mat10), "\n")

r100 <- randomForest(trdata, trlabels, ntree = 100, nodesize = 25, keep.forest = TRUE)
p100 <- predict(r100, tedata)
conf_mat100 <- table(p100, telabels)
#get_misc_err(conf_mat100)
cat("Mean misclassification error for 100 trees:", get_misc_err(conf_mat100), "\n")



get_training <- function(){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<x2)
  trlabels<-as.factor(y)
  return(list(trdata, trlabels))
}
meR1 <- c()

meR10 <- c()

meR100 <- c()

for(i in 1:1000){

  #cat("Iteration:", i, "\r")
  #Creating the training data
  training <- get_training()
  trdata <- training[[1]]
  trlabels <- training[[2]]


  r1 <- randomForest(trdata, trlabels, ntree = 1, nodesize = 25, keep.forest = TRUE)
  p1 <- predict(r1, tedata)
  conf_mat1 <- table(p1, telabels)
  meR1 <- c(meR1, get_misc_err(conf_mat1))

  r10 <- randomForest(trdata, trlabels, ntree = 10, nodesize = 25, keep.forest = TRUE)
  p10 <- predict(r10, tedata)
  conf_mat10 <- table(p10, telabels)
  meR10 <- c(meR10, get_misc_err(conf_mat10))

  r100 <- randomForest(trdata, trlabels, ntree = 100, nodesize = 25, keep.forest = TRUE)
  p100 <- predict(r100, tedata)
  conf_mat100 <- table(p100, telabels)
  meR100 <- c(meR100, get_misc_err(conf_mat100))
}

#Overall mean
cat("Overall mean for 1 tree:", mean(meR1), "\n")
cat("Overall variance for 1 tree:", var(meR1), "\n")
cat("Overall mean for 10 trees:", mean(meR10), "\n")
cat("Overall variance for 10 trees:", var(meR10), "\n")
cat("Overall mean for 100 trees:", mean(meR100), "\n")
cat("Overall variance for 100 trees:", var(meR100), "\n")

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

get_training <- function(){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<0.5)
  trlabels<-as.factor(y)
  return(list(trdata, trlabels))
}
meR1 <- c()
meR10 <- c()
meR100 <- c()

for(i in 1:1000){
  #cat("Iteration:", i, "\r")
  #Creating the training data
  training <- get_training()
  trdata <- training[[1]]
  trlabels <- training[[2]]

  r1 <- randomForest(trdata, trlabels, ntree = 1, nodesize = 25, keep.forest = TRUE)
  p1 <- predict(r1, tedata)
  conf_mat1 <- table(p1, telabels)
  meR1 <- c(meR1, get_misc_err(conf_mat1))
```

```r
  r10 <- randomForest(trdata, trlabels, ntree = 10, nodesize = 25, keep.forest = TRUE)
  p10 <- predict(r10, tedata)
  conf_mat10 <- table(p10, telabels)
  meR10 <- c(meR10, get_misc_err(conf_mat10))

  r100 <- randomForest(trdata, trlabels, ntree = 100, nodesize = 25, keep.forest = TRUE)
  p100 <- predict(r100, tedata)
  conf_mat100 <- table(p100, telabels)
  meR100 <- c(meR100, get_misc_err(conf_mat100))
}

#Overall means and variances
cat("Overall mean for 1 tree:", mean(meR1), "\n")
cat("Overall variance for 1 tree:", var(meR1), "\n")
cat("Overall mean for 10 trees:", mean(meR10), "\n")
cat("Overall variance for 10 trees:", var(meR10), "\n")
cat("Overall mean for 100 trees:", mean(meR100), "\n")
cat("Overall variance for 100 trees:", var(meR100), "\n")
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))

get_training <- function(){
  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)
  return(list(trdata, trlabels))
}
meR1 <- c()
meR10 <- c()
meR100 <- c()

for(i in 1:1000){
  #cat("Iteration:", i, "\r")
  #Creating the training data
  training <- get_training()
  trdata <- training[[1]]
  trlabels <- training[[2]]


  r1 <- randomForest(trdata, trlabels, ntree = 1, nodesize = 12, keep.forest = TRUE)
  p1 <- predict(r1, tedata)
  conf_mat1 <- table(p1, telabels)
  meR1 <- c(meR1, get_misc_err(conf_mat1))

  r10 <- randomForest(trdata, trlabels, ntree = 10, nodesize = 12, keep.forest = TRUE)
  p10 <- predict(r10, tedata)
  conf_mat10 <- table(p10, telabels)
  meR10 <- c(meR10, get_misc_err(conf_mat10))

  r100 <- randomForest(trdata, trlabels, ntree = 100, nodesize = 12, keep.forest = TRUE)
  p100 <- predict(r100, tedata)
  conf_mat100 <- table(p100, telabels)
  meR100 <- c(meR100, get_misc_err(conf_mat100))
}

#Overall mean
cat("Overall mean for 1 tree:", mean(meR1), "\n")
cat("Overall variance for 1 tree:", var(meR1), "\n")
cat("Overall mean for 10 trees:", mean(meR10), "\n")
cat("Overall variance for 10 trees:", var(meR10), "\n")
cat("Overall mean for 100 trees:", mean(meR100), "\n")
cat("Overall variance for 100 trees:", var(meR100), "\n")
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")

# 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)
```

```r
}
pi
mu

for(it in 1:max_it) {
  # plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  # points(mu[2,], type="o", col="red")
  # points(mu[3,], type="o", col="green")
  # # Sys.sleep(0.5)

  # E-step: Computation of the weights
  # Compute x given y=m probabilities
  bern <- matrix(nrow=n, ncol=M)
  for (i in 1:n) {
    for (m in 1:M) {
      v_bern <- vector()
      for (d in 1:D) {
          bern_im <- mu[m, d] ^ x[i, d] * (1 - mu[m, d]) ^ (1 - x[i, d])
          v_bern <- append(v_bern, bern_im)
      }
      bern[i, m] <- pi[m] * prod(v_bern)
    }
  }

  # Compute weights according to P(y=m|x) = P(x|y=m)P(y=m) / P(x)
  for (i in 1:n) {
    for (m in 1:M) {
      w[i, m] <- bern[i, m] / sum(bern[i,])
    }
  }

  #Log likelihood computation.
  llik[it] <- sum(log(rowSums(bern)))

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the log likelihood has not changed significantly
  # Your code here
  if (length(llik[llik != 0]) >= 2) {
    if (abs(llik[it] - llik[it - 1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and weights
  # Compute new pi and mu from weights
  pi <- 1 / n * colSums(w)

  for (m in 1:M) {
    mu[m,] <- 1 / sum(w[, m]) * colSums(w[, m] * x)
  }
}
```

```r
pi
mu

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")

plot(llik[1:it], type="o")

M=1 # 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)
}
pi
mu

for(it in 1:max_it) {
  # plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  # points(mu[2,], type="o", col="red")
  # points(mu[3,], type="o", col="green")
  # # Sys.sleep(0.5)

  # E-step: Computation of the weights
  # Compute x given y=m probabilities
  bern <- matrix(nrow=n, ncol=M)
  for (i in 1:n) {
    for (m in 1:M) {
      v_bern <- vector()
      for (d in 1:D) {
          bern_im <- mu[m, d] ^ x[i, d] * (1 - mu[m, d]) ^ (1 - x[i, d])
          v_bern <- append(v_bern, bern_im)
      }
      bern[i, m] <- pi[m] * prod(v_bern)
    }
  }

  # Compute weights according to P(y=m|x) = P(x|y=m)P(y=m) / P(x)
  for (i in 1:n) {
    for (m in 1:M) {
      w[i, m] <- bern[i, m] / sum(bern[i,])
    }
  }

  #Log likelihood computation.
  llik[it] <- sum(log(rowSums(bern)))
  # cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the log likelihood has not changed significantly
  # Your code here
  if (length(llik[llik != 0]) >= 2) {
    if (abs(llik[it] - llik[it - 1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and weights
  # Compute new pi and mu from weights
  pi <- 1 / n * colSums(w)

  for (m in 1:M) {
    mu[m,] <- 1 / sum(w[, m]) * colSums(w[, m] * x)
  }
}

pi
mu

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="orange")

plot(llik[1:it], type="o")

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)
}
pi
mu

for(it in 1:max_it) {
  # plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  # points(mu[2,], type="o", col="red")
  # points(mu[3,], type="o", col="green")
  # # Sys.sleep(0.5)

  # E-step: Computation of the weights
  # Compute x given y=m probabilities
  bern <- matrix(nrow=n, ncol=M)
  for (i in 1:n) {
    for (m in 1:M) {
      v_bern <- vector()
      for (d in 1:D) {
          bern_im <- mu[m, d] ^ x[i, d] * (1 - mu[m, d]) ^ (1 - x[i, d])
          v_bern <- append(v_bern, bern_im)
      }
      bern[i, m] <- pi[m] * prod(v_bern)
    }
  }

  # Compute weights according to P(y=m|x) = P(x|y=m)P(y=m) / P(x)
  for (i in 1:n) {
    for (m in 1:M) {
      w[i, m] <- bern[i, m] / sum(bern[i,])
    }
  }

  #Log likelihood computation.
  llik[it] <- sum(log(rowSums(bern)))

  # cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the log likelihood has not changed significantly
  # Your code here
  if (length(llik[llik != 0]) >= 2) {
    if (abs(llik[it] - llik[it - 1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and weights
  # Compute new pi and mu from weights
  pi <- 1 / n * colSums(w)

  for (m in 1:M) {
    mu[m,] <- 1 / sum(w[, m]) * colSums(w[, m] * x)
  }
}

pi
mu

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

plot(llik[1:it], type="o")

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
```

```r
# 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)
}
pi
mu

for(it in 1:max_it) {
  # plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  # points(mu[2,], type="o", col="red")
  # points(mu[3,], type="o", col="green")
  # # Sys.sleep(0.5)

  # E-step: Computation of the weights
  # Compute x given y=m probabilities
  bern <- matrix(nrow=n, ncol=M)
  for (i in 1:n) {
    for (m in 1:M) {
      v_bern <- vector()
      for (d in 1:D) {
          bern_im <- mu[m, d] ^ x[i, d] * (1 - mu[m, d]) ^ (1 - x[i, d])
          v_bern <- append(v_bern, bern_im)
      }
      bern[i, m] <- pi[m] * prod(v_bern)
    }
  }

  # Compute weights according to P(y=m|x) = P(x|y=m)P(y=m) / P(x)
  for (i in 1:n) {
    for (m in 1:M) {
      w[i, m] <- bern[i, m] / sum(bern[i,])
    }
  }

  #Log likelihood computation.
  llik[it] <- sum(log(rowSums(bern)))

  # cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the log likelihood has not changed significantly
  # Your code here
  if (length(llik[llik != 0]) >= 2) {
    if (abs(llik[it] - llik[it - 1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and weights
  # Compute new pi and mu from weights
  pi <- 1 / n * colSums(w)
  for (m in 1:M) {
    mu[m,] <- 1 / sum(w[, m]) * colSums(w[, m] * x)
  }
}

pi
mu

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="orange")

plot(llik[1:it], type="o")

M=5 # 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)
}
pi
mu

for(it in 1:max_it) {
  # plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  # points(mu[2,], type="o", col="red")
  # points(mu[3,], type="o", col="green")
  # # Sys.sleep(0.5)

  # E-step: Computation of the weights
  # Compute x given y=m probabilities
  bern <- matrix(nrow=n, ncol=M)
  for (i in 1:n) {
    for (m in 1:M) {
      v_bern <- vector()
      for (d in 1:D) {
          bern_im <- mu[m, d] ^ x[i, d] * (1 - mu[m, d]) ^ (1 - x[i, d])
          v_bern <- append(v_bern, bern_im)
      }
      bern[i, m] <- pi[m] * prod(v_bern)
    }
  }

  # Compute weights according to P(y=m|x) = P(x|y=m)P(y=m) / P(x)
  for (i in 1:n) {
    for (m in 1:M) {
      w[i, m] <- bern[i, m] / sum(bern[i,])
    }
  }

  #Log likelihood computation.
  llik[it] <- sum(log(rowSums(bern)))

  # cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the log likelihood has not changed significantly
  # Your code here
  if (length(llik[llik != 0]) >= 2) {
    if (abs(llik[it] - llik[it - 1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and weights
  # Compute new pi and mu from weights
  pi <- 1 / n * colSums(w)

  for (m in 1:M) {
    mu[m,] <- 1 / sum(w[, m]) * colSums(w[, m] * x)
  }
}

pi
mu

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="orange")
points(mu[5,], type="o", col="purple")

plot(llik[1:it], type="o")
```