Machine Learning Lab1-block2

Lepeng Zhang, Xuan Wang, Priyarani Patil

2023-12-03

Statement of Contribution

The group report was made based on the discussion after all of us had finished all three assignments. Assignment 1 was mainly contributed by Lepeng Zhang. Assignment 2 was mainly contributed by Xuan Wang. Assignment 3 was mainly contributed by Priyarani Patil.

Assignment 1. ENSEMBLE METHODS

Q1

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
```

Table 1: Classification Error

	Mean	Variance
1 tree	0.206625	0.0030445
10 trees	0.137777	0.0009647
100 trees	0.112063	0.0008307

Q2

Table 2: Classification Error

	Mean	Variance
1 tree	0.097530	0.0187001
10 trees	0.016116	0.0006983
100 trees	0.006754	0.0000764

Q3

Table 3: Classification Error

	Mean	Variance
1 tree	0.245286	0.0136981
10 trees	0.120254	0.0028291
100 trees	0.073590	0.0012035

Q4

Assignment 2. MIXTURE MODELS

Implementation explanation

Computation of the weights

According to the literature and instruction,

$$w_i(m) = p(y_i = m|x_i, \hat{\theta}) = \frac{\hat{\pi_m} Bern(x_i|\mu_m)}{\sum_{j=1}^{M} \hat{\pi}_j Bern(x_i|\mu_j)}$$

We calculate $\hat{\pi_m} Bern(x_i|\mu_m)$ using *log* operation first and then *exp* operation. The reason is that it's much more convenient to use *log* for a product term and *exp* operation is for setting it back.

$$\hat{\pi_m} Bern(x_i|\mu_m) = exp[log(\hat{\pi_m} Bern(x_i|\mu_m))] = exp[log(\hat{\pi_m}) + \sum_{d=1}^{D}(x_{i,d} log(\mu_{m,d}) + (1 - x_{i,d}) log(1 - \mu_{m,d}))]$$

$w_i(m)$ can easily be computed after getting all $\hat{\pi_m} Bern(x_i|\mu_m)$ with m from 1 to M.

Log likelihood computation

$$llik[it] = \sum_{i=1}^{n} log(p(x_i)) = \sum_{i=1}^{n} log(\sum_{m=1}^{M} \hat{\pi_m} Bern(x_i|\mu_m))$$

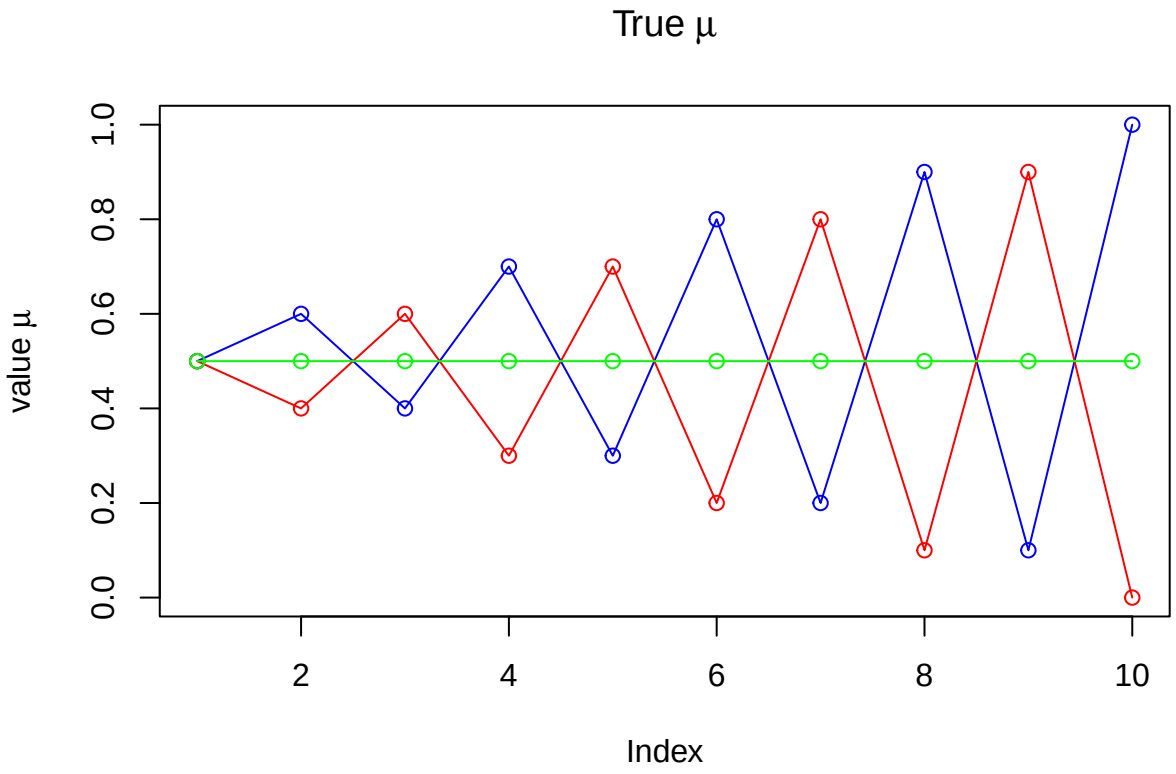
$\hat{\pi_m} Bern(x_i|\mu_m)$ has already been computed. A vector named w_sum was created to store $\sum_{m=1}^{M} \hat{\pi_m} Bern(x_i|\mu_m)$

$$w\_sum[i] = \sum_{m=1}^{M} \hat{\pi_m} Bern(x_i|\mu_m)$$

ML parameter estimation

Just follow the equations 10.16 a and b in the slide.

The true μ shows in the graph below.



when M=3

```
## [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
```

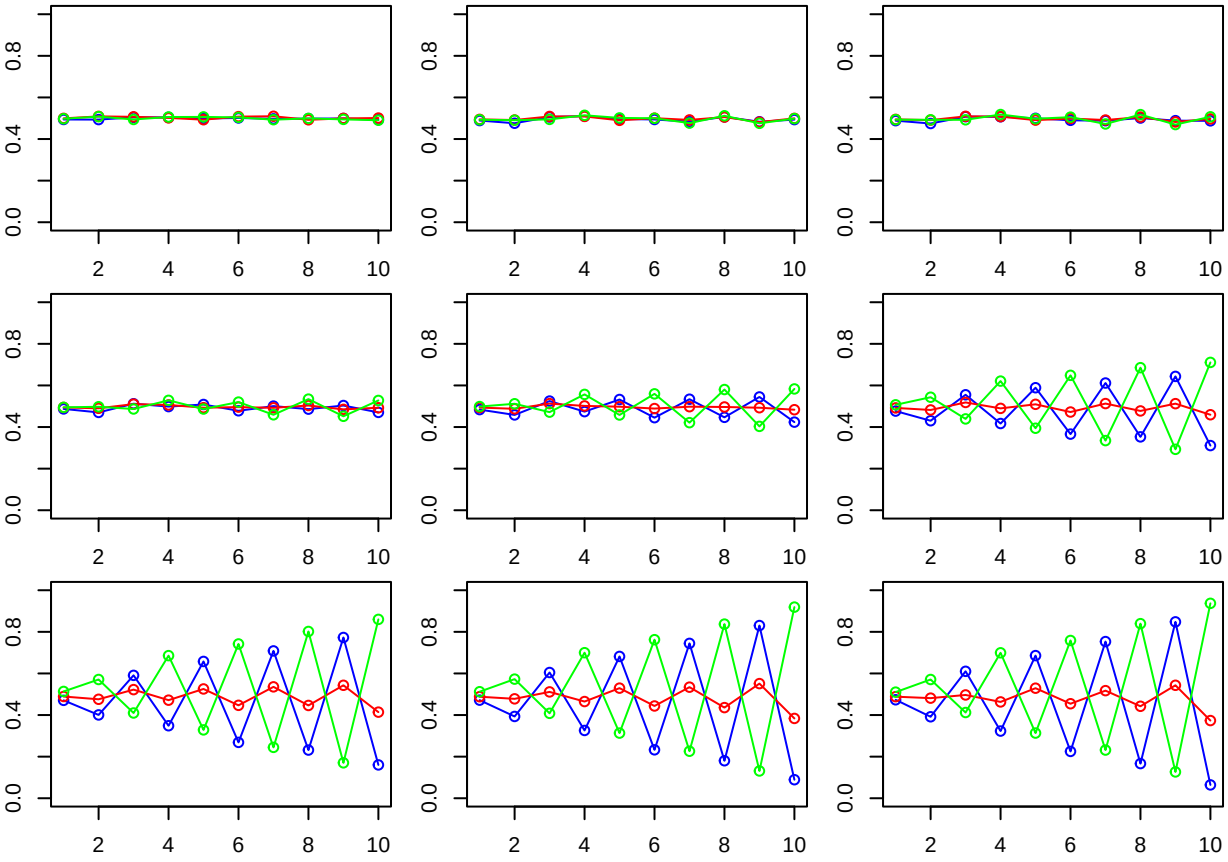
```
## iteration:  9 log likelihood:  -6351.637
```

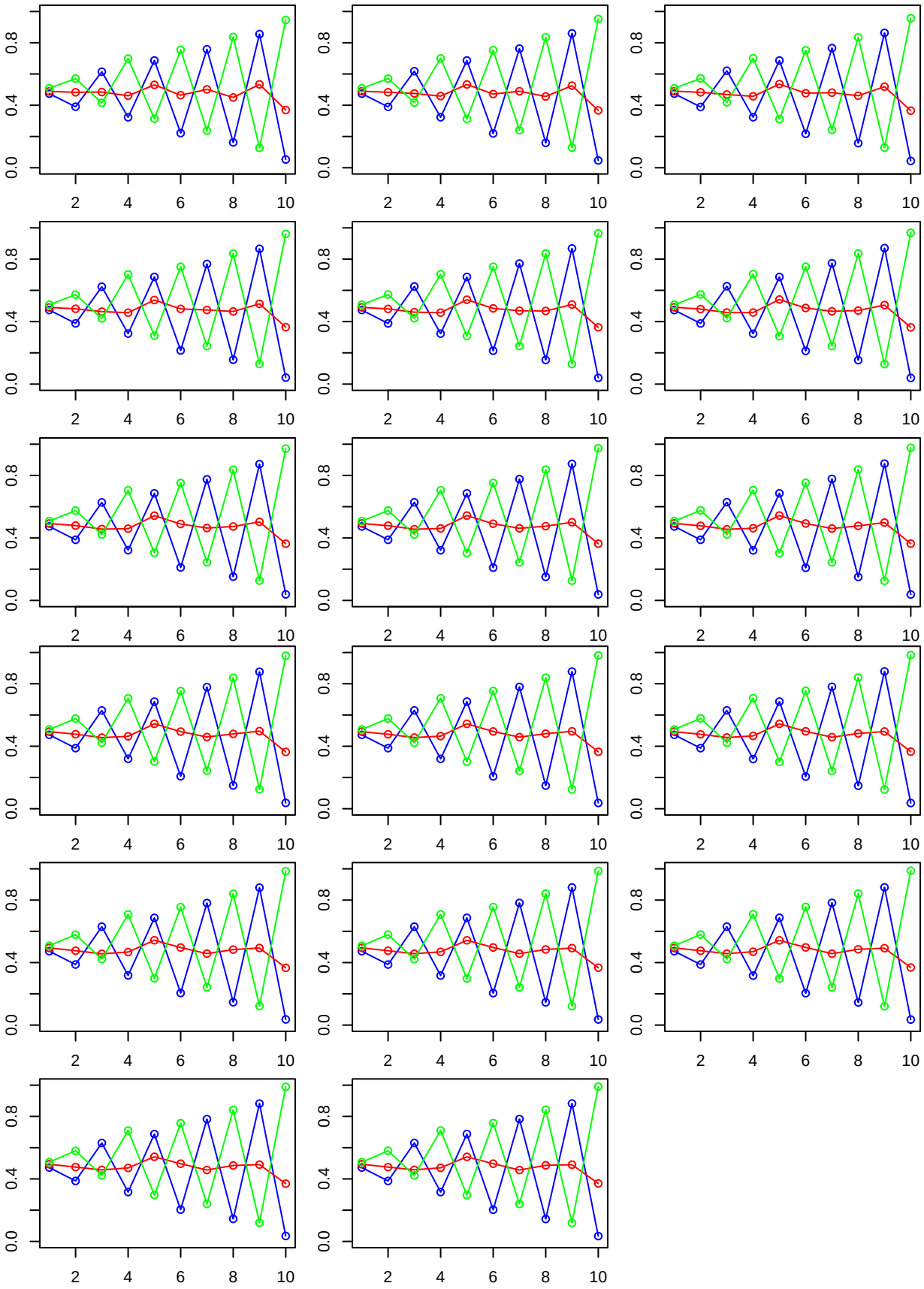
```
## iteration:  10 log likelihood:  -6349.59
```

```
## iteration:  11 log likelihood:  -6348.513
```

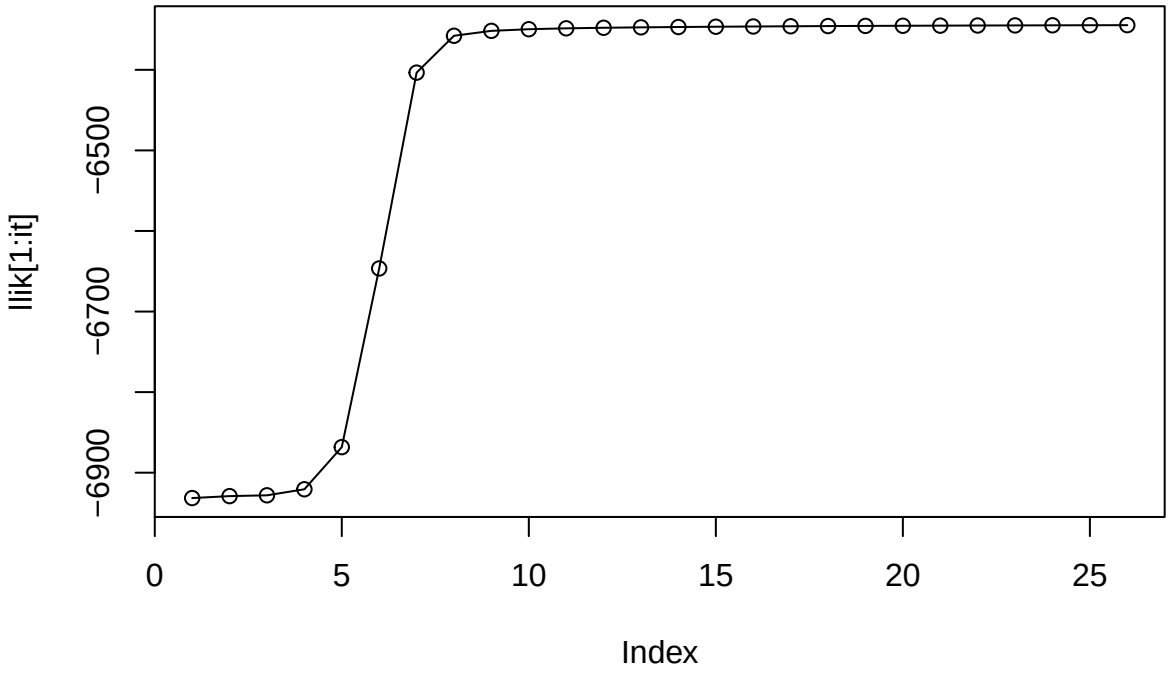
```
## iteration:  12 log likelihood:  -6347.809
```

```
## 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
```



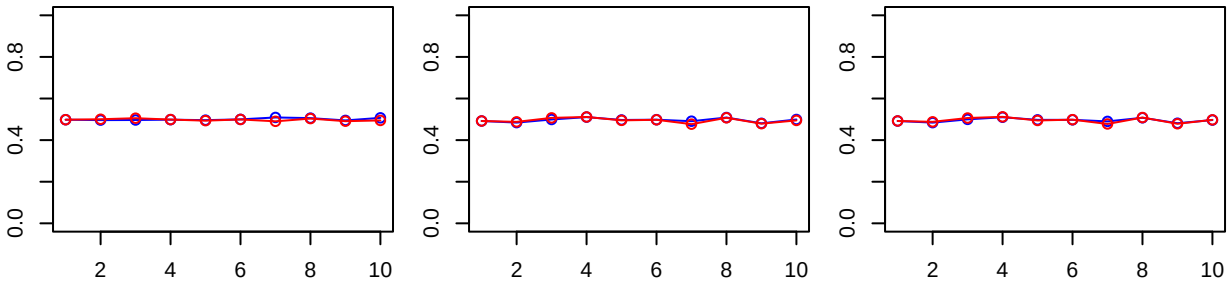

0.0
0.4
0.8
2
4
6
8
10

```
## The final pi is:
## [1] 0.3416794 0.2690298 0.3892909
## The final mu is:
##           [,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
```



when M=2

```
## [1] 0.5089877 0.4910123
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4982342 0.4961948 0.4967226 0.4984220 0.4960055 0.4997165 0.5090205
## [2,] 0.4980423 0.5003773 0.5053941 0.4988233 0.4936057 0.4994058 0.4904128
##           [,8]      [,9]    [,10]
## [1,] 0.5057927 0.4947660 0.507127
## [2,] 0.5035921 0.4910555 0.494725
## iteration:  1 log likelihood:  -6930.885
## iteration:  2 log likelihood:  -6929.223
## iteration:  3 log likelihood:  -6929.161
```

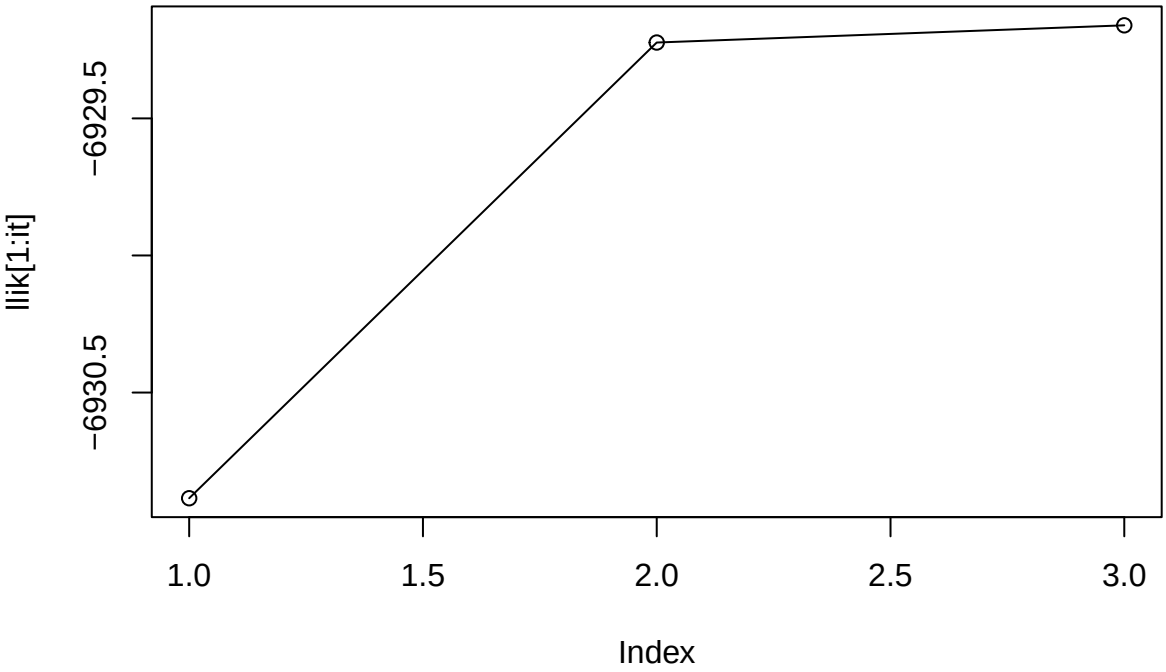



```
## The final pi is:
```

```
## [1] 0.5086338 0.4913662
```

```
## The final mu is:
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4918863 0.4840752 0.4997708 0.5100499 0.4974422 0.4976326 0.4900480
## [2,] 0.4921177 0.4879925 0.5063427 0.5119835 0.4945072 0.4983803 0.4777395
##           [,8]      [,9]     [,10]
## [1,] 0.5074063 0.4813071 0.4965251
## [2,] 0.5086146 0.4786470 0.4974916
```

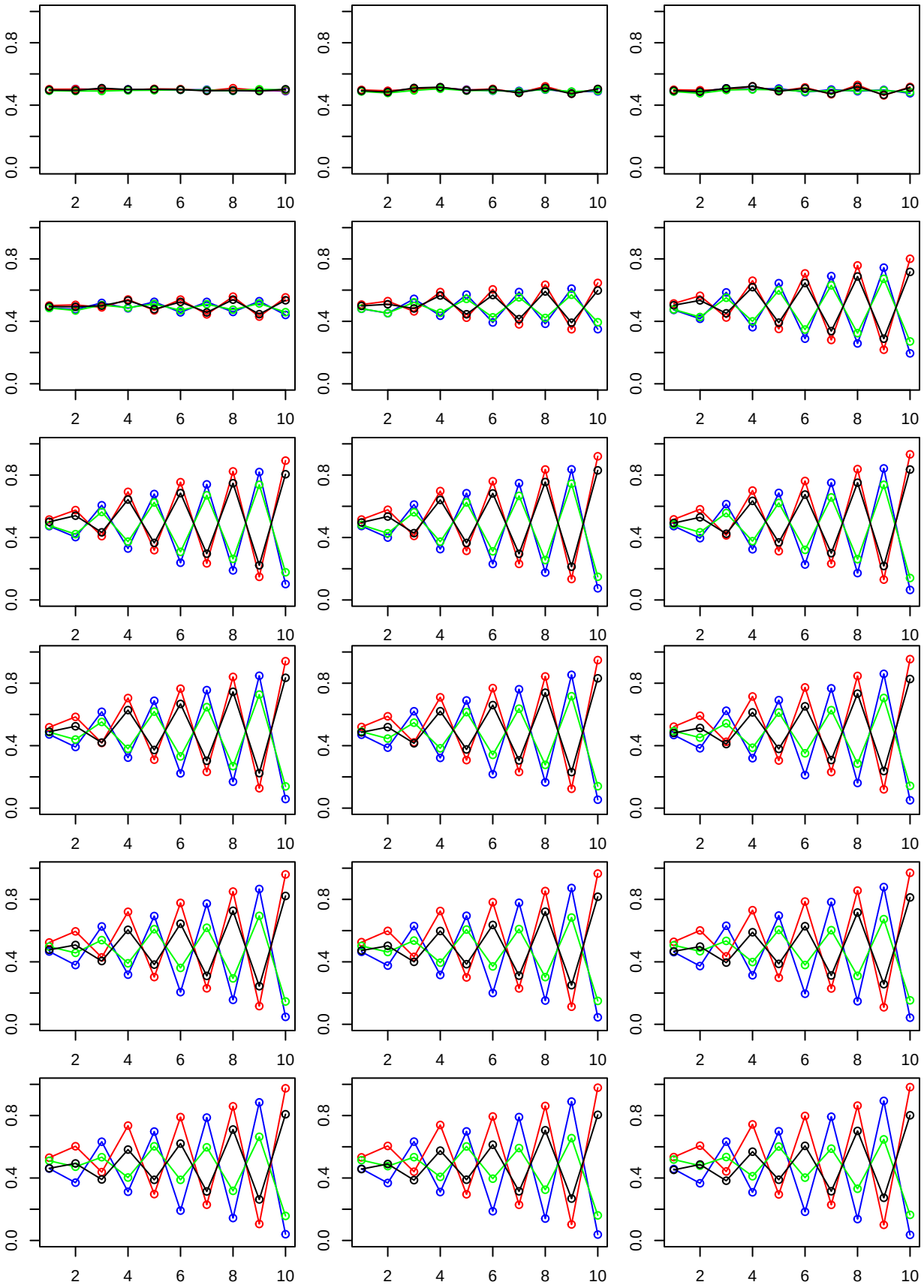


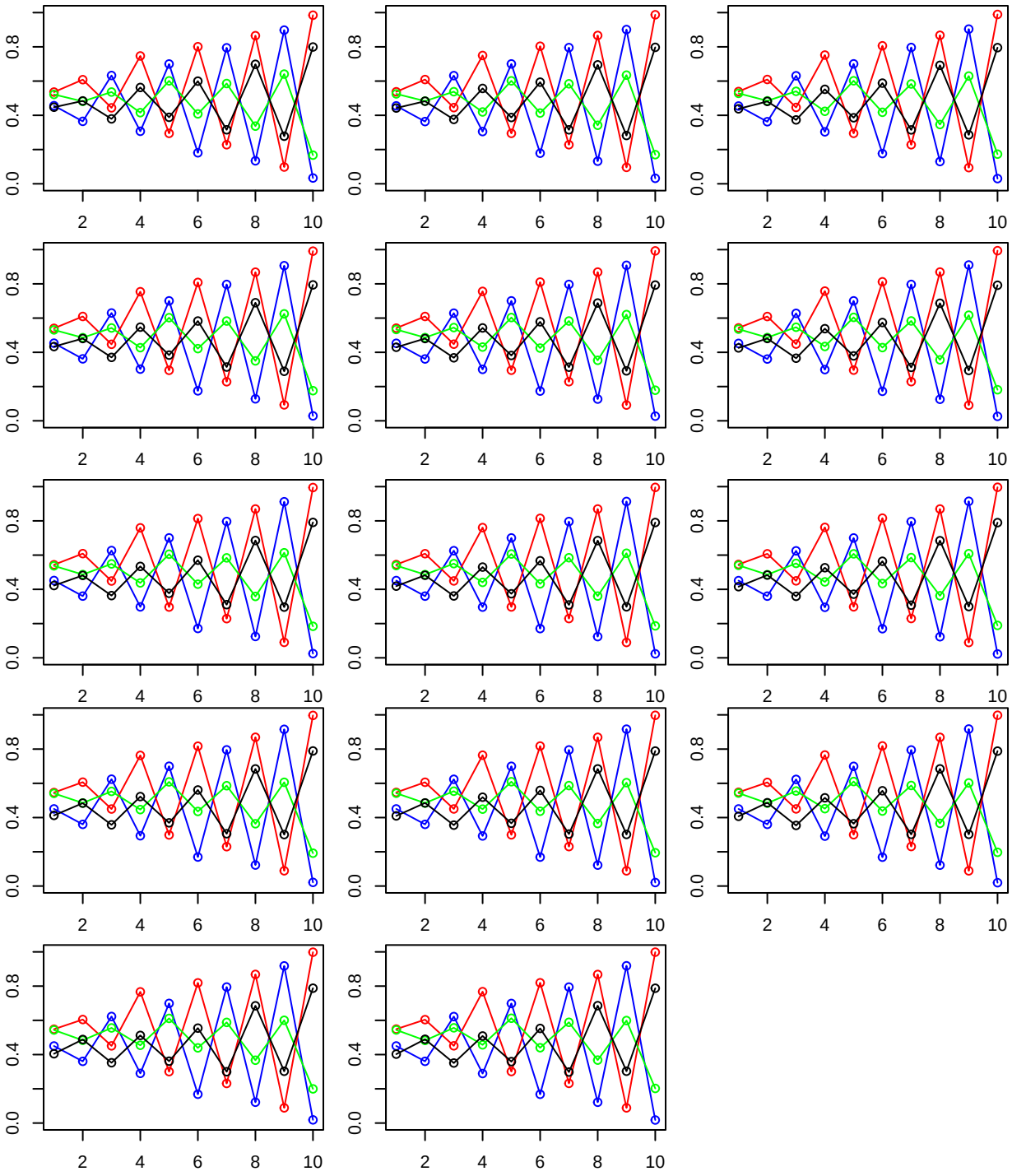
when M=4

```
## [1] 0.2461659 0.2478605 0.2544456 0.2515279
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4943551 0.5036051 0.4993455 0.5012054 0.5004571 0.4987214 0.4989227
## [2,] 0.5020358 0.5024204 0.5022187 0.4978065 0.5042903 0.5028750 0.4925821
## [3,] 0.4925133 0.4908125 0.4906579 0.4948314 0.4957019 0.4988231 0.4954166
## [4,] 0.4970116 0.4946235 0.5090379 0.5002896 0.5015360 0.4993958 0.4920966
##           [,8]      [,9]     [,10]
## [1,] 0.4917922 0.4954500 0.4901624
## [2,] 0.5086689 0.4947591 0.4942368
```

```
## [3,] 0.4925546 0.5016196 0.4978594
## [4,] 0.4947109 0.4902704 0.5019741
## 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
```


0.0
0.4
0.8
2
4
6
8
10



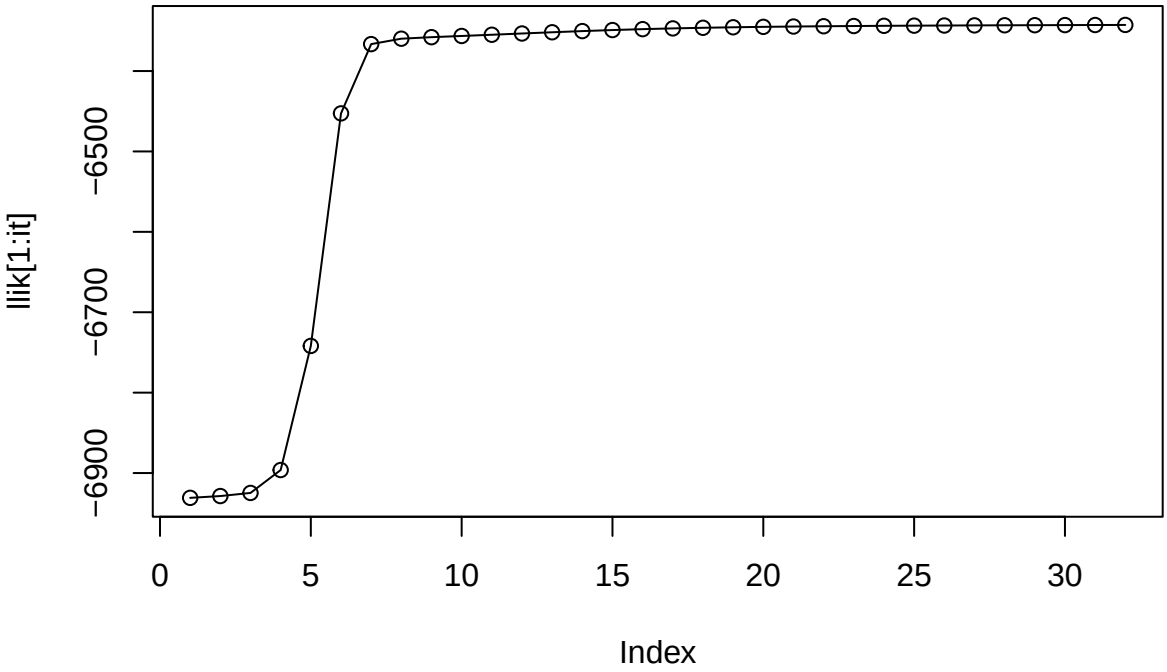

```
## The final pi is:

## [1] 0.2690190 0.2863050 0.2457246 0.1989515

## The final mu is:

##           [,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
```



According to above R results, it can be seen that:

1) When $M = 3$, the number of clusters in the model matches the true underlying structure of the data, thus in this scenario the estimated parameters can closely match the true mixing coefficients and conditional distributions(the final plot for μ distribution are quite similar to the plot for true μ distribution), and the model are able to capture the complexity of the data with three clusters. While, in terms of $M = 2$ or $M = 4$, the plots show that the estimated parameter deviate from the true distribution. When $M = 2$, the model oversimplify the data, leading to a less accurate representation of the true distribution. And when $M = 4$, the model try to fit into four clusters. As a result, the extra cluster might try to fit all the data, and overcomplicate the distribution. Besides, as the EM algorithm is for unsupervised learning, we don't the label each group orders for the training data, thus the order of final three clusters are not consistent with the order of true groups, but their distributions are similar.

2) According to the plots for log likelihood over iterations, when $M = 3$, the log likelihood increase steadily at the beginning, then reach a plateau after several iterations. The μ distribution plots also align with such behavior.

Appendix

Code for Assignment 1

Q1

```
library(randomForest)
library(knitr)
```

```r
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))

num_tree <- c(1,10,100)
num_repeat <- 1000
store_mat <- matrix(nrow = num_repeat, ncol = length(num_tree))
for (i in 1:length(num_tree)){
  for (j in 1:num_repeat){
    x1<-runif(100)
    x2<-runif(100)
    trdata<-cbind(x1,x2)
    y<-as.numeric(x1<x2)
    trlabels<-as.factor(y)

    m1 <- randomForest(x = trdata, y = trlabels, ntree = num_tree[i], nodesize = 25, keep.forest = TRUE)
    predicted_labels <- predict(m1, newdata = tedata)
    misclassification_error <- mean(predicted_labels != telabels)

    store_mat[j,i] <- misclassification_error
  }
}

mean_values <- colMeans(store_mat)
variance_values <- apply(store_mat, 2, var)

result_df <- data.frame(Mean = mean_values, Variance = variance_values)
rownames(result_df) <- c("1 tree", "10 trees", "100 trees")
kable(result_df, caption = "Classification Error", format = "markdown")
```

Q2

```r
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))

num_tree <- c(1,10,100)
num_repeat <- 1000
store_mat <- matrix(nrow = num_repeat, ncol = length(num_tree))
for (i in 1:length(num_tree)){
  for (j in 1:num_repeat){
    x1<-runif(100)
    x2<-runif(100)
    trdata<-cbind(x1,x2)
    y<-as.numeric(x1<0.5)
```

```
    trlabels<-as.factor(y)

    m1 <- randomForest(x = trdata, y = trlabels, ntree = num_tree[i], nodesize = 25, keep.forest = TRUE)
    predicted_labels <- predict(m1, newdata = tedata)
    misclassification_error <- mean(predicted_labels != telabels)

    store_mat[j,i] <- misclassification_error
  }
}

mean_values <- colMeans(store_mat)
variance_values <- apply(store_mat, 2, var)

result_df <- data.frame(Mean = mean_values, Variance = variance_values)
rownames(result_df) <- c("1 tree", "10 trees", "100 trees")
kable(result_df, caption = "Classification Error", format = "markdown")
```

Q3

```
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))

num_tree <- c(1,10,100)
num_repeat <- 1000
store_mat <- matrix(nrow = num_repeat, ncol = length(num_tree))
for (i in 1:length(num_tree)){
  for (j in 1:num_repeat){
    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)

    m1 <- randomForest(x = trdata, y = trlabels, ntree = num_tree[i], nodesize = 12, keep.forest = TRUE)
    predicted_labels <- predict(m1, newdata = tedata)
    misclassification_error <- mean(predicted_labels != telabels)

    store_mat[j,i] <- misclassification_error
  }
}

mean_values <- colMeans(store_mat)
variance_values <- apply(store_mat, 2, var)

result_df <- data.frame(Mean = mean_values, Variance = variance_values)
rownames(result_df) <- c("1 tree", "10 trees", "100 trees")
kable(result_df, caption = "Classification Error", format = "markdown")
```

Code for 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), ylab = expression("value "*mu), main = expression(
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
#legend("bottomleft", legend = c(expression(mu[1]), expression(mu[2]), expression(mu[3])), col = c("blu

# 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])
  }
}
```

when M=3

```
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)
}
pi
mu

# create this vector to store value
w_sum <- vector(length = n)
# make labels and margins smaller
par(cex=0.8, mar=c(2,2,0.5,1), mfrow = c(3, 3))

for(it in 1:max_it) {
  plot(mu[1,], type="o", col="blue", ylim=c(0,1), ylab = expression("value "*mu))
  points(mu[2,], type="o", col="red")
  points(mu[3,], type="o", col="green")
  #points(mu[4,], type="o", col="yellow")
  Sys.sleep(0.5)
```

```
  # E-step: Computation of the weights
  for (i in 1:n) {
    for (m in 1:M) {
      # log operation for convenience
      w[i, m] <- log(pi[m]) + sum(x[i, ] * log(mu[m, ]) + (1 - x[i, ]) * log(1 - mu[m, ]))
    }
    # exp operation for setting value back
    w[i, ] <- exp(w[i, ])
    # get the real w[i,m]
    w_sum[i] <- sum(w[i, ])
    w[i, ] <- w[i, ] / w_sum[i]
  }
  #Log likelihood computation.
  llik[it] <- sum(log(w_sum))

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1 && abs(llik[it] - llik[it - 1]) < min_change) {
    break
  }
  #M-step: ML parameter estimation from the data and weights
  pi <- colMeans(w)
  for (m in 1:M) {
    mu[m, ] <- colSums(w[, m] * x) / sum(w[, m])
  }
}

cat("The final pi is:\n")
pi
cat("The final mu is:\n")
mu
plot(llik[1:it], type="o")
```

when M=2

```
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

# create this vector to store value
w_sum <- vector(length = n)
# make labels and margins smaller
```

```
par(cex=0.8, mar=c(2,2,0.5,1), mfrow = c(3, 3))

for(it in 1:max_it) {
  plot(mu[1,], type="o", col="blue", ylim=c(0,1), ylab = expression("value "*mu))
  points(mu[2,], type="o", col="red")
 # points(mu[3,], type="o", col="green")
  #points(mu[4,], type="o", col="yellow")
  Sys.sleep(0.5)
  # E-step: Computation of the weights
  for (i in 1:n) {
    for (m in 1:M) {
      # log operation for convenience
      w[i, m] <- log(pi[m]) + sum(x[i, ] * log(mu[m, ]) + (1 - x[i, ]) * log(1 - mu[m, ]))
    }
    # exp operation for setting value back
    w[i, ] <- exp(w[i, ])
    # get the real w[i,m]
    w_sum[i] <- sum(w[i, ])
    w[i, ] <- w[i, ] / w_sum[i]
  }
  #Log likelihood computation.
  llik[it] <- sum(log(w_sum))

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1 && abs(llik[it] - llik[it - 1]) < min_change) {
    break
  }
  #M-step: ML parameter estimation from the data and weights
  pi <- colMeans(w)
  for (m in 1:M) {
    mu[m, ] <- colSums(w[, m] * x) / sum(w[, m])
  }
}

cat("The final pi is:\n")
pi
cat("The final mu is:\n")
mu
plot(llik[1:it], type="o")
```

when M=4

```
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)
}
pi
mu

# create this vector to store value
w_sum <- vector(length = n)
# make labels and margins smaller
par(cex=0.8, mar=c(2,2,0.5,1), mfrow = c(3, 3))

for(it in 1:max_it) {
  plot(mu[1,], type="o", col="blue", ylim=c(0,1), ylab = expression("value "*mu))
  points(mu[2,], type="o", col="red")
  points(mu[3,], type="o", col="green")
  points(mu[4,], type="o", col="black")
  Sys.sleep(0.5)
  # E-step: Computation of the weights
  for (i in 1:n) {
    for (m in 1:M) {
      # log operation for convenience
      w[i, m] <- log(pi[m]) + sum(x[i, ] * log(mu[m, ]) + (1 - x[i, ]) * log(1 - mu[m, ]))
    }
    # exp operation for setting value back
    w[i, ] <- exp(w[i, ])
    # get the real w[i,m]
    w_sum[i] <- sum(w[i, ])
    w[i, ] <- w[i, ] / w_sum[i]
  }
  #Log likelihood computation.
  llik[it] <- sum(log(w_sum))

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1 && abs(llik[it] - llik[it - 1]) < min_change) {
    break
  }
  #M-step: ML parameter estimation from the data and weights
  pi <- colMeans(w)
  for (m in 1:M) {
    mu[m, ] <- colSums(w[, m] * x) / sum(w[, m])
  }
}

cat("The final pi is:\n")
pi
cat("The final mu is:\n")
mu
plot(llik[1:it], type="o")
```