Lab block 2

johmo870, nisra674

2024-11-30

Statement of Contribution

- Nisal Amashan(nisra674) - Assignment 2, Assignment 3, Report
- John Möller (johmo870) - Assignment 1, Report

Assignment 1

Loading the library

```
library(randomForest)
```

```
## randomForest 4.7-1.2
```

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

Random forest with 1, 10 and 100 trees

For this task, we generat training data based on the conditions provided, and then fit random forrests with different numbers of trees (1, 10, and 100). We then calcualte the misclassification error on the test dataset.

```
run_rf_and_compute_error <- function(ntree, nodesize) {

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

  rf_model <- randomForest(trdata, trlabels, ntree = ntree, nodesize = nodesize, keep.forest = TRUE)

  x1_test <- runif(1000)
  x2_test <- runif(1000)
  tedata <- cbind(x1_test, x2_test)
  colnames(tedata) <- colnames(trdata)
  y_test <- as.numeric(x1_test < x2_test)
  telabels <- as.factor(y_test)

  predictions <- predict(rf_model, tedata)
  error_rate <- mean(predictions != telabels)
  return(error_rate)
}

errors_1_tree <- run_rf_and_compute_error(1, 25)
```

```r
errors_10_trees <- run_rf_and_compute_error(10, 25)
errors_100_trees <- run_rf_and_compute_error(100, 25)

errors_1_tree
```

```
## [1] 0.171
```

```r
errors_10_trees
```

```
## [1] 0.174
```

```r
errors_100_trees
```

```
## [1] 0.101
```

Repeating procedure for 1000 datasets

```r
# Function to repeat procedure for 1000 datasets
repeat_rf_for_multiple_datasets <- function(ntree_value, nodesize_value) {
  errors <- numeric(1000)
  for (i in 1:1000) {
    errors[i] <- run_rf_and_compute_error(ntree_value, nodesize_value)
  }
  mean_error <- mean(errors)
  var_error <- var(errors)
  return(c(mean_error, var_error))
}

# Run for 1, 10, and 100 trees
result_1_tree <- repeat_rf_for_multiple_datasets(1, 25)
result_10_trees <- repeat_rf_for_multiple_datasets(10, 25)
result_100_trees <- repeat_rf_for_multiple_datasets(100, 25)

# Display the results for mean and variance of errors
result_1_tree
```

```
## [1] 0.204211000 0.002995916
```

```r
result_10_trees
```

```
## [1] 0.1362480000 0.0009983468
```

```r
result_100_trees
```

```
## [1] 0.1104660000 0.0008024113
```

(Column 1 is mean, column 2 is variance)

Using condition x1 < 0.5

```r
run_rf_condition_x1 <- function(ntree, nodesize) {

  x1 <- runif(100)
  x2 <- runif(100)
  trdata <- cbind(x1, x2)
  y <- as.numeric(x1 < 0.5)
  trlabels <- as.factor(y)
  rf_model <- randomForest(trdata, trlabels, ntree = ntree, nodesize = nodesize, keep.forest = TRUE)

  # Generate test data
  x1_test <- runif(1000)
  x2_test <- runif(1000)
  tedata <- cbind(x1_test, x2_test)
  colnames(tedata) <- colnames(trdata)
  y_test <- as.numeric(x1_test < 0.5)
  telabels <- as.factor(y_test)

  # Make predictions and calculate misclassification error
  predictions <- predict(rf_model, tedata)
  error_rate <- mean(predictions != telabels)
  return(error_rate)
}

# Run for 1, 10, and 100 trees
errors_1_tree_x1 <- run_rf_condition_x1(1, 25)
errors_10_trees_x1 <- run_rf_condition_x1(10, 25)
errors_100_trees_x1 <- run_rf_condition_x1(100, 25)

# Display the error rates for different numbers of trees
errors_1_tree_x1
```

```
## [1] 0.006
```

```
errors_10_trees_x1
```

```
## [1] 0.005
```

```
errors_100_trees_x1
```

```
## [1] 0.003
```

Using condition ((x1 $<$ 0.5 & x2 $<$ 0.5) | (x1 $>$ 0.5 & x2 $>$ 0.5)) and node size 12

```
# Function to run random forest with more complex condition
run_rf_complex_condition <- function(ntree_value, nodesize_value) {

  # Generate training data with the complex condition
  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)

  # Train the random forest
  rf_model <- randomForest(trdata, trlabels, ntree = ntree_value, nodesize = nodesize_value, keep.fores

  # Generate test data
  x1_test <- runif(1000)
  x2_test <- runif(1000)
  tedata <- cbind(x1_test, x2_test)
  colnames(tedata) <- colnames(trdata)
  y_test <- as.numeric((x1_test < 0.5 & x2_test < 0.5) | (x1_test > 0.5 & x2_test > 0.5))
  telabels <- as.factor(y_test)

  # Make predictions and calculate misclassification error
  predictions <- predict(rf_model, tedata)
  error_rate <- mean(predictions != telabels)
  return(error_rate)
}

# Run for 1, 10, and 100 trees with nodesize = 12
errors_1_tree_complex <- run_rf_complex_condition(1, 12)
errors_10_trees_complex <- run_rf_complex_condition(10, 12)
errors_100_trees_complex <- run_rf_complex_condition(100, 12)

# Display the error rates for different numbers of trees
errors_1_tree_complex
```

```
## [1] 0.081
```

```
errors_10_trees_complex
```

```
## [1] 0.114
```

```
errors_100_trees_complex
```

```
## [1] 0.045
```

Discussion

Since this method reduces variance with more trees, the error rate reduces. The first model has a desicsion boundary that is on an angle, whereas the last one has horizontal and vertical lines. Since trees divides the region up with horizontal and vertical lines, it performs better on the latter.

Assignment 2

MIXTURE MODELS

The EM algorithm was run for a Bernoulli Mixture Model with different values of M (number of clusters), specifically $M = 2$, $M = 3$, and $M = 4$. The objective was to analyze how the number of clusters influences the model's behavior, particularly its impact on the log-likelihood values and the convergence process.

Experimental Setup

- **Number of Data Points (n):** 1000
- **Number of Features (D):** 10
- **Maximum Iterations (max_it):** 100
- **Minimum Change in Log-Likelihood (min_change):** 0.1
- **True Mixture Components (true_pi):** $[1/3, 1/3, 1/3]$
- **True Conditional Distributions (true_mu):**
  - Cluster 1: $[0.5, 0.6, 0.4, 0.7, 0.3, 0.8, 0.2, 0.9, 0.1, 1]$
  - Cluster 2: $[0.5, 0.4, 0.6, 0.3, 0.7, 0.2, 0.8, 0.1, 0.9, 0]$
  - Cluster 3: $[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]$

Results

$M = 2$ (Too Few Clusters)

- **Final Log-Likelihood:** -6362.897
- **Number of Iterations for Convergence:** 12
- **Mixing Coefficients (π):**
  - Cluster 1: 0.4971
  - Cluster 2: 0.5029
- **Observed Behavior:** The log-likelihood converged quickly, but the two clusters attempted to merge three distinct distributions into two groups, leading to suboptimal modeling of the true data.

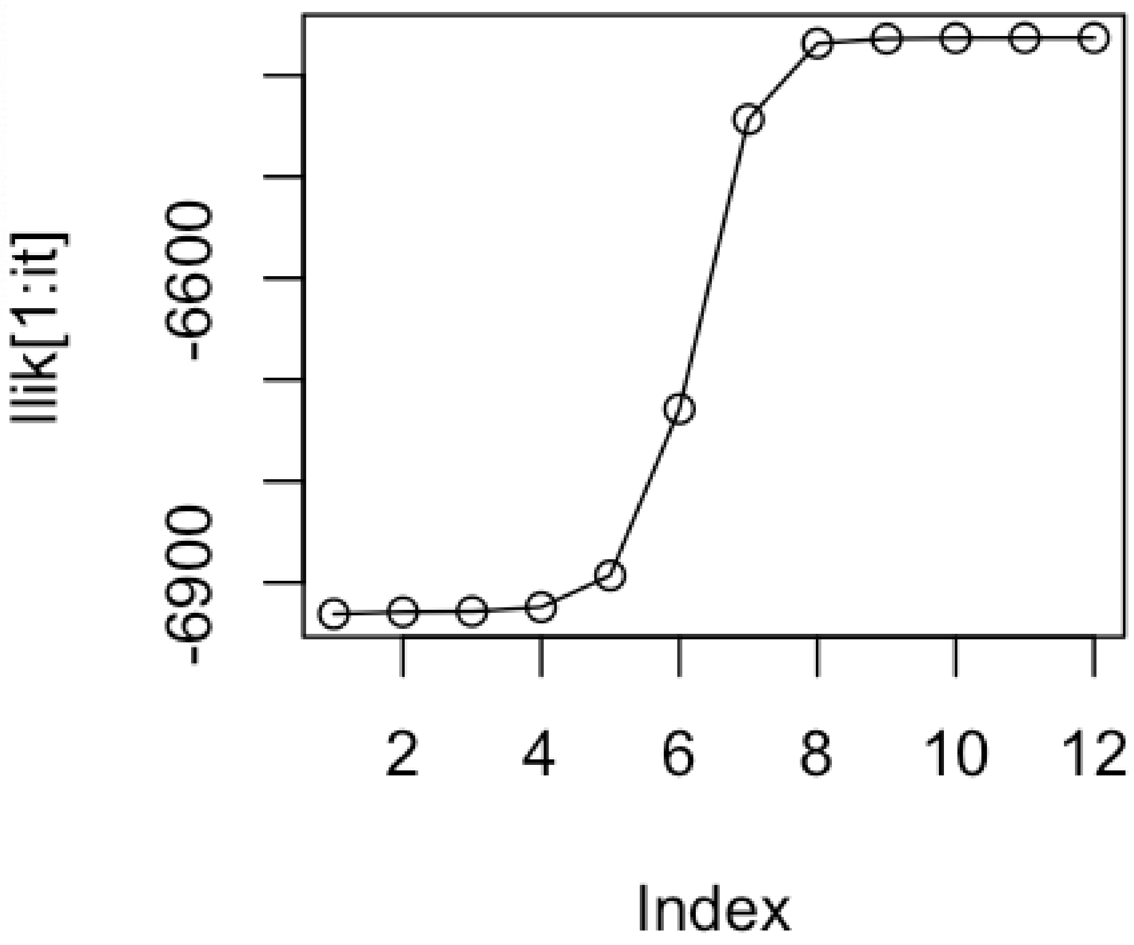

Figure 1: Log-Likelihood Plot for M=2

$M = 3$ (Ideal Number of Clusters)

- **Final Log-Likelihood:** -6344.57
- **Number of Iterations for Convergence:** 26
- **Mixing Coefficients (π):**
  - Cluster 1: 0.3417

    - Cluster 2: 0.2690
    - Cluster 3: 0.3893
- **Observed Behavior:** The log-likelihood achieved a higher value, indicating a better fit to the data. The three clusters accurately represented the true data distribution.

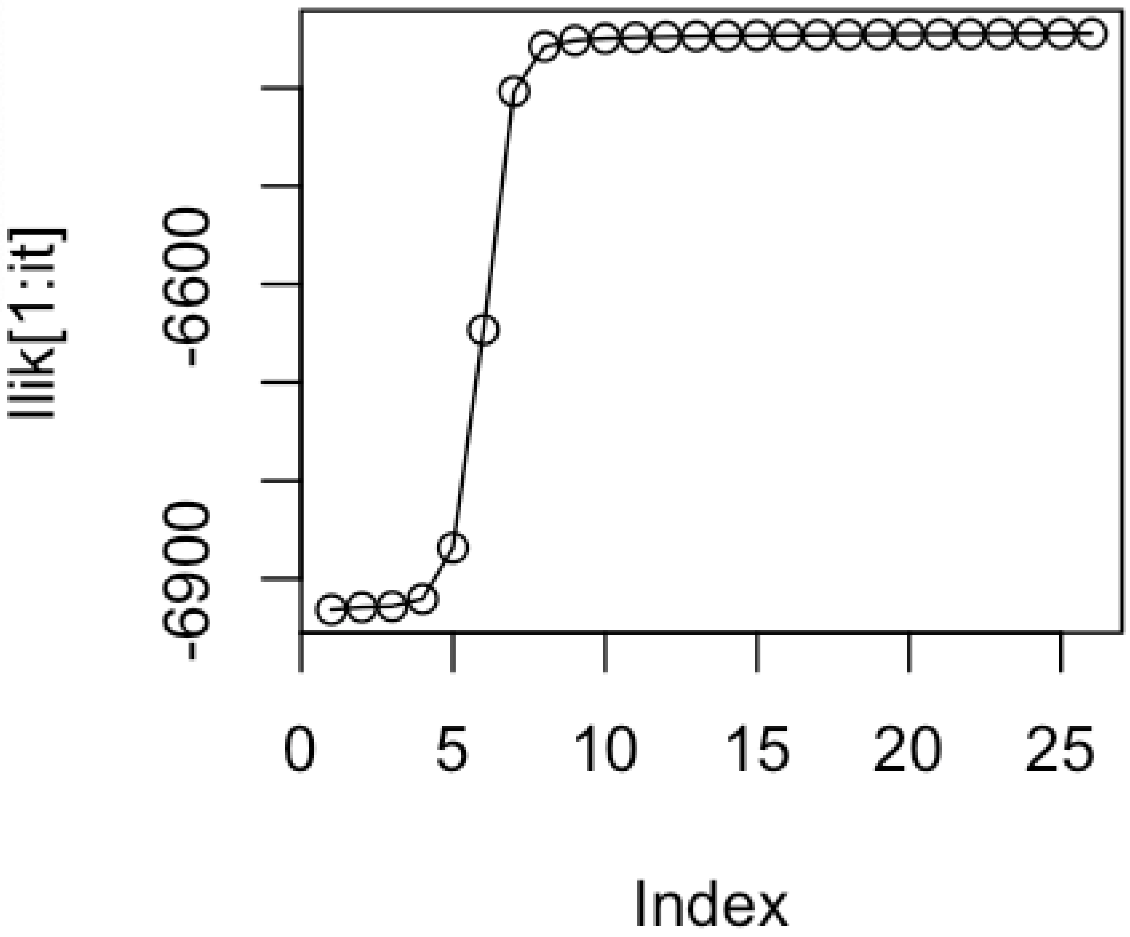

Figure 2: Log-Likelihood Plot for M=3

$M = 4$ **(Too Many Clusters)**

- **Final Log-Likelihood:** -6338.228
- **Number of Iterations for Convergence:** 44
- **Mixing Coefficients (π):**
    - Cluster 1: 0.1547
    - Cluster 2: 0.1419
    - Cluster 3: 0.3514
    - Cluster 4: 0.3520
- **Observed Behavior:** The model overfits the data by creating additional clusters that attempt to segment the data unnecessarily. This results in a marginal improvement in log-likelihood but at the cost of increased complexity.

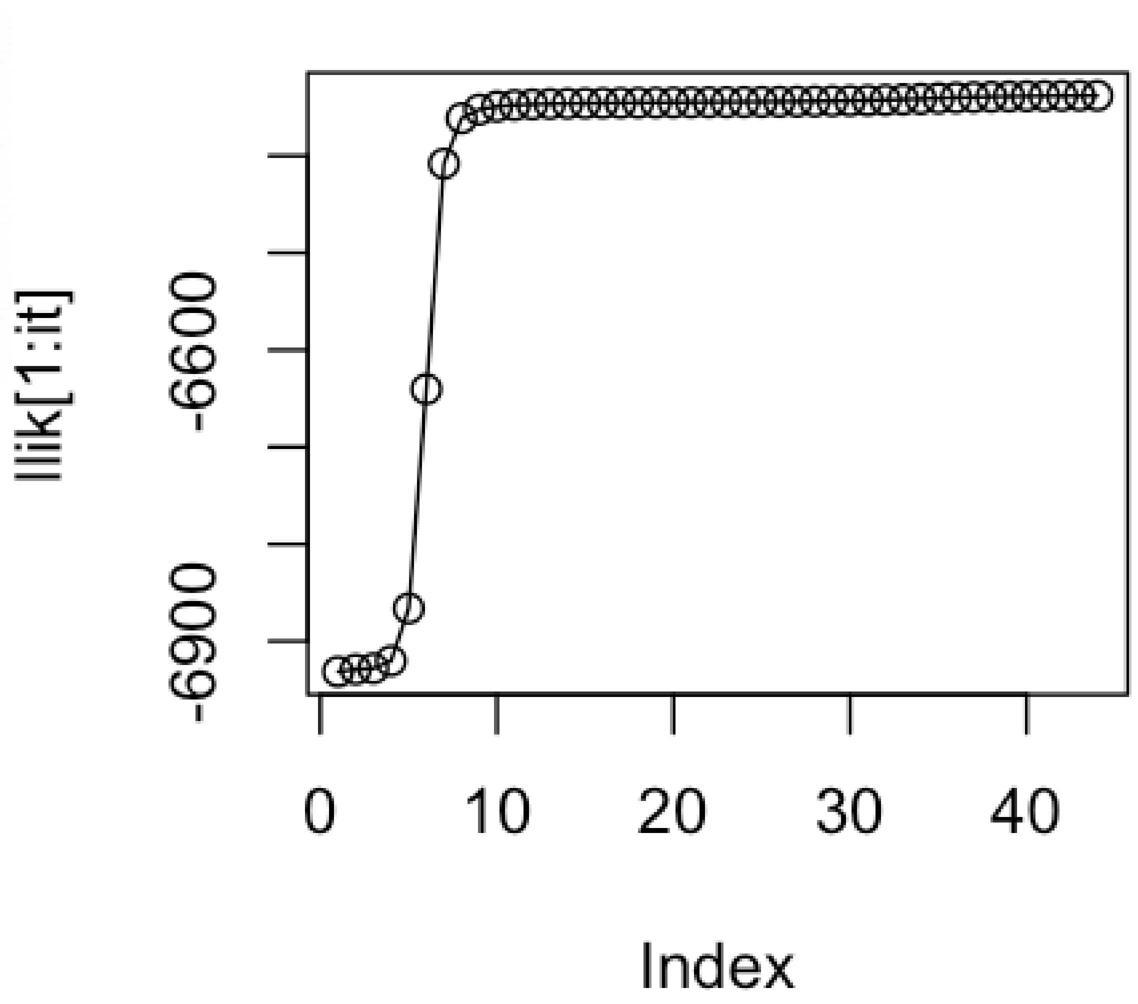

Figure 3: Log-Likelihood Plot for M=4

While the log-likelihood improves as the number of clusters increases, the additional clusters for $M = 4$ do not significantly enhance the fit, indicating overfitting. The ideal choice of $M = 3$ aligns with the true distribution and balances model complexity and fit quality.

Assignment 3

In an ensemble model, is it true that the larger the number B of ensemble members the more flexible the ensemble model?

No, increasing the number B of ensemble members does not make the model more flexible. Instead, it reduces variance. As stated on **page 169** of the main course book, the flexibility is determined by the base learners, not B.

In AdaBoost, what is the loss function used to train the boosted classifier at each iteration?

In AdaBoost, the loss function used to train the boosted classifier at each iteration is the **exponential loss**, given by:

$$L(y \cdot f(x)) = \exp(-y \cdot f(x))$$

This is discussed on **page 177** of the main course book.

Sketch how you would use cross-validation to select the number of components (or clusters) in unsupervised learning of GMMs.

To use cross-validation for selecting the number of components M in GMMs:

1. Split the data into a **training set** and a **validation set**.
2. Train GMM models on the training set with different numbers of components M.
3. Compute the **log-likelihood** of the validation set for each trained model.
4. Select the model with the **highest log-likelihood** on the validation set as the best choice for M.

This ensures the optimal number of clusters is selected while balancing model flexibility and generalization. This process is explained on **page 267** of the course book.

Appendix

Asssignment 1

```r
library(randomForest)

run_rf_and_compute_error <- function(ntree, nodesize) {

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

  rf_model <- randomForest(trdata, trlabels, ntree = ntree, nodesize = nodesize, keep.forest = TRUE)

  x1_test <- runif(1000)
  x2_test <- runif(1000)
  tedata <- cbind(x1_test, x2_test)
  colnames(tedata) <- colnames(trdata)
  y_test <- as.numeric(x1_test < x2_test)
  telabels <- as.factor(y_test)

  predictions <- predict(rf_model, tedata)
  error_rate <- mean(predictions != telabels)
  return(error_rate)
}

errors_1_tree <- run_rf_and_compute_error(1, 25)
errors_10_trees <- run_rf_and_compute_error(10, 25)
errors_100_trees <- run_rf_and_compute_error(100, 25)

errors_1_tree
```

```
## [1] 0.152
```

```r
errors_10_trees
```

```
## [1] 0.119
```

```r
errors_100_trees
```

```
## [1] 0.16
```

```r
# Function to repeat procedure for 1000 datasets
repeat_rf_for_multiple_datasets <- function(ntree_value, nodesize_value) {
  errors <- numeric(1000)
  for (i in 1:1000) {
    errors[i] <- run_rf_and_compute_error(ntree_value, nodesize_value)
  }
  mean_error <- mean(errors)
  var_error <- var(errors)
  return(c(mean_error, var_error))
}

# Run for 1, 10, and 100 trees
result_1_tree <- repeat_rf_for_multiple_datasets(1, 25)
result_10_trees <- repeat_rf_for_multiple_datasets(10, 25)
result_100_trees <- repeat_rf_for_multiple_datasets(100, 25)

# Display the results for mean and variance of errors
result_1_tree
```

```
## [1] 0.204486000 0.002879141
```

```r
result_10_trees
```

```
## [1] 0.134851000 0.000971202
```

```r
result_100_trees
```

```
## [1] 0.1120180000 0.0007980778
```

```r
run_rf_condition_x1 <- function(ntree, nodesize) {
  x1 <- runif(100)
  x2 <- runif(100)
  trdata <- cbind(x1, x2)
  y <- as.numeric(x1 < 0.5)
  trlabels <- as.factor(y)

  rf_model <- randomForest(trdata, trlabels, ntree = ntree, nodesize = nodesize, keep.forest = TRUE)

  # Generate test data
  x1_test <- runif(1000)
  x2_test <- runif(1000)
  tedata <- cbind(x1_test, x2_test)
  colnames(tedata) <- colnames(trdata)
  y_test <- as.numeric(x1_test < 0.5)
  telabels <- as.factor(y_test)

  # Make predictions and calculate misclassification error
  predictions <- predict(rf_model, tedata)
  error_rate <- mean(predictions != telabels)
  return(error_rate)
}

# Run for 1, 10, and 100 trees
errors_1_tree_x1 <- run_rf_condition_x1(1, 25)
errors_10_trees_x1 <- run_rf_condition_x1(10, 25)
errors_100_trees_x1 <- run_rf_condition_x1(100, 25)

# Display the error rates for different numbers of trees
errors_1_tree_x1
```

```
## [1] 0.474
```

```
errors_10_trees_x1
```

```
## [1] 0.019
```

```
errors_100_trees_x1
```

```
## [1] 0.006
```

```
# Function to run random forest with more complex condition
run_rf_complex_condition <- function(ntree_value, nodesize_value) {

  # Generate training data with the complex condition
  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)

  # Train the random forest
  rf_model <- randomForest(trdata, trlabels, ntree = ntree_value, nodesize = nodesize_value, keep.fores

  # Generate test data
  x1_test <- runif(1000)
  x2_test <- runif(1000)
```

```r
  tedata <- cbind(x1_test, x2_test)
  colnames(tedata) <- colnames(trdata)
  y_test <- as.numeric((x1_test < 0.5 & x2_test < 0.5) | (x1_test > 0.5 & x2_test > 0.5))
  telabels <- as.factor(y_test)

  # Make predictions and calculate misclassification error
  predictions <- predict(rf_model, tedata)
  error_rate <- mean(predictions != telabels)
  return(error_rate)
}

# Run for 1, 10, and 100 trees with nodesize = 12
errors_1_tree_complex <- run_rf_complex_condition(1, 12)
errors_10_trees_complex <- run_rf_complex_condition(10, 12)
errors_100_trees_complex <- run_rf_complex_condition(100, 12)

# Display the error rates for different numbers of trees
errors_1_tree_complex
```

```
## [1] 0.319
```

```r
errors_10_trees_complex
```

```
## [1] 0.111
```

```r
errors_100_trees_complex
```

```
## [1] 0.051
```

Assignment 2

```r
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood 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])
  }
}
```

```r
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

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
  for(data_point in 1:n) {
    for(cluster in 1:M) {
      cluster_probability <- pi[cluster]
      for(feature in 1:D) {
        cluster_probability <- cluster_probability * (mu[cluster, feature]^x[data_point, feature]) *
          ((1 - mu[cluster, feature])^(1 - x[data_point, feature]))
      }
      w[data_point, cluster] <- cluster_probability
    }
    w[data_point, ] <- w[data_point, ] / sum(w[data_point, ])
  }

  # Log likelihood computation
  llik[it] <- 0
  for(data_point in 1:n) {
    total_probability <- 0
    for(cluster in 1:M) {
      cluster_probability <- pi[cluster]
      for(feature in 1:D) {
        cluster_probability <- cluster_probability * (mu[cluster, feature]^x[data_point, feature]) *
          ((1 - mu[cluster, feature])^(1 - x[data_point, feature]))
      }
      total_probability <- total_probability + cluster_probability
    }
    llik[it] <- llik[it] + log(total_probability)
  }

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

  # Stop if the log 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
  for(cluster in 1:M) {
    pi[cluster] <- sum(w[, cluster]) / n
    for(feature in 1:D) {
      mu[cluster, feature] <- sum(w[, cluster] * x[, feature]) / sum(w[, cluster])
    }
  }
}

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