Machine Learning Block 2, Lab 1

Group A13: Connor Turner, Arash Haratian, Yi-Hung Chen

2022-12-07

Statement of Contribution: *For this lab, each member of the group worked on each assignment individually, and the answers in this report are a collaboration between the answers of all three members.*

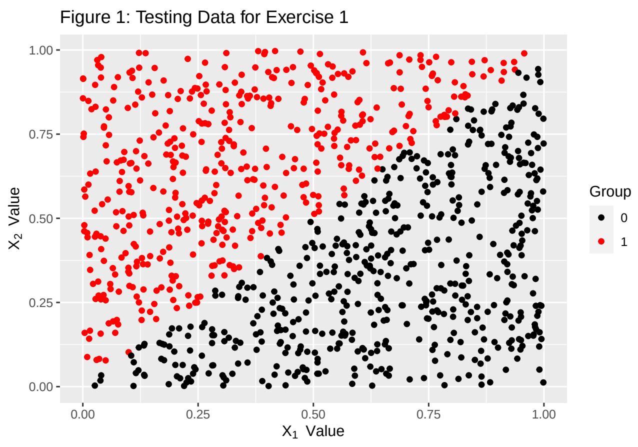
Assignment 1: Ensemble Methods

The purpose of this assignment was to run random forest algorithms with different settings and different levels of classification. To do this, three separate exercises were performed using the `randomForest` package and function in R, creating random forest models with 1, 10, and 100 trees. For each exercise, 1000 testing data points were generated with parameters x_1 and x_2, where each parameter was randomly generated from the Uniform distribution with a minimum of 0 and a maximum of 1. The groups to be learned were generated based on the values of x_1 and x_2, as will be described in the exercises below.

For each exercise, a total of 3000 random forest models were trained using 1000 generated training datasets with 100 values each for x_1 and x_2. These training data were generated in the exact same way as the testing data in each exercise. For each set of training data, three different forests were trained: one with 1 tree, one with 10 trees, and one with 100 trees. For each of these random forests, the models that were learned were used to fit the testing data, and the misclassification rate was recorded for each. After 1000 iterations of each forest were tested, the mean and variance of the misclassification rates were calculated and compared to the measurements from the other forest sizes.

Exercise 1

In the first exercise, the data were split according to the condition $x_1 < x_2$, where the true values were given a label of 1 (shown in red below) and the false values were given a label of 0 (shown in black). For each of the three types of forests being tested, the minimum size of the terminal nodes was set to 25. Figure 1 below shows the true pattern of the testing data:



.
The mean and variance of the misclassification rate for each type of forest is shown in Table 1 below:

Table 1: Misclassification Rates for Models in Exercise 1

Number of Trees	Mean Training Error	Training Error Variance	Mean Testing Error	Testing Error Variance
1	0.1572	0.0034127	0.2099	0.0034127
10	0.0752	0.0009463	0.1354	0.0009463
100	0.0477	0.0008627	0.1113	0.0008627

Exercise 2

Compared to Exercise 1, the groups in Exercise 2 were slightly simpler to classify. This time, the groups were split using the condition $x_1 < 0.5$, with x_2 having no say in the group labels. These forests were trained using the exact same settings as in Exercise 1. The true pattern of the testing data is shown in Figure 2 below, and the summary of the misclassification rates is shown in Table 2:

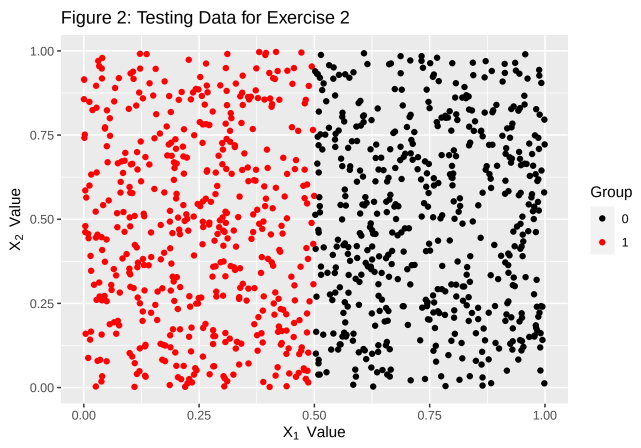


Table 2: Misclassification Rates for Models in Exercise 2

Number of Trees	Mean Training Error	Training Error Variance	Mean Testing Error	Testing Error Variance
1	0.0654	0.0172945	0.092951	0.0172945
10	0.0043	0.0004637	0.013719	0.0004637
100	0.0003	0.0000672	0.006411	0.0000672

Exercise 3

In Exercise 3, the group classification was more complex compared to the previous exercises. In this case, the data points were only put in Group 1 if either:

a) both x_1 and x_2 were less than 0.5
b) both x_1 and x_2 were greater than 0.5

For this exercise, the minimum size of the terminal nodes were set to 12 to allow larger, more complex decision trees to be grown for each model. The true pattern of the testing data and the summary of the misclassification rates are shown below:

Figure 3: Testing Data for Exercise 3

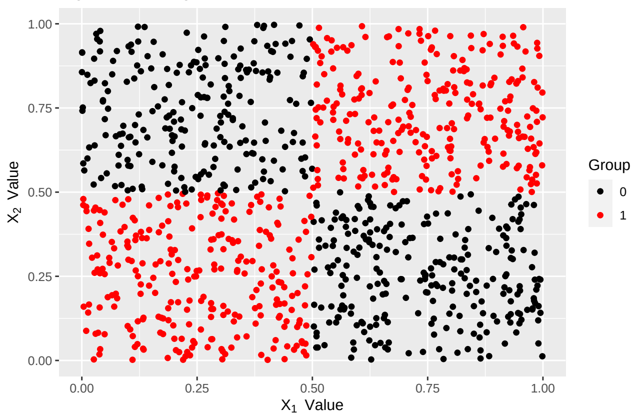


Table 3: Misclassification Rates for Models in Exercise 3

Number of Trees	Mean Training Error	Training Error Variance	Mean Testing Error	Testing Error Variance
1	0.1566	0.0072773	0.2512	0.0132559
10	0.0313	0.0006222	0.1227	0.0031819
100	0.0073	0.0000959	0.0770	0.0014162

From these exercises, two conclusions become quite clear. First, in each of these three exercises, both the mean and the variance of the misclassification rate decreased as the number of trees grown by the model increased. This was expected, as the aim of boosting is to mitigate overfitting by training the model with different boosting datasets and using the wisdom of crowds to create a generalized prediction. In other words, when a random forest has a greater number of decision trees, a greater number of predictions are being made based on the training data in different ways. And when enough of these predictions are combined via majority voting, the knowledge learned from each model comes together to paint the big picture. As a result, the model will be more accurate (i.e. have a lower error rate) and be more consistent (i.e. have a lower variance in the error rate) as the number of trees in the forest increases.

Second, even though the testing data in Exercise 3 was more complex compared to the other exercises, the misclassification rates were at the same level, if not lower, than the rates in Exercise 1 – particularly when a greater number of trees were grown. There are three reasons for this: two that have to do with the model itself and one that has to do with the data.

1. The number of trees is crucial to the accuracy of the predictions. As explained earlier, random forest models rely on the wisdom of crowds to reduce overfitting on complex models and provide a better generalized prediction model. Exercise 3 shows this in action when it comes to more complex data. As expected, the model with only one decision tree had a difficult time parsing through the data and making classifications on its own. But when more trees were added, the collective knowledge of each tree allowed the model to better classify each point in the testing data, resulting in a more accurate model.

2. The minimum node size, which was briefly mentioned earlier, plays a key role here. In the first two exercises, the minimum terminal node size was set to 25, whereas in Exercise 3, the minimum node size was set to 12, less than half the original size. The larger value causes the first six models to grow smaller trees, as the data can only be divided so much before reaching the minimum size of 25. However, the models in Exercise 3 were able to grow much larger trees that split the data more finely, which ultimately allowed the predictions of these models to be just as good as some others despite the increased complexity.

3. The way the groups were split in the data in Example 3 is more conducive to accurate predictions from decision trees. In Figure 1, the groups are split along a diagonal in the graph. Since the trees in the model can only split the data along the x-axis (vertically) or the y-axis (horizontally), it is difficult for the model to perfectly capture the relationship between the data and group labels in the training datasets. In Example 3, however, the groups were split along horizontal and vertical lines, which allows the random forest models to capture this relationship easier than the relationship in Example 1.

Assignment 2: The Bernoulli Mixture Model

The aim of this assignment is to implement an expectation-maximization (EM) algorithm for a Bernoulli mixture model to find the behavior of different unknown clusters within a set of data.

The distribution of the Bernoulli mixture model is defined by the following function:

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

Where M is the number of clusters, μ_m is the mean value for the given cluster m, and π_m is the probability that the data point belongs to cluster m. The data, x, is a d-dimensional vector of binary values $x = (x_1, ..., x_d)$ that have a Bernoulli distribution defined by the following likelihood function:

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

Where μ_m =($\mu_{m,1}$,. . . ,$\mu_{m,d}$) is a d-dimensional vector describing the mean probabilities of $x_{m,d} = 1$ for cluster m and dimension d.

The first step in building this model is finding a set of weights, $w_i(m)$, for each training point i and cluster m that contribute proportionally to the probability estimate of training point i belonging to each cluster. To do so, one must first calculate the expected weights for each cluster (the expectation step), and then derive, calculate, and maximize the log-likelihood function for $p(x)$ to find $\hat{\pi}_m$ and $\hat{\mu}$.

Expectation Step:

$$w_m(weight) = p(y=m|x_d) = \frac{\hat{\pi}_m * Bern(x_d|\mu_{m,d})}{\sum_d \hat{\pi}_m * Bern(x_d|\mu_{m,d})}$$

Where the log-likelihood is defined by the following formula:

$$log\_likelihood = \sum_m ln \sum_d \pi_m * Bern(x_d|\mu_{m,d})$$

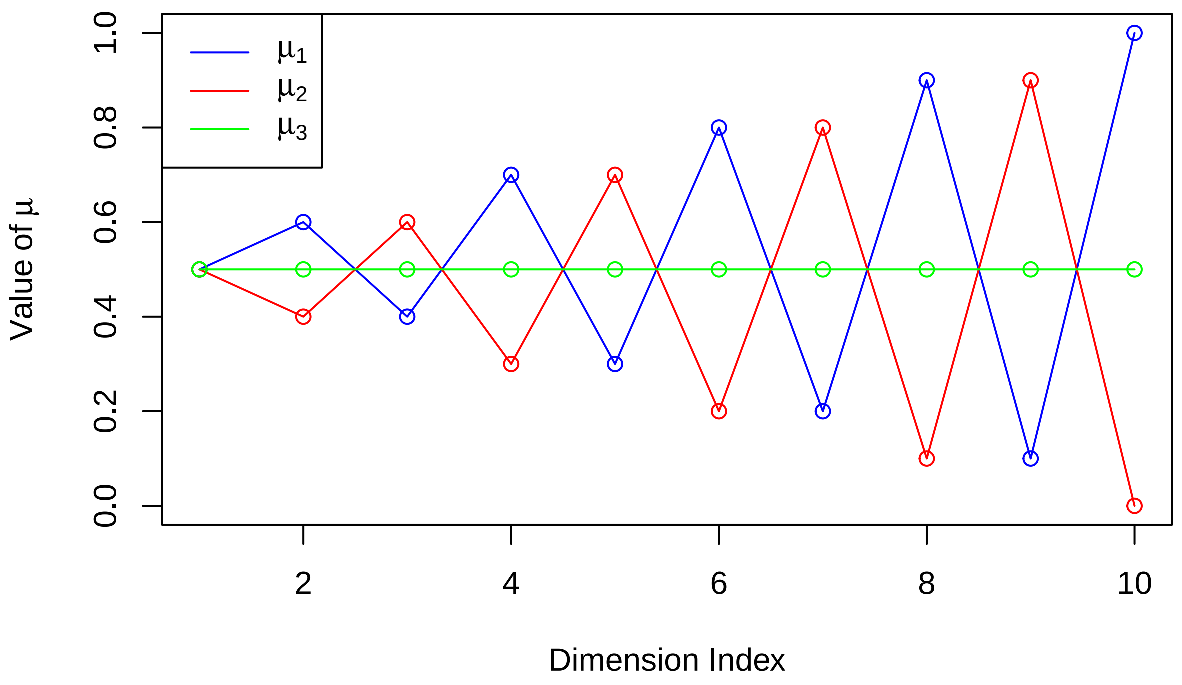
Maximization Step"

$$\hat{\pi}_m = \frac{\sum_d \pi_m * Bern(x_d|\mu_{m,d})}{n} = \frac{\sum weight}{n} = mean(weight)$$

$$\hat{\mu} = \frac{\sum_n^{i=1} w_i(m)x_i}{\sum_n^{i=1} w_i(m)} = \frac{p(y=m|x_d) * x}{p(y=m|x_d)}$$

In the following figures, the results of the mixture model are compared for across different numbers of clusters, $M = 2, 3, 4$:

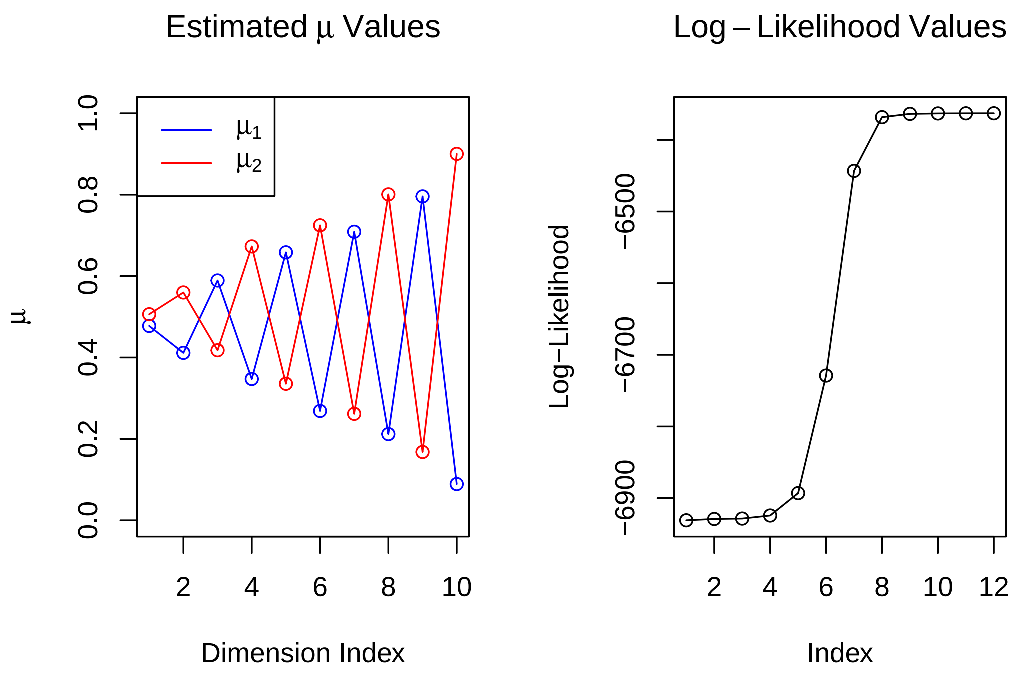
Figure 4 : True Values of μ for Clusters 1 – 3



.

Two-Cluster Model

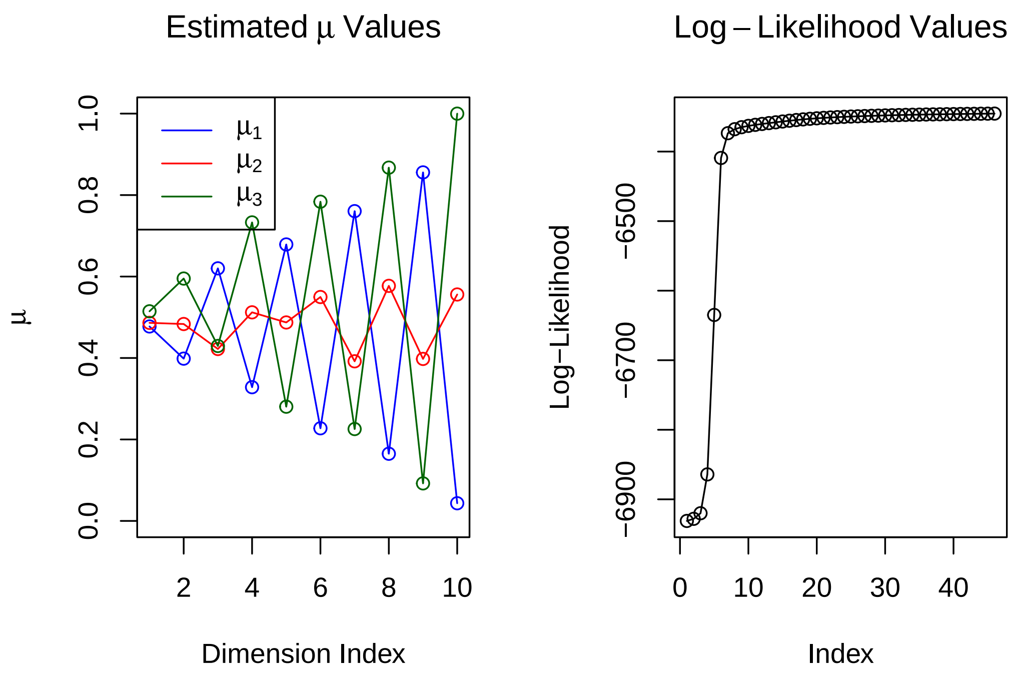
.



.

Three-Cluster Model

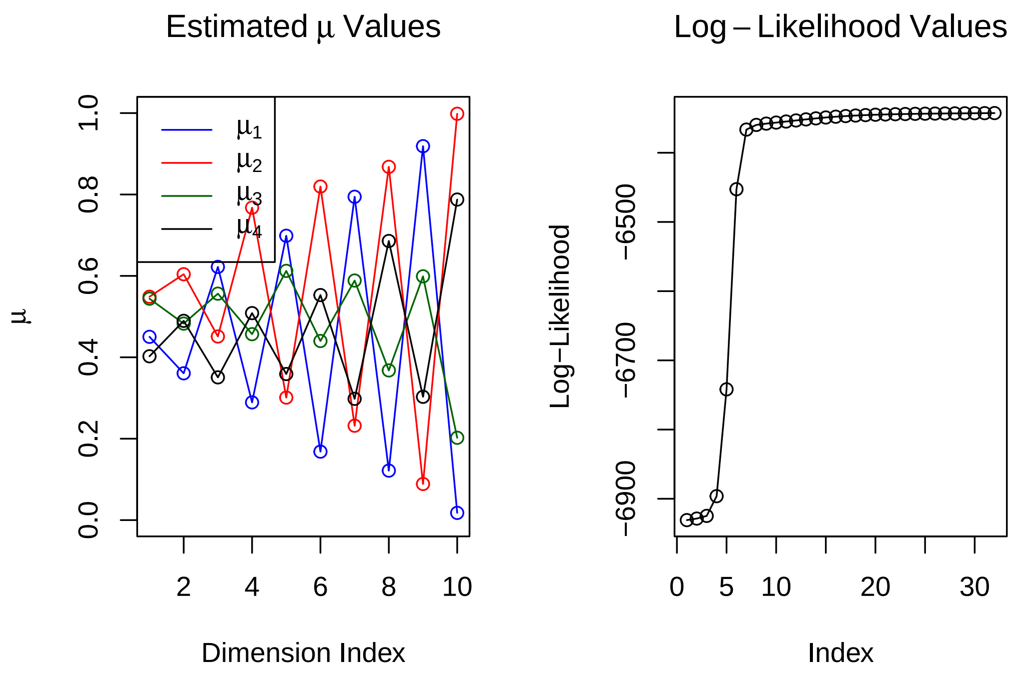
.



.

Four-Cluster Model

.



Comparing the results for the estimated μ values to the true μ values shown in Figure 4, one can conclude that finding the middle cluster (where $\mu = 0.5$ for all dimensions) is more difficult for the model than finding the other two clusters (where the μ values vary across dimensions). For example, when $M = 3$, the μ values estimated by the model almost perfectly match the values in Figure 4. However, instead of a straight line going straight across the middle of the plot representing the middle cluster, the μ values vary up and down across dimensions.

When $M = 2$, the model had too few clusters compared to the true data. In this model, the EM algorithm captured the pattern of both clusters with fluctuating μ values and used those μ values to define the two clusters. However, this relationship was slightly off compared to the true μ values shown in Figure 4. The reason for this is that the middle cluster with true $\mu = 0.5$ was not recognized by the model. Instead, points in that cluster affected the calculations for the other two clusters, which slightly condensed the mu values toward $\mu = 0.5$ and resulted in μ estimates that are slightly lower or higher than the true values for the other two clusters.

When $M = 4$, the model had too many clusters compared to the true data. This created a fourth cluster that was built by mixing up the 3 "true" clusters. In the plot above, it appears that the middle cluster with true $\mu = 0.5$ was divided up such that the middle line was replaced by two more jagged lines. However, the two clusters with the fluctuating μ values discussed earlier still (mostly) retained the same shape. This further bolsters the conclusion that the middle cluster was the most difficult for the model to estimate.

Note: The formulas for the EM algorithm were gathered from the lecture slides and the link below:

https://cedar.buffalo.edu/~srihari/CSE574/Chap9/Ch9.4-MixturesofBernoulli.pdf

Appendix 1: Code for Assignment 1

```r
library(randomForest)

# Exercise 1 -------------------------------------------------------------

# Create vectors for error rates
trmisclassrates1 <- c()
trmisclassrates10 <- c()
trmisclassrates100 <- c()
misclassrates1 <- c()
misclassrates10 <- c()
misclassrates100 <- c()

# Create testing data
set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
telabels<-as.factor(y)
testdata1 <- data.frame(x1, x2, telabels)

# Plot testing data
ggplot(data = testdata1) +
  geom_point(mapping = aes(x = x1, y = x2, color = telabels)) +
  scale_color_manual(values = c("black", "red")) +
  labs(title = "Figure 1: Testing Data for Exercise 1",
       x = expression(X[1]~~Value),
       y = expression(X[2]~~Value),
       color = "Group")

for (i in 1:1000){
  # Build training data
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<x2)
  trlabels<-as.factor(y)

  # Train models
  forest1 <- randomForest(trdata, y = trlabels, ntree = 1,
                         nodesize = 25, keep.forest = TRUE)
  forest10 <- randomForest(trdata, y = trlabels, ntree = 10,
                          nodesize = 25, keep.forest = TRUE)
  forest100 <- randomForest(trdata, y = trlabels, ntree = 100,
                           nodesize = 25, keep.forest = TRUE)

  # Use models to predict group labels
  trghat1 <- predict(object = forest1, newdata = trdata)
  trghat10 <- predict(object = forest10, newdata = trdata)
  trghat100 <- predict(object = forest100, newdata = trdata)
  ghat1 <- predict(object = forest1, newdata = tedata)
  ghat10 <- predict(object = forest10, newdata = tedata)
  ghat100 <- predict(object = forest100, newdata = tedata)

  # Create confusion matrix to split correct and incorrect predictions
  trcompare1 <- table(trlabels, trghat1)
  trcompare10 <- table(trlabels, trghat10)
  trcompare100 <- table(trlabels, trghat100)
  compare1 <- table(telabels, ghat1)
  compare10 <- table(telabels, ghat10)
  compare100 <- table(telabels, ghat100)

  # Find total number of misclassifications
  trmisclass1 <- trcompare1[1, 2] + trcompare1[2, 1]
  trmisclass10 <- trcompare10[1, 2] + trcompare10[2, 1]
  trmisclass100 <- trcompare100[1, 2] + trcompare100[2, 1]
  misclass1 <- compare1[1, 2] + compare1[2, 1]
  misclass10 <- compare10[1, 2] + compare10[2, 1]
  misclass100 <- compare100[1, 2] + compare100[2, 1]

  # Calculate misclassification rates
  trmisclassrate1 <- trmisclass1 / 100
  trmisclassrate10 <- trmisclass10 / 100
  trmisclassrate100 <- trmisclass100 / 100
  misclassrate1 <- misclass1 / 1000
  misclassrate10 <- misclass10 / 1000
  misclassrate100 <- misclass100 / 1000

  # Add misclassification rates to vector
  trmisclassrates1 <- append(trmisclassrates1, trmisclassrate1)
  trmisclassrates10 <- append(trmisclassrates10, trmisclassrate10)
  trmisclassrates100 <- append(trmisclassrates100, trmisclassrate100)
  misclassrates1 <- append(misclassrates1, misclassrate1)
  misclassrates10 <- append(misclassrates10, misclassrate10)
  misclassrates100 <- append(misclassrates100, misclassrate100)
}

# Report mean and variance in training and testing errors
numtrees1 <- c(1, 10, 100)
trmean1 <- c(mean(trmisclassrates1), mean(trmisclassrates10),mean(trmisclassrates100))
trmean1 <- round(trmean1, 4)
mean1 <- c(mean(misclassrates1), mean(misclassrates10),mean(misclassrates100))
mean1 <- round(mean1, 4)
trvar1 <- c(var(misclassrates1), var(misclassrates10), var(misclassrates100))
var1 <- c(var(misclassrates1), var(misclassrates10), var(misclassrates100))
tree1results <- data.frame(numtrees1, trmean1, trvar1, mean1, var1)
kable(tree1results, caption = "Misclassification Rates for Models in Exercise 1",
      col.names = c("Number of Trees", "Mean Training Error", "Training Error Variance",
                    "Mean Testing Error", "Testing Error Variance"))


# Exercise 2 ------------------------------------------------------------

trmisclassrates2_1 <- c()
```

```r
trmisclassrates2_10 <- c()
trmisclassrates2_100 <- c()
misclassrates2_1 <- c()
misclassrates2_10 <- c()
misclassrates2_100 <- c()

set.seed(1234)
x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(x1<0.5)
telabels<-as.factor(y)
testdata2 <- data.frame(x1, x2, telabels)

ggplot(data = testdata2) +
  geom_point(mapping = aes(x = x1, y = x2, color = telabels)) +
  scale_color_manual(values = c("black", "red")) +
  labs(title = "Figure 2: Testing Data for Exercise 2",
       x = expression(X[1]~~Value),
       y = expression(X[2]~~Value),
       color = "Group")

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)

  forest2_1 <- randomForest(trdata, y = trlabels, ntree = 1,
                            nodesize = 25, keep.forest = TRUE)
  forest2_10 <- randomForest(trdata, y = trlabels, ntree = 10,
                             nodesize = 25, keep.forest = TRUE)
  forest2_100 <- randomForest(trdata, y = trlabels, ntree = 100,
                              nodesize = 25, keep.forest = TRUE)

  trghat2_1 <- predict(object = forest2_1, newdata = trdata)
  trghat2_10 <- predict(object = forest2_10, newdata = trdata)
  trghat2_100 <- predict(object = forest2_100, newdata = trdata)
  ghat2_1 <- predict(object = forest2_1, newdata = tedata)
  ghat2_10 <- predict(object = forest2_10, newdata = tedata)
  ghat2_100 <- predict(object = forest2_100, newdata = tedata)

  trcompare2_1 <- table(trlabels, trghat2_1)
  trcompare2_10 <- table(trlabels, trghat2_10)
  trcompare2_100 <- table(trlabels, trghat2_100)
  compare2_1 <- table(telabels, ghat2_1)
  compare2_10 <- table(telabels, ghat2_10)
  compare2_100 <- table(telabels, ghat2_100)

  trmisclass2_1 <- trcompare2_1[1, 2] + trcompare2_1[2, 1]
  trmisclass2_10 <- trcompare2_10[1, 2] + trcompare2_10[2, 1]
  trmisclass2_100 <- trcompare2_100[1, 2] + trcompare2_100[2, 1]
  misclass2_1 <- compare2_1[1, 2] + compare2_1[2, 1]
  misclass2_10 <- compare2_10[1, 2] + compare2_10[2, 1]
  misclass2_100 <- compare2_100[1, 2] + compare2_100[2, 1]

  trmisclassrate2_1 <- trmisclass2_1 / 100
  trmisclassrate2_10 <- trmisclass2_10 / 100
  trmisclassrate2_100 <- trmisclass2_100 / 100
  misclassrate2_1 <- misclass2_1 / 1000
  misclassrate2_10 <- misclass2_10 / 1000
  misclassrate2_100 <- misclass2_100 / 1000

  trmisclassrates2_1 <- append(trmisclassrates2_1, trmisclassrate2_1)
  trmisclassrates2_10 <- append(trmisclassrates2_10, trmisclassrate2_10)
  trmisclassrates2_100 <- append(trmisclassrates2_100, trmisclassrate2_100)
  misclassrates2_1 <- append(misclassrates2_1, misclassrate2_1)
  misclassrates2_10 <- append(misclassrates2_10, misclassrate2_10)
  misclassrates2_100 <- append(misclassrates2_100, misclassrate2_100)
}

numtrees2 <- c(1, 10, 100)
trmean2 <- c(mean(trmisclassrates2_1), mean(trmisclassrates2_10),mean(trmisclassrates2_100))
trmean2 <- round(trmean2, 4)
mean2 <- c(mean(misclassrates2_1), mean(misclassrates2_10),mean(misclassrates2_100))
trvar2 <- c(var(misclassrates2_1), var(misclassrates2_10), var(misclassrates2_100))
var2 <- c(var(misclassrates2_1), var(misclassrates2_10), var(misclassrates2_100))
tree2results <- data.frame(numtrees2, trmean2, trvar2, mean2, var2)
kable(tree2results, caption = "Misclassification Rates for Models in Exercise 2",
      col.names = c("Number of Trees", "Mean Training Error", "Training Error Variance",
                    "Mean Testing Error", "Testing Error Variance"))


# Exercise 3 -------------------------------------------------------------

trmisclassrates3_1 <- c()
trmisclassrates3_10 <- c()
trmisclassrates3_100 <- c()
misclassrates3_1 <- c()
misclassrates3_10 <- c()
misclassrates3_100 <- c()

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)
testdata3 <- data.frame(x1, x2, telabels)

ggplot(data = testdata3) +
  geom_point(mapping = aes(x = x1, y = x2, color = telabels)) +
  scale_color_manual(values = c("black", "red")) +
  labs(title = "Figure 3: Testing Data for Exercise 3",
       x = expression(X[1]~~Value),
       y = expression(X[2]~~Value),
       color = "Group")

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)

  forest3_1 <- randomForest(trdata, y = trlabels, ntree = 1, nodesize = 12, keep.forest = TRUE)
  forest3_10 <- randomForest(trdata, y = trlabels, ntree = 10, nodesize = 12, keep.forest = TRUE)
  forest3_100 <- randomForest(trdata, y = trlabels, ntree = 100, nodesize = 12, keep.forest = TRUE)

  trghat3_1 <- predict(object = forest3_1, newdata = trdata)
  trghat3_10 <- predict(object = forest3_10, newdata = trdata)
  trghat3_100 <- predict(object = forest3_100, newdata = trdata)
  ghat3_1 <- predict(object = forest3_1, newdata = tedata)
  ghat3_10 <- predict(object = forest3_10, newdata = tedata)
  ghat3_100 <- predict(object = forest3_100, newdata = tedata)

  trcompare3_1 <- table(trlabels, trghat3_1)
  trcompare3_10 <- table(trlabels, trghat3_10)
  trcompare3_100 <- table(trlabels, trghat3_100)
  compare3_1 <- table(telabels, ghat3_1)
  compare3_10 <- table(telabels, ghat3_10)
  compare3_100 <- table(telabels, ghat3_100)

  trmisclass3_1 <- trcompare3_1[1, 2] + trcompare3_1[2, 1]
  trmisclass3_10 <- trcompare3_10[1, 2] + trcompare3_10[2, 1]
  trmisclass3_100 <- trcompare3_100[1, 2] + trcompare3_100[2, 1]
  misclass3_1 <- compare3_1[1, 2] + compare3_1[2, 1]
  misclass3_10 <- compare3_10[1, 2] + compare3_10[2, 1]
  misclass3_100 <- compare3_100[1, 2] + compare3_100[2, 1]

  trmisclassrate3_1 <- trmisclass3_1 / 100
  trmisclassrate3_10 <- trmisclass3_10 / 100
  trmisclassrate3_100 <- trmisclass3_100 / 100
  misclassrate3_1 <- misclass3_1 / 1000
  misclassrate3_10 <- misclass3_10 / 1000
  misclassrate3_100 <- misclass3_100 / 1000

  trmisclassrates3_1 <- append(trmisclassrates3_1, trmisclassrate3_1)
  trmisclassrates3_10 <- append(trmisclassrates3_10, trmisclassrate3_10)
  trmisclassrates3_100 <- append(trmisclassrates3_100, trmisclassrate3_100)
  misclassrates3_1 <- append(misclassrates3_1, misclassrate3_1)
  misclassrates3_10 <- append(misclassrates3_10, misclassrate3_10)
  misclassrates3_100 <- append(misclassrates3_100, misclassrate3_100)
}

numtrees3 <- c(1, 10, 100)
trmean3 <- c(mean(trmisclassrates3_1), mean(trmisclassrates3_10),mean(trmisclassrates3_100))
trmean3 <- round(trmean3, 4)
```

```r
mean3 <- c(mean(misclassrates3_1), mean(misclassrates3_10),mean(misclassrates3_100))
mean3 <- round(mean3, 4)
trvar3 <- c(var(trmisclassrates3_1), var(trmisclassrates3_10), var(trmisclassrates3_100))
var3 <- c(var(misclassrates3_1), var(misclassrates3_10), var(misclassrates3_100))
tree3results <- data.frame(numtrees3, trmean3, trvar3, mean3, var3)
kable(tree3results, caption = "Misclassification Rates for Models in Exercise 3",
      col.names = c("Number of Trees", "Mean Training Error", "Training Error Variance",
                    "Mean Testing Error", "Testing Error Variance"))
```

Additional Code for Assignment 1

```r
#=====Alternative solution for assignment 1 from YiHung =====

#=====Training Data=====
set.seed(12345)
x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)
train_data<-as.data.frame(cbind(trdata,trlabels))

#=====Testing data=====
set.seed(1234)

x1<-runif(1000)
x2<-runif(1000)
tedata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
telabels<-as.factor(y)
test_data<-as.data.frame(cbind(tedata,telabels))
test_data$telabels<-as.factor(test_data$telabels)
plot(x1,x2,col=(y+1))


#=====Model training with 1,10,100 trees=====
model_1<-randomForest(factor(trlabels)~.,data=train_data,ntree=1,nodesize=25,keep.forest=TRUE)
model_10<-randomForest(factor(trlabels)~.,data=train_data,ntree=10,nodesize=25,keep.forest=TRUE)
model_100<-randomForest(factor(trlabels)~.,data=train_data,ntree=100,nodesize=25,keep.forest=TRUE)

#=====Predicting Model with different trees=====
pred_1<-predict(model_1,newdata=test_data)
pred_10<-predict(model_10,newdata=test_data)
pred_100<-predict(model_100,newdata=test_data,type="class")

missclass <- function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
missclass1 <- missclass(test_data$telabels , pred_1)
missclass10 <- missclass(test_data$telabels , pred_10)
missclass100 <- missclass(test_data$telabels , pred_100)
```

```r
error_table <- as.data.frame(cbind(missclass1,missclass10,missclass100))
error_table



#For 1000 data set, create a function to directly calculate misclassification error so it can be called
mis_class_error_x1_x2 <- function(tedata, telabels) {


  #training model is create every time the function call
  x1 <- runif(100)
  x2 <- runif(100)
  trdata <- cbind(x1, x2)
  y <- as.numeric(x1 < x2)
  trlabels <- as.factor(y)

  model_1 <- randomForest(trdata, trlabels, ntree = 1, nodesize = 25, keep.forest = TRUE)
  model_10 <- randomForest(trdata, trlabels, ntree = 10, nodesize = 25, keep.forest = TRUE)
  model_100 <- randomForest(trdata, trlabels, ntree = 100, nodesize = 25, keep.forest = TRUE)

  # Make predictions on the test data and calculate the misclassification error
  predictions_1 <- predict(model_1, tedata)
  predictions_10 <- predict(model_10, tedata)
  predictions_100 <- predict(model_100, tedata)
  error_1 <- mean(predictions_1 != telabels)
  error_10 <- mean(predictions_10 != telabels)
  error_100 <- mean(predictions_100 != telabels)



  return(c(error_1,error_10,error_100))
}


# Now we can apply the mis_class_error_x1_x2() function to 1000 training datasets of size 100
# and compute the misclassification error for random forests with 1, 10, and 100 trees
errors1 <- replicate(1000, mis_class_error_x1_x2(tedata, telabels))

# Finally, we can calculate the mean and variance of the misclassification errors for each number of tr
mean(errors1[1,])
mean(errors1[2,])
mean(errors1[3,])
var(errors1[1,])
var(errors1[2,])
var(errors1[3,])

#=====x1>0,5=====

mis_class_error_x1_05 <- function(tedata, telabels) {
  #training model is create every time the function call
  x1 <- runif(100)
  x2 <- runif(100)
  trdata <- cbind(x1, x2)
  y <- as.numeric(x1 < 0.5)
  trlabels <- as.factor(y)

  model_1 <- randomForest(trdata, trlabels, ntree = 1, nodesize = 25, keep.forest = TRUE)
  model_10 <- randomForest(trdata, trlabels, ntree = 10, nodesize = 25, keep.forest = TRUE)
  model_100 <- randomForest(trdata, trlabels, ntree = 100, nodesize = 25, keep.forest = TRUE)

  # Make predictions on the test data and calculate the misclassification error
  predictions_1 <- predict(model_1, tedata)
  predictions_10 <- predict(model_10, tedata)
  predictions_100 <- predict(model_100, tedata)
  error_1 <- mean(predictions_1 != telabels)
  error_10 <- mean(predictions_10 != telabels)
  error_100 <- mean(predictions_100 != telabels)



  return(c(error_1,error_10,error_100))
}
#new test set
x1<-runif(1000)
x2<-runif(1000)
tedata_05<-cbind(x1,x2)
y_05<-as.numeric(x1<0.5)
telabels_05<-as.factor(y_05)

errors_05 <- replicate(1000, mis_class_error_x1_05(tedata_05, telabels_05))

mean(errors_05[1,])
mean(errors_05[2,])
mean(errors_05[3,])
var(errors_05[1,])
var(errors_05[2,])
var(errors_05[3,])


#=====x1<0.5 & x2<0.5 | x1>0.5 & x2>0.5=====


mis_class_error_both05 <- function(tedata, telabels) {
  #training model is create every time the function call
  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)


  model_1 <- randomForest(trdata, trlabels, ntree = 1, nodesize = 12, keep.forest = TRUE)
  model_10 <- randomForest(trdata, trlabels, ntree = 10, nodesize = 12, keep.forest = TRUE)
  model_100 <- randomForest(trdata, trlabels, ntree = 100, nodesize = 12, keep.forest = TRUE)

  # Make predictions on the test data and calculate the misclassification error
  predictions_1 <- predict(model_1, tedata)
  predictions_10 <- predict(model_10, tedata)
  predictions_100 <- predict(model_100, tedata)
  error_1 <- mean(predictions_1 != telabels)
  error_10 <- mean(predictions_10 != telabels)
  error_100 <- mean(predictions_100 != telabels)



  return(c(error_1,error_10,error_100))
}

#new test set
# | (x1>0.5 & x2>0.5)
x1<-runif(1000)
x2<-runif(1000)
tedata_both05<-cbind(x1,x2)
y_both05<-as.numeric((x1 < 0.5 & x2 < 0.5) | (x1 > 0.5 & x2 > 0.5))
telabels_both05<-as.factor(y_both05)

errors_both05 <- replicate(1000, mis_class_error_both05(tedata_both05, telabels_both05))

mean(errors_both05[1,])
mean(errors_both05[2,])
mean(errors_both05[3,])
var(errors_both05[1,])
var(errors_both05[2,])
var(errors_both05[3,])

#=====End Of Alternative solution for assignment 1 by YiHung =====
```

Appendix 2: Code for Assignment 2

```r
# 2-0 ---------------------------------------------------------------------

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


true_pi <- vector(length = 3) # true mixing coefficients
true_pi <- c(1/3, 1/3, 1/3)

true_mu <- matrix(nrow = 3, ncol = D) # true conditional distributions
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
x <- matrix(nrow = n, ncol = D) # 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
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the parameters

pi <- vector(length = M) # mixing coefficients
pi <- runif(M, 0.49, 0.51)
pi <- pi / sum(pi)
pi

mu <- matrix(nrow = M, ncol = D) # conditional distributions
for(m in 1:M) {
  mu[m, ] <- runif(D, 0.49, 0.51)
}
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")
  # points(mu[4,], type="o", col="yellow")
  Sys.sleep(0.5)
  # E-step: Computation of the weights ----------------------------------------------------------------
  # Your code here

  for(i in 1:n){
    bern <- dbinom(x[i, ], 1, t(mu))
    bern <- apply(t(bern), prod, MARGIN = 1)
    w[i, ] <- (pi * bern) / (sum(pi * bern))
  }
  #
  # for(i in 1:1000){
  #   for(j in 1:3){
  #     bern <- 1
  #     for(m in 1:10){
  #       bern <-  bern * dbinom(x[1, m], 1, mu[j, m])
  #     }
  #     w[i, j] <- (pi[j] * bern) / (sum(pi[j] * bern))
  #   }
  # }

  #Log likelihood computation. -------------------------------------------------------------------
  # Your code here
  prob_x <- vector(length = 1000)
  for(i in 1:n){
    bern <- dbinom(x[i, ], 1, t(mu))
    bern <- apply(t(bern), prod, MARGIN = 1)
    prob_x[i] <- sum(pi * bern)
  }
  llik[it] <- sum(log(prob_x))

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly ------------------------------------
  # Your code here

  if(it != 1 && abs(llik[it] - llik[it-1]) < min_change)
    break

  #M-step: ML parameter estimation from the data and weights ------------------------------------
  # Your code here
  pi <- colMeans(w)
  mu <- (1/colSums(w)) * (t(w) %*% x)
}
pi
mu
plot(llik[1:it], type = "o")
```