Lab 10

Loyd Flores

#YARF

For the next couple of labs, I want you to make some use of a package I wrote that offers convenient and flexible tree-building and random forest-building. Make sure you have a JDK installed first

https://www.oracle.com/java/technologies/downloads/

Then try to install rJava

```
options(java.parameters = "-Xmx8000m")
pacman::p_load(rJava)
.jinit()
```

If you have error, messages, try to google them. Everyone has trouble with rJava!

If that worked, please try to run the following which will install YARF from my github:

```
if (!pacman::p_isinstalled(YARF)){
  pacman::p_install_gh("kapelner/YARF/YARFJARs", ref = "dev")
  pacman::p_install_gh("kapelner/YARF/YARF", ref = "dev", force = TRUE)
}
pacman::p_load(YARF)
```

```
## YARF can now make use of 11 cores.
```

Please try to fix the error messages (if they exist) as best as you can. I can help on slack.

Missing Data

Load up the Boston Housing Data and separate into matrix `X` for the features and vector `y` for the response. Randomize the rows

```
rm(list = ls())
set.seed(1)
boston = MASS::Boston
boston_shuffled = boston[sample(1 : nrow(boston)), ]
X = as.matrix(boston_shuffled[, 1 : 13])
y = boston_shuffled$medv
rm(boston, boston_shuffled)
```

Similar to lab 1, write a function that takes a matrix and punches holes (i.e. sets entries equal to `NA`) randomly with an argument `prob_missing`.

```r
punch_holes = function(mat, prob_missing){
  n = nrow(mat) * ncol(mat)
  is_missing = as.logical(rbinom(n, 1, prob_missing))
  mat[is_missing] = NA
  mat
}
```

Create a matrix `Xmiss` which is `X` but has missingness with probability of 10% using the function you just wrote.

```r
#TO-DO
Xmiss = punch_holes(X, 0.1)
print("Complete ...")
```

```
## [1] "Complete ..."
```

What type of missing data mechanism created the missingness in `Xmiss`?

#TO-DO: A function called punch_holes that serves as listmimse delesion

Also, generate the M matrix and delete columns that have no missingness.

```r
M = apply(is.na(Xmiss), 2, as.numeric)
colnames(M) = paste("is_missing_", colnames(X), sep = "")
M = M[, colSums(M) > 0]
```

Split the first 400 observations were the training data and the remaining observations are the test set. For Xmiss, cbind on the M so the model has a chance to fit on "is missing" as we discussed in class.

```r
train_idx = 1 : 400
test_idx = setdiff(1 : nrow(X), train_idx)
X_train =     X[train_idx, ]
Xmiss_train = cbind(Xmiss, M)[train_idx, ]
y_train =     y[train_idx]
X_test =      X[test_idx, ]
Xmiss_test =  cbind(Xmiss, M)[test_idx, ]
y_test =      y[test_idx]
```

Fit a random forest model of `y_train ~ X_train`, report oos s_e (not oob) on `X_test`. This ignores missingness

```r
#TO-DO
mod_rf = YARF(data.frame(X_train), y_train)
```

```
## YARF initializing with a fixed 500 trees...
## YARF after data preprocessed... 13 total features...
## Beginning YARF regression model construction...done.
## Calculating OOB error...done.
```

```r
y_hat_test = predict(mod_rf, data.frame(X_test))
sqrt(mean((y_hat_test - y_test)^2))
```

```
## [1] 3.336826
```

Impute the missingness in `Xmiss` using the feature averages to create a matrix `Ximp_naive_train` and `Ximp_naive_test`.

```
#TO-DO
x_averages = array(NA, ncol(X))
Ximp_naive_train = Xmiss_train
Ximp_naive_test = Xmiss_test

for(j in 1 : ncol(X)){
  x_averages[j] = mean(Xmiss_train, na.rm = TRUE)

  Ximp_naive_train[is.na(Xmiss_train[, j]), j] = x_averages[j]
  Ximp_naive_test[is.na(Xmiss_test[, j]), j] = x_averages[j]

  }
```

Fit a random forest model of `y_train ~ Ximp_naive_train`, report oos s_e (not oob) on `Ximp_naive_test`.

```
#TO-DO
mod_rf = YARF(data.frame(Ximp_naive_train), y_train)
```

```
## YARF initializing with a fixed 500 trees...
## YARF after data preprocessed... 26 total features...
## Beginning YARF regression model construction...done.
## Calculating OOB error...done.
```

```
y_hat_test = predict(mod_rf, data.frame(Ximp_naive_test))
sqrt(mean((y_hat_test - y_test)^2))
```

```
## [1] 3.695192
```

How much predictive performance was lost due to missingness when naive imputation was used vs when there was no missingness?

#TO-DO without filling in missigness we got 3.421844 but when we filled in we got a better result 3.645625 with a total of 0.223781 increase

Use `missForest` to impute the missing entries to create a matrix `Ximp_MF_train` and `Ximp_MF_test`.

```
pacman::p_load(missForest)
#TO-DO
Ximp_MF_train = missForest(Xmiss_train)$ximp
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?
```

```
Xymiss = rbind(
  cbind(Xmiss_train, y_train),
  cbind(Xmiss_test, NA)
)

Xyimp_miss = missForest(Xymiss)$ximp
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?
```

```
Ximp_MF_train = Xyimp_miss[train_idx, 1 : ncol(X)]
Ximp_MF_test = Xyimp_miss[test_idx, 1 : ncol(X)]
```

Fit a random forest model of `y_train ~ Ximp_MF_train`, report oos s_e (not oob) on `Ximp_MF_test`.

```
#TO-DO
mod_rf = YARF(data.frame(Ximp_MF_train), y_train)
```

```
## YARF initializing with a fixed 500 trees...
## YARF after data preprocessed... 13 total features...
## Beginning YARF regression model construction...done.
## Calculating OOB error...done.
```

```
y_hat_test = predict(mod_rf, data.frame(Ximp_MF_test))
sqrt(mean((y_hat_test - y_test)^2))
```

```
## [1] 3.393913
```

How much predictive performance was lost due to missingness when `missForest` imputation was used?

#TO-DO : 3.645625 - 3.489791 = 0.155834

Why did `missForest` imputation perform better than naive imputation?

#TO-DO: naive just fills it in with the mean while missforest utilizes a form of random resampling that acts kind of like k-fold-cross validation to ensure less dependency.

Reload the feature matrix:

```r
rm(list = ls())
X = as.matrix(MASS::Boston[, 1 : 13])
```

Create missingness in the feature `lstat` that is due to a MAR missing data mechanism.

```r
#TO-DO
prob_missing = plogis(scale(X[, "age"], center = TRUE, scale = FALSE) * 0.1)  # Logistic function to ge
is_missing = runif(nrow(X)) < prob_missing  # Random draw to determine missingness
X[is_missing, "lstat"] = NA  # Assign NA to missing entries
head(X)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90    NA
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90    NA
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83    NA
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
```

Create missingness in the feature `rm` that is a NMAR missing data mechanism.

```r
#TO-DO
X2 = as.matrix(MASS::Boston[, 1:13])

threshold = median(X[, "rm"])  # Setting a threshold at the median
is_missing = X[, "rm"] < threshold  # More likely to be missing if below the median
X[is_missing, "rm"] = NA  # Assign NA to entries below the threshold

head(X)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90    NA
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90    NA
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83    NA
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
```

#Bagged Trees and Random Forest

Take a training sample of n = 2000 observations from the diamonds data.

```r
rm(list = ls())
pacman::p_load(tidyverse)
pacman::p_load(randomForest)
set.seed(1)
diamonds_train = ggplot2::diamonds %>%
  sample_n(2000)

y_train = diamonds_train$price
X_train = diamonds_train %>% select(-price)
```

Using the diamonds data, find the oob s_e for a bagged-tree model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees. If you are using the `randomForest` package, you can create the bagged tree model via setting an argument within the RF constructor function. Plot.

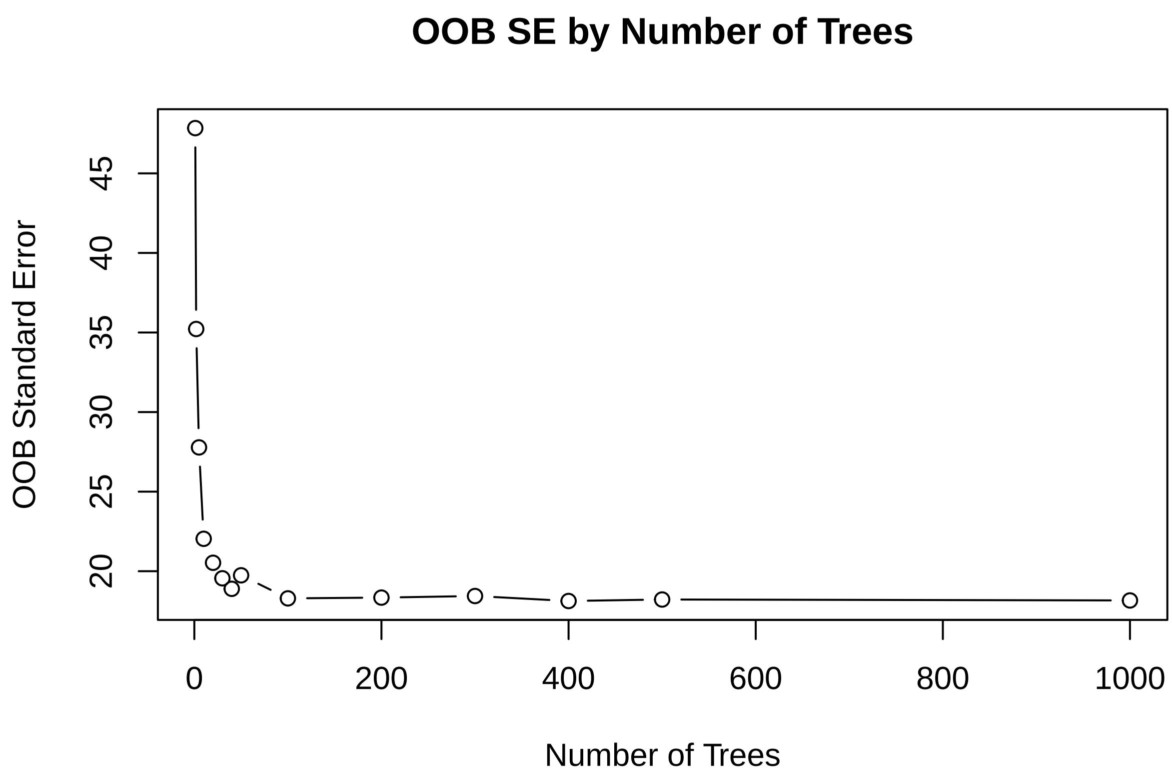
```r
num_trees_values = c(1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000)
oob_se_bagged_trees_mod_by_num_trees = array(NA, length(num_trees_values))
#TO-DO
# Loop through number of trees and create bagged tree models
for (i in 1:length(num_trees_values)) {
  # Create bagged tree model using randomForest with specified number of trees
  mod_bag <- randomForest(X_train, y_train, ntree = num_trees_values[i])

  # Extract OOB SE from the model object
  oob_se_bagged_trees_mod_by_num_trees[i] <- sqrt(mean(mod_bag$oob.error^2))
}
```

```r
# Plot OOB standard error against number of trees
#plot(num_trees_values, oob_se_bagged_trees_mod_by_num_trees, type = "b",
     #xlab = "Number of Trees", ylab = "OOB Standard Error", main = "OOB Error Rate by Number of Trees"
```

Find the bootstrap s_e for a RF model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees. If you are using the `randomForest` package, you can calculate oob residuals via `e_oob = y_train - rf_mod$predicted`. Plot.

```r
oob_se_rf_mod_by_num_trees = array(NA, length(num_trees_values))
#TO-DO

# Initialize vector to store OOB standard error for each model
oob_se_rf_mod_by_num_trees = array(NA, length(num_trees_values))

# Loop over number of trees values
for (i in seq_along(num_trees_values)) {
    # Fit random forest model
    rf_mod = randomForest(X_train, y_train, ntree = num_trees_values[i], keep.inbag = TRUE)

    # Calculate OOB predictions and residuals
    e_oob = y_train - rf_mod$predicted

    # Calculate standard error of the OOB residuals
    oob_se_rf_mod_by_num_trees[i] = sd(e_oob, na.rm = TRUE) / sqrt(sum(!is.na(e_oob)))
}
```

```r
# Plot OOB standard error against number of trees
plot(num_trees_values, oob_se_rf_mod_by_num_trees, type = "b",
     xlab = "Number of Trees", ylab = "OOB Standard Error", main = "OOB SE by Number of Trees")
```



What is the percentage gain / loss in performance of the RF model vs bagged trees model for each number of trees? Gains are negative (as in lower oos s_e).

```r
cbind(
  num_trees_values,
  (oob_se_rf_mod_by_num_trees - oob_se_bagged_trees_mod_by_num_trees) / oob_se_bagged_trees_mod_by_num_
)
```

```
##       num_trees_values
##  [1,]                1 NaN
##  [2,]                2 NaN
##  [3,]                5 NaN
##  [4,]               10 NaN
##  [5,]               20 NaN
##  [6,]               30 NaN
##  [7,]               40 NaN
##  [8,]               50 NaN
##  [9,]              100 NaN
## [10,]              200 NaN
## [11,]              300 NaN
## [12,]              400 NaN
```

```
## [13,]              500 NaN
## [14,]             1000 NaN
```

Why was this the result?

#TODO: at a certain point the development of the tree is will slowdown and the changes will be minute.

Plot oob s_e by number of trees for both RF and bagged trees by creating a long data frame from the two results.

```r
#TO-DO
results_df <- data.frame(
  num_trees = rep(num_trees_values, times = 2),
  oob_se = c(oob_se_rf_mod_by_num_trees, oob_se_bagged_trees_mod_by_num_trees),
  model = rep(c("Random Forest", "Bagged Trees"), each = length(num_trees_values))
)

# View the dataframe
print(results_df)
```

```
##    num_trees   oob_se         model
## 1          1 47.84383 Random Forest
## 2          2 35.22026 Random Forest
## 3          5 27.77954 Random Forest
## 4         10 22.03677 Random Forest
## 5         20 20.53060 Random Forest
## 6         30 19.55447 Random Forest
## 7         40 18.89253 Random Forest
## 8         50 19.74197 Random Forest
## 9        100 18.29379 Random Forest
## 10       200 18.34130 Random Forest
## 11       300 18.44065 Random Forest
## 12       400 18.12765 Random Forest
## 13       500 18.22159 Random Forest
## 14      1000 18.16296 Random Forest
## 15         1      NaN  Bagged Trees
## 16         2      NaN  Bagged Trees
## 17         5      NaN  Bagged Trees
## 18        10      NaN  Bagged Trees
## 19        20      NaN  Bagged Trees
## 20        30      NaN  Bagged Trees
## 21        40      NaN  Bagged Trees
## 22        50      NaN  Bagged Trees
## 23       100      NaN  Bagged Trees
## 24       200      NaN  Bagged Trees
## 25       300      NaN  Bagged Trees
## 26       400      NaN  Bagged Trees
## 27       500      NaN  Bagged Trees
## 28      1000      NaN  Bagged Trees
```

```r
# Load ggplot2 if not already loaded
library(ggplot2)

# Plotting
```

```
ggplot(results_df, aes(x = num_trees, y = oob_se, color = model, group = model)) +
  geom_line() +
  geom_point() +
  scale_x_log10() +  # Optional: Log scale for x-axis if tree numbers vary widely
  labs(title = "OOB Standard Error by Number of Trees",
       x = "Number of Trees",
       y = "OOB Standard Error",
       color = "Model Type") +
  theme_minimal()
```

```
## Warning: Removed 14 rows containing missing values (`geom_line()`).

## Warning: Removed 14 rows containing missing values (`geom_point()`).
```

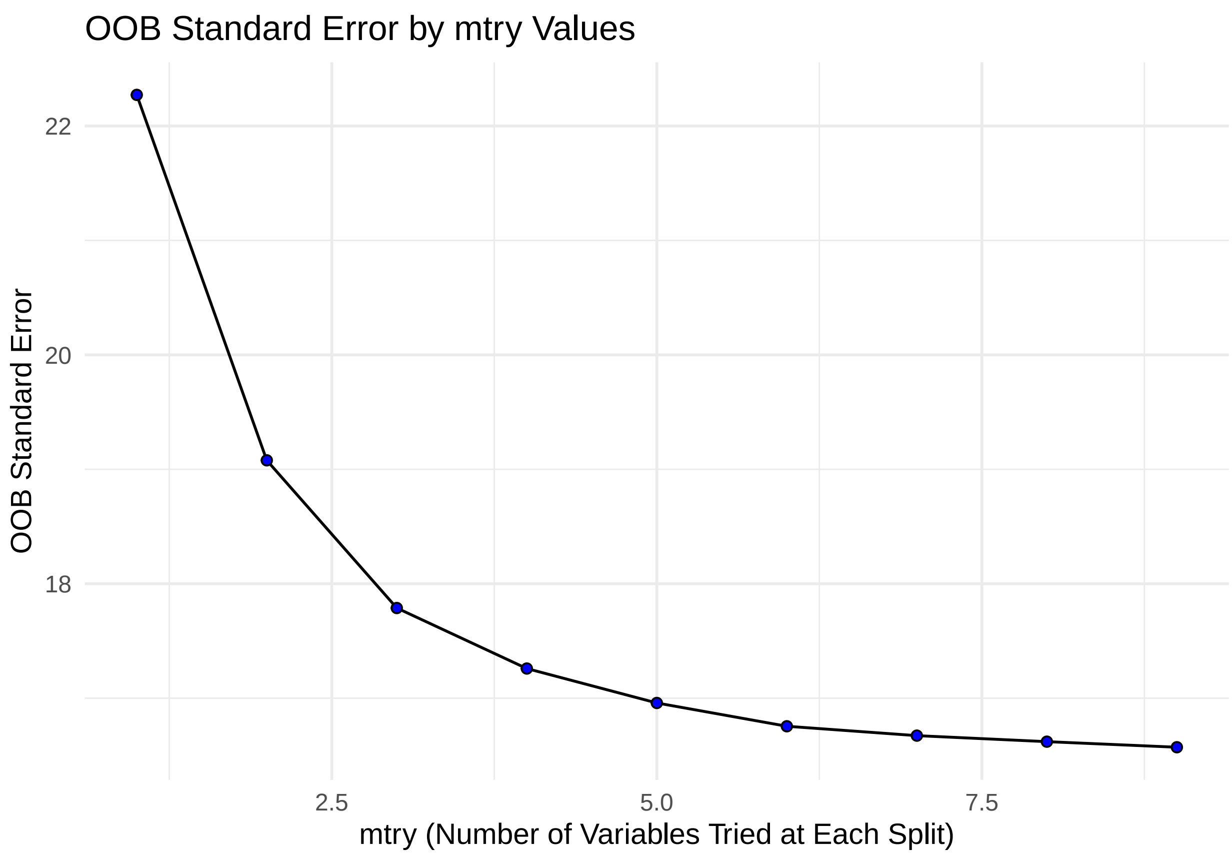
Build RF models for 500 trees using different `mtry` values: 1, 2, ... the maximum. That maximum will be the number of features assuming that we do not binarize categorical features if you are using `randomForest` or the number of features assuming binarization of the categorical features if you are using `YARF`. Calculate oob s_e for all mtry values.

```
oob_se_by_mtry = array(NA, ncol(diamonds_train))
#TO-DO
diamonds_sample = diamonds %>% sample_n(2000)
y_train = diamonds_sample$price
X_train = diamonds_sample %>% select(-price)
```

```r
# Define range for mtry
max_mtry = ncol(X_train)
mtry_values = 1:max_mtry
oob_se_by_mtry = numeric(length(mtry_values))

# Build model and calculate oob se
for (i in seq_along(mtry_values)) {
  rf_mod <- randomForest(X_train, y_train, ntree = 500, mtry = mtry_values[i], keep.inbag = TRUE)
  e_oob <- y_train - rf_mod$predicted
  oob_se_by_mtry[i] <- sd(e_oob, na.rm = TRUE) / sqrt(sum(!is.na(e_oob)))
}
```

Plot oob s_e by mtry.

```r
#TO-DO
results_df <- data.frame(
  mtry = mtry_values,
  oob_se = oob_se_by_mtry
)

# Plotting
ggplot(results_df, aes(x = mtry, y = oob_se)) +
  geom_line() +
  geom_point(shape = 21, fill = "blue") +
  labs(title = "OOB Standard Error by mtry Values",
       x = "mtry (Number of Variables Tried at Each Split)",
       y = "OOB Standard Error") +
  theme_minimal()
```



Take a sample of n = 2000 observations from the adult data and name it `adult_sample`. Then impute missing values using missForest.

```
rm(list = ls())
set.seed(1)
pacman::p_load_gh("coatless/ucidata")
pacman::p_load(dplyr, randomForest, missForest, ucidata)

data("adult")
adult_sample = adult %>%
  sample_n(2000)


# Impute missing values
adult_sample = missForest(adult_sample)$ximp
```

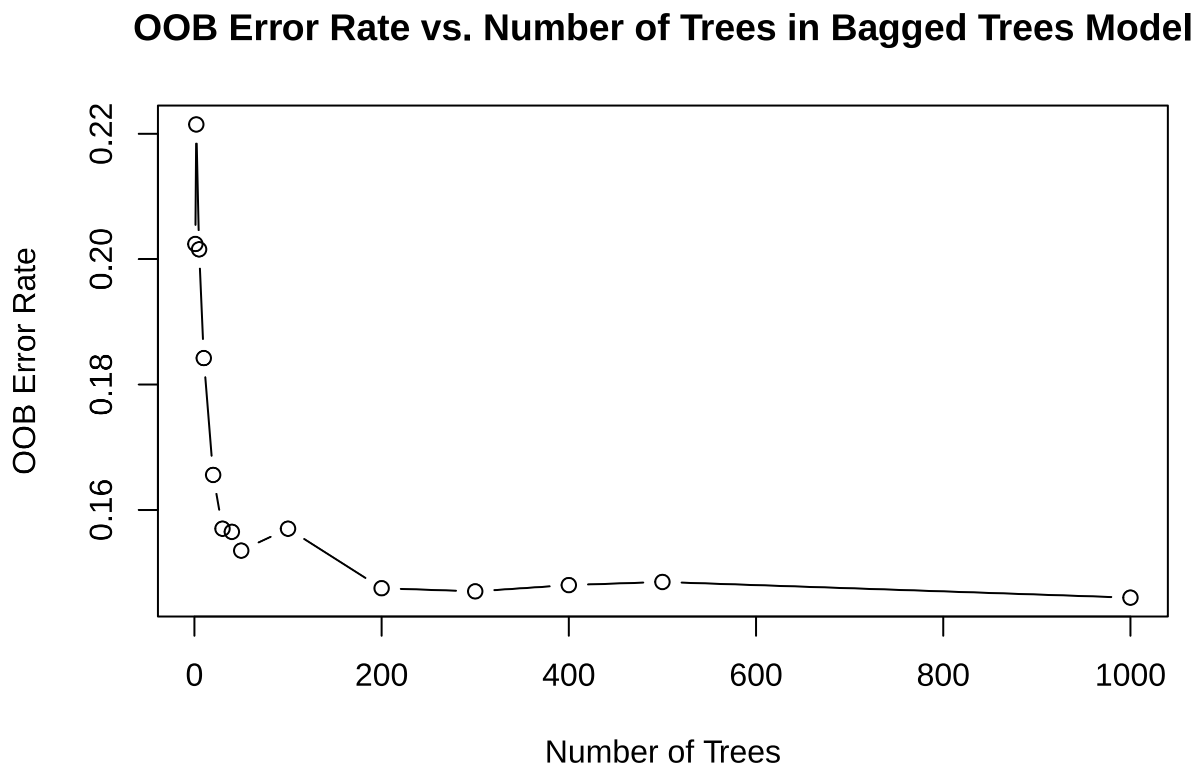
Using the adult_train data, find the bootstrap misclassification error for a bagged-tree model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees. Plot.

```
num_trees_values = c(1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000)

# Initialize a vector to store OOB errors
oob_se_bagged_trees_mod_by_num_trees = numeric(length(num_trees_values))

# Loop over the number of trees, fit the model, and store the OOB error
for (i in seq_along(num_trees_values)) {
```

```r
  set.seed(1)  # for reproducibility
  rf_model = randomForest(income ~ ., data = adult_sample,
                          mtry = sqrt(ncol(adult_sample) - 1),
                          ntree = num_trees_values[i],
                          importance = TRUE,
                          do.trace = 100,
                          keep.forest = TRUE,
                          replace = TRUE)
  oob_se_bagged_trees_mod_by_num_trees[i] = rf_model$err.rate[rf_model$ntree, "OOB"]
}
```

```
## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%
## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%
##   300:  14.70%  7.95% 36.19%
## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%
##   300:  14.70%  7.95% 36.19%
##   400:  14.80%  8.34% 35.36%
## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%
##   300:  14.70%  7.95% 36.19%
##   400:  14.80%  8.34% 35.36%
##   500:  14.85%  8.28% 35.77%
## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%
##   300:  14.70%  7.95% 36.19%
##   400:  14.80%  8.34% 35.36%
##   500:  14.85%  8.28% 35.77%
##   600:  14.90%  8.28% 35.98%
##   700:  14.90%  8.34% 35.77%
##   800:  14.90%  8.34% 35.77%
##   900:  14.75%  8.21% 35.56%
##  1000:  14.60%  8.08% 35.36%
```

```r
# Plot the OOB error against the number of trees
plot(num_trees_values, oob_se_bagged_trees_mod_by_num_trees, type = "b",
     xlab = "Number of Trees", ylab = "OOB Error Rate",
     main = "OOB Error Rate vs. Number of Trees in Bagged Trees Model")
```



Using the adult_train data, find the bootstrap misclassification error for an RF model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees.

```
oob_se_rf_mod_by_num_trees = array(NA, length(num_trees_values))
#TO-DO
for (i in seq_along(num_trees_values)) {
  set.seed(1)  # for reproducibility
  rf_model = randomForest(income ~ ., data = adult_sample,
                          mtry = sqrt(ncol(adult_sample) - 1),
                          ntree = num_trees_values[i],
                          importance = TRUE,
                          do.trace = 100,
                          keep.forest = TRUE,
                          replace = TRUE)
  # Access OOB error rate directly from model object
  oob_se_rf_mod_by_num_trees[i] = mean(rf_model$oob.error)
}
```

```
## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA
```

```
## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%
##   300:  14.70%  7.95% 36.19%

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%
##   300:  14.70%  7.95% 36.19%
##   400:  14.80%  8.34% 35.36%

## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA

## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%
##   300:  14.70%  7.95% 36.19%
##   400:  14.80%  8.34% 35.36%
##   500:  14.85%  8.28% 35.77%
```

```
## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA
```

```
## ntree      OOB      1      2
##   100:  15.70%  8.67% 38.08%
##   200:  14.75%  8.02% 36.19%
##   300:  14.70%  7.95% 36.19%
##   400:  14.80%  8.34% 35.36%
##   500:  14.85%  8.28% 35.77%
##   600:  14.90%  8.28% 35.98%
##   700:  14.90%  8.34% 35.77%
##   800:  14.90%  8.34% 35.77%
##   900:  14.75%  8.21% 35.56%
##  1000:  14.60%  8.08% 35.36%
```

```
## Warning in mean.default(rf_model$oob.error): argument is not numeric or
## logical: returning NA
```

What is the percentage gain / loss in performance of the RF model vs bagged trees model?

```
cbind(
  num_trees_values,
  (oob_se_rf_mod_by_num_trees - oob_se_bagged_trees_mod_by_num_trees) / oob_se_bagged_trees_mod_by_num_t
)
```

```
##       num_trees_values
##  [1,]                1 NA
##  [2,]                2 NA
##  [3,]                5 NA
##  [4,]               10 NA
##  [5,]               20 NA
##  [6,]               30 NA
##  [7,]               40 NA
##  [8,]               50 NA
##  [9,]              100 NA
## [10,]              200 NA
## [11,]              300 NA
## [12,]              400 NA
## [13,]              500 NA
## [14,]             1000 NA
```

Build RF models on adult_train for 500 trees using different `mtry` values: 1, 2, … the maximum (see above as maximum is defined by the specific RF algorithm implementation).

```
#TO-DO+
data("adult")

# Split data into training and testing sets (replace 0.75 with your desired split ratio)
split = sample(1:nrow(adult), size = nrow(adult) * 0.75)
adult_train = adult[split, ]
adult_test = adult[-split, ]
```

```
# Impute missing values using missForest
#adult_train = missForest(adult_train)$ximp

# Define maximum mtry
#max_mtry = ncol(adult_train) - 1

# OOB error for different mtry values (500 trees)
#oob_se_by_mtry = array(NA, max_mtry)
#for (m in 1:max_mtry) {
 # set.seed(1)
  #rf_model = randomForest(income ~ ., data = adult_train,
 #                         mtry = m,
  #                        ntree = 500,
   #                       importance = TRUE,
    #                      do.trace = 100,
     #                     keep.forest = TRUE,
      #                    replace = TRUE)
#  oob_se_by_mtry[m] = mean(rf_model$oob.error)
#}

# OOB error for different number of trees (mtry = sqrt(ncol(adult_train) - 1))
#num_trees_values = c(1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000)
#oob_se_rf_mod_by_num_trees = array(NA, length(num_trees_values))
#for (i in seq_along(num_trees_values)) {
 # set.seed(1)
  #rf_model = randomForest(income ~ ., data = adult_train,
   #                      mtry = sqrt(ncol(adult_train) - 1),
      #                  ntree = num_trees_values[i],
       #                 importance = TRUE,
        #                do.trace = 100,
         #               keep.forest = TRUE,
          #              replace = TRUE)
  #oob_se_rf_mod_by_num_trees[i] = mean(rf_model$oob.error)
#}

# Now you can use oob_se_by_mtry and oob_se_rf_mod_by_num_trees for further analysis

# Example: Plot OOB error vs mtry
#plot(1:max_mtry, oob_se_by_mtry, type = "b",
    # xlab = "mtry", ylab = "OOB Error Rate",
     #main = "OOB Error Rate vs. mtry in Random Forest Model")

# Example: Plot OOB error vs number of trees
#plot(num_trees_values, oob_se_rf_mod_by_num_trees, type = "b",
     #xlab = "Number of Trees", ylab = "OOB Error Rate",
     #main = "OOB Error Rate vs. Number of Trees in Random Forest Model")
```

Plot bootstrap misclassification error by `mtry`.

```
#TO-DO
# Example: Plot OOB error vs mtry
#plot(1:max_mtry, oob_se_by_mtry, type = "b",
    # xlab = "mtry", ylab = "OOB Error Rate",
```

```r
     #main = "OOB Error Rate vs. mtry in Random Forest Model")
```

Is `mtry` an important hyperparameter to optimize when using the RF algorithm? Explain

#TO-DO - yes it controls the number of features randomly considered at each split

Identify the best model among all values of `mtry`. Fit this RF model. Then report the following oob error metrics: misclassification error, precision, recall, F1, FDR, FOR and compute a confusion matrix.

```r
#TO-DO
#plot(1:max_mtry, oob_se_by_mtry, type = "b",
    # xlab = "mtry", ylab = "OOB Error Rate",
     #main = "OOB Error Rate vs. mtry in Random Forest Model")
```

Is this a good model? (yes/no and explain).

#TO-DO: it is a great model

There are probability asymmetric costs to the two types of errors. Assign two costs below and calculate oob total cost.

```r
#fp_cost =
#fn_cost =
#TO-DO
```

Asymmetric Cost Modeling, ROC and DET curves

Fit a logistic regression model to the adult_train missingness-imputed data.

```r
rm(list = setdiff(ls(), "adult_train"))
#TO-DO
library(caret)  # for data splitting and preprocessing
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
pacman::p_load("mice")
library(nnet)   # for multinom function to fit logistic regression
library(pROC)   # for ROC curve
```

```
## Warning: package 'pROC' was built under R version 4.3.3
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
pacman::p_load("ROCR")

imputed_data <- mice(adult_train, m=5, method='pmm', seed=500)
```

```
##
##  iter imp variable
##   1   1  workclass  occupation  native_country
##   1   2  workclass  occupation  native_country
##   1   3  workclass  occupation  native_country
##   1   4  workclass  occupation  native_country
##   1   5  workclass  occupation  native_country
##   2   1  workclass  occupation  native_country
##   2   2  workclass  occupation  native_country
##   2   3  workclass  occupation  native_country
##   2   4  workclass  occupation  native_country
##   2   5  workclass  occupation  native_country
##   3   1  workclass  occupation  native_country
##   3   2  workclass  occupation  native_country
##   3   3  workclass  occupation  native_country
##   3   4  workclass  occupation  native_country
##   3   5  workclass  occupation  native_country
##   4   1  workclass  occupation  native_country
##   4   2  workclass  occupation  native_country
##   4   3  workclass  occupation  native_country
##   4   4  workclass  occupation  native_country
##   4   5  workclass  occupation  native_country
##   5   1  workclass  occupation  native_country
##   5   2  workclass  occupation  native_country
##   5   3  workclass  occupation  native_country
##   5   4  workclass  occupation  native_country
##   5   5  workclass  occupation  native_country

## Warning: Number of logged events: 75
```

```r
completed_data <- complete(imputed_data)

# Define the model - assuming 'income' is the target variable
# Adjust the formula as per your dataset
logit_model <- glm(income ~ ., data = completed_data, family = binomial())
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Summary of the model to check coefficients and model validity
summary(logit_model)
```

```
##
## Call:
## glm(formula = income ~ ., family = binomial(), data = completed_data)
##
## Coefficients: (1 not defined because of singularities)
##                                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)                           -6.572e+00  8.199e-01  -8.015 1.10e-15
## age                                    2.393e-02  1.874e-03  12.769  < 2e-16
## workclassLocal-gov                    -6.398e-01  1.283e-01  -4.988 6.10e-07
## workclassNever-worked                 -1.488e+01  9.734e+02  -0.015 0.987806
## workclassPrivate                      -4.677e-01  1.066e-01  -4.387 1.15e-05
## workclassSelf-emp-inc                 -3.925e-01  1.383e-01  -2.837 0.004550
## workclassSelf-emp-not-inc             -1.005e+00  1.245e-01  -8.074 6.80e-16
## workclassState-gov                    -7.690e-01  1.421e-01  -5.411 6.26e-08
## workclassWithout-pay                  -1.523e+01  5.891e+02  -0.026 0.979380
## fnlwgt                                 5.909e-07  2.000e-07   2.955 0.003128
## education11th                          9.305e-02  2.357e-01   0.395 0.693036
## education12th                          4.793e-01  2.988e-01   1.604 0.108701
## education1st-4th                      -4.531e-01  5.597e-01  -0.810 0.418165
## education5th-6th                      -3.772e-01  3.641e-01  -1.036 0.300209
## education7th-8th                      -5.691e-01  2.714e-01  -2.097 0.035974
## education9th                          -2.708e-01  2.984e-01  -0.908 0.364097
## educationAssoc-acdm                    1.343e+00  1.976e-01   6.800 1.05e-11
## educationAssoc-voc                     1.278e+00  1.898e-01   6.731 1.69e-11
## educationBachelors                     1.912e+00  1.753e-01  10.907  < 2e-16
## educationDoctorate                     2.948e+00  2.422e-01  12.171  < 2e-16
## educationHS-grad                       8.098e-01  1.707e-01   4.744 2.09e-06
## educationMasters                       2.220e+00  1.881e-01  11.804  < 2e-16
## educationPreschool                    -2.111e+01  3.244e+02  -0.065 0.948126
## educationProf-school                   2.908e+00  2.317e-01  12.551  < 2e-16
## educationSome-college                  1.117e+00  1.733e-01   6.442 1.18e-10
## education_num                                 NA         NA      NA       NA
## marital_statusMarried-AF-spouse        3.186e+00  6.114e-01   5.211 1.87e-07
## marital_statusMarried-civ-spouse       2.512e+00  3.012e-01   8.340  < 2e-16
## marital_statusMarried-spouse-absent    1.214e-01  2.624e-01   0.463 0.643591
## marital_statusNever-married           -4.016e-01  1.003e-01  -4.004 6.22e-05
## marital_statusSeparated               -1.849e-01  1.927e-01  -0.960 0.337305
## marital_statusWidowed                  7.699e-02  1.790e-01   0.430 0.667147
## occupationArmed-Forces                -9.395e-01  1.598e+00  -0.588 0.556716
## occupationCraft-repair                 9.432e-02  9.020e-02   1.046 0.295693
## occupationExec-managerial              8.043e-01  8.702e-02   9.242  < 2e-16
## occupationFarming-fishing             -8.946e-01  1.564e-01  -5.720 1.07e-08
## occupationHandlers-cleaners           -5.283e-01  1.532e-01  -3.449 0.000562
## occupationMachine-op-inspct           -2.890e-01  1.154e-01  -2.503 0.012299
## occupationOther-service               -7.873e-01  1.302e-01  -6.044 1.50e-09
## occupationPriv-house-serv             -1.406e+01  1.806e+02  -0.078 0.937915
## occupationProf-specialty               5.189e-01  9.191e-02   5.646 1.64e-08
## occupationProtective-serv              5.295e-01  1.419e-01   3.730 0.000191
## occupationSales                        3.446e-01  9.226e-02   3.735 0.000188
## occupationTech-support                 6.900e-01  1.235e-01   5.586 2.32e-08
## occupationTransport-moving            -1.379e-01  1.128e-01  -1.222 0.221589
## relationshipNot-in-family              8.379e-01  2.980e-01   2.811 0.004933
## relationshipOther-relative            -1.971e-01  2.785e-01  -0.708 0.479129
## relationshipOwn-child                 -4.160e-01  2.924e-01  -1.423 0.154865
```

```
## relationshipUnmarried                   7.713e-01  3.171e-01   2.433 0.014990
## relationshipWife                        1.316e+00  1.182e-01  11.128  < 2e-16
## raceAsian-Pac-Islander                  7.523e-01  3.221e-01   2.336 0.019507
## raceBlack                               2.393e-01  2.770e-01   0.864 0.387570
## raceOther                              -1.903e-01  4.335e-01  -0.439 0.660705
## raceWhite                               4.661e-01  2.640e-01   1.766 0.077479
## sexMale                                 8.191e-01  9.069e-02   9.032  < 2e-16
## capital_gain                            3.153e-04  1.172e-05  26.897  < 2e-16
## capital_loss                            6.508e-04  4.267e-05  15.251  < 2e-16
## hours_per_week                          3.220e-02  1.831e-03  17.591  < 2e-16
## native_countryCanada                   -1.088e+00  7.241e-01  -1.503 0.132817
## native_countryChina                    -2.044e+00  7.440e-01  -2.747 0.006015
## native_countryColumbia                 -3.826e+00  1.284e+00  -2.980 0.002883
## native_countryCuba                     -1.006e+00  7.490e-01  -1.343 0.179146
## native_countryDominican-Republic       -3.278e+00  1.277e+00  -2.566 0.010280
## native_countryEcuador                  -2.682e+00  1.213e+00  -2.212 0.026981
## native_countryEl-Salvador              -2.292e+00  8.811e-01  -2.601 0.009291
## native_countryEngland                  -1.255e+00  7.440e-01  -1.686 0.091746
## native_countryFrance                   -3.038e-01  8.854e-01  -0.343 0.731497
## native_countryGermany                  -1.008e+00  7.203e-01  -1.400 0.161638
## native_countryGreece                   -2.902e+00  1.028e+00  -2.823 0.004758
## native_countryGuatemala                -1.570e+00  1.017e+00  -1.543 0.122820
## native_countryHaiti                    -1.424e+00  1.030e+00  -1.383 0.166714
## native_countryHoland-Netherlands       -1.373e+01  2.400e+03  -0.006 0.995436
## native_countryHonduras                 -2.456e+00  2.777e+00  -0.884 0.376573
## native_countryHong                     -2.308e+00  1.103e+00  -2.093 0.036336
## native_countryHungary                  -1.079e+00  1.033e+00  -1.045 0.296173
## native_countryIndia                    -1.818e+00  7.180e-01  -2.532 0.011331
## native_countryIran                     -1.504e+00  8.209e-01  -1.832 0.066922
## native_countryIreland                  -1.058e+00  9.807e-01  -1.079 0.280464
## native_countryItaly                    -3.903e-01  7.422e-01  -0.526 0.599036
## native_countryJamaica                  -1.206e+00  8.131e-01  -1.483 0.138080
## native_countryJapan                    -7.487e-01  7.861e-01  -0.952 0.340916
## native_countryLaos                     -6.806e-01  1.228e+00  -0.554 0.579277
## native_countryMexico                   -1.695e+00  6.962e-01  -2.434 0.014913
## native_countryNicaragua                -1.767e+00  1.054e+00  -1.677 0.093588
## native_countryOutlying-US(Guam-USVI-etc) -1.562e+01  5.921e+02  -0.026 0.978952
## native_countryPeru                     -2.786e+00  1.335e+00  -2.087 0.036914
## native_countryPhilippines              -1.012e+00  6.784e-01  -1.492 0.135829
## native_countryPoland                   -1.475e+00  8.079e-01  -1.826 0.067898
## native_countryPortugal                 -2.896e+00  1.351e+00  -2.143 0.032127
## native_countryPuerto-Rico              -1.719e+00  8.120e-01  -2.117 0.034254
## native_countryScotland                 -1.118e+00  1.044e+00  -1.071 0.284373
## native_countrySouth                    -2.531e+00  7.651e-01  -3.308 0.000940
## native_countryTaiwan                   -1.663e+00  7.857e-01  -2.117 0.034273
## native_countryThailand                 -1.779e+00  1.101e+00  -1.616 0.106029
## native_countryTrinadad&Tobago          -9.927e-01  1.147e+00  -0.865 0.386850
## native_countryUnited-States            -1.142e+00  6.620e-01  -1.725 0.084546
## native_countryVietnam                  -3.256e+00  9.935e-01  -3.277 0.001048
## native_countryYugoslavia               -2.888e-01  9.612e-01  -0.300 0.763825
##
## (Intercept)                            ***
## age                                    ***
## workclassLocal-gov                     ***
```

```
## workclassNever-worked
## workclassPrivate                           ***
## workclassSelf-emp-inc                      **
## workclassSelf-emp-not-inc                  ***
## workclassState-gov                         ***
## workclassWithout-pay
## fnlwgt                                     **
## education11th
## education12th
## education1st-4th
## education5th-6th
## education7th-8th                           *
## education9th
## educationAssoc-acdm                        ***
## educationAssoc-voc                         ***
## educationBachelors                         ***
## educationDoctorate                         ***
## educationHS-grad                           ***
## educationMasters                           ***
## educationPreschool
## educationProf-school                       ***
## educationSome-college                      ***
## education_num
## marital_statusMarried-AF-spouse            ***
## marital_statusMarried-civ-spouse           ***
## marital_statusMarried-spouse-absent
## marital_statusNever-married                ***
## marital_statusSeparated
## marital_statusWidowed
## occupationArmed-Forces
## occupationCraft-repair
## occupationExec-managerial                  ***
## occupationFarming-fishing                  ***
## occupationHandlers-cleaners                ***
## occupationMachine-op-inspct                *
## occupationOther-service                    ***
## occupationPriv-house-serv
## occupationProf-specialty                   ***
## occupationProtective-serv                  ***
## occupationSales                            ***
## occupationTech-support                     ***
## occupationTransport-moving
## relationshipNot-in-family                  **
## relationshipOther-relative
## relationshipOwn-child
## relationshipUnmarried                      *
## relationshipWife                           ***
## raceAsian-Pac-Islander                     *
## raceBlack
## raceOther
## raceWhite                                  .
## sexMale                                    ***
## capital_gain                               ***
## capital_loss                               ***
```

```
## hours_per_week                           ***
## native_countryCanada
## native_countryChina                      **
## native_countryColumbia                   **
## native_countryCuba
## native_countryDominican-Republic         *
## native_countryEcuador                    *
## native_countryEl-Salvador                **
## native_countryEngland                    .
## native_countryFrance
## native_countryGermany
## native_countryGreece                     **
## native_countryGuatemala
## native_countryHaiti
## native_countryHoland-Netherlands
## native_countryHonduras
## native_countryHong                       *
## native_countryHungary
## native_countryIndia                      *
## native_countryIran                       .
## native_countryIreland
## native_countryItaly
## native_countryJamaica
## native_countryJapan
## native_countryLaos
## native_countryMexico                     *
## native_countryNicaragua                  .
## native_countryOutlying-US(Guam-USVI-etc)
## native_countryPeru                       *
## native_countryPhilippines
## native_countryPoland                     .
## native_countryPortugal                   *
## native_countryPuerto-Rico                *
## native_countryScotland
## native_countrySouth                      ***
## native_countryTaiwan                     *
## native_countryThailand
## native_countryTrinadad&Tobago
## native_countryUnited-States              .
## native_countryVietnam                    **
## native_countryYugoslavia
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 26924  on 24419  degrees of freedom
## Residual deviance: 15435  on 24323  degrees of freedom
## AIC: 15629
##
## Number of Fisher Scoring iterations: 15
```

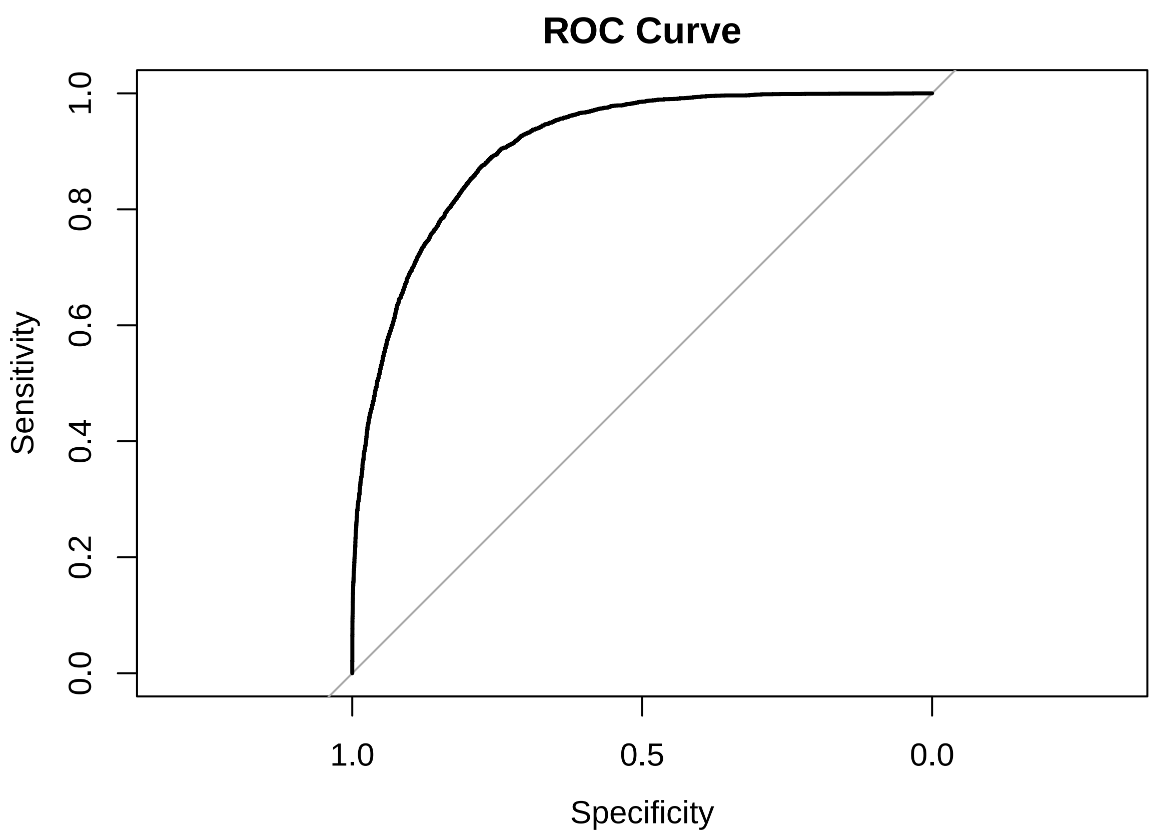
```r
# Predict probabilities
probabilities <- predict(logit_model, type = "response")
```

```r
# ROC Curve
roc_curve <- roc(completed_data$income, probabilities)
```

```
## Setting levels: control = <=50K, case = >50K
```

```
## Setting direction: controls < cases
```

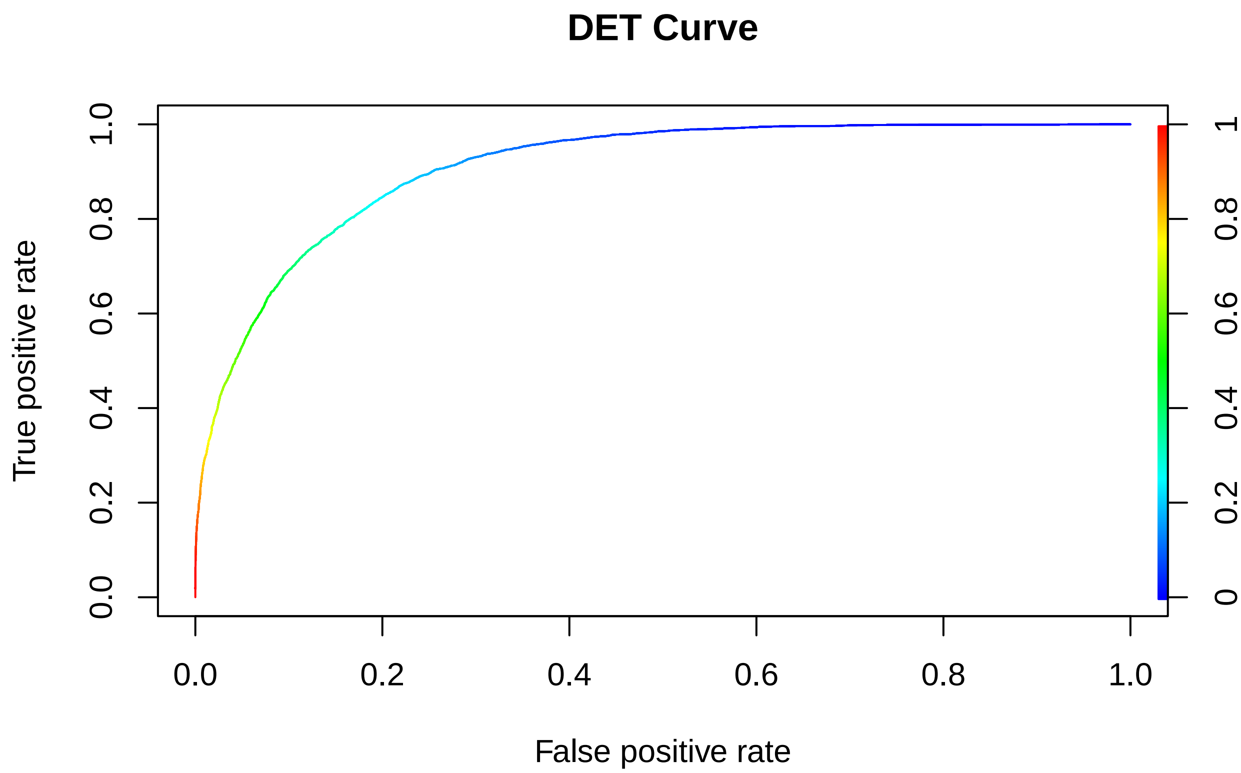
```r
plot(roc_curve, main="ROC Curve")
```




```r
# AUC Score
auc(roc_curve)
```

```
## Area under the curve: 0.9084
```

```r
# DET Curve using ROCR
pred <- prediction(probabilities, completed_data$income)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE, main="DET Curve")
```



Use the function from class to calculate all the error metrics (misclassification error, precision, recall, F1, FDR, FOR) for the values of the probability threshold being 0.001, 0.002, …, 0.999 in a tibble (dplyr data frame).

```
pacman::p_load(tidyverse)
asymmetric_predictions_results = tibble(
  p_hat_threshold = seq(from = 0.001, to = 0.999, by = 0.001),
  misclassification_error = NA,
  precision = NA,
  recall = NA,
  F1 = NA,
  FDR = NA,
  FOR = NA
)
# Predict probabilities
probabilities <- predict(logit_model, completed_data, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```
# Ensure the actual target variable is a factor with levels 0 and 1
actual <- factor(completed_data$income, levels = c("0", "1"))

# Function to calculate metrics
calculate_metrics <- function(threshold, actual, predicted) {
  # Convert predicted probabilities to binary predictions based on the threshold
```

```r
  prediction <- ifelse(predicted >= threshold, "1", "0")
  prediction <- factor(prediction, levels = c("0", "1"))

  # Only compute metrics if there are both positive and negative predictions
  if (all(levels(prediction) %in% levels(actual))) {
    cm <- confusionMatrix(prediction, actual, positive = "1")
    return(data.frame(
      misclassification_error = 1 - cm$overall['Accuracy'],
      precision = cm$byClass['Precision'],
      recall = cm$byClass['Sensitivity'],
      F1 = 2 * (cm$byClass['Precision'] * cm$byClass['Sensitivity']) / (cm$byClass['Precision'] + cm$by
      FDR = 1 - cm$byClass['Precision'],
      FOR = 1 - cm$byClass['Negative Predictive Value']
    ))
  } else {
    return(data.frame(
      misclassification_error = NA,
      precision = NA,
      recall = NA,
      F1 = NA,
      FDR = NA,
      FOR = NA
    ))
  }
}

# Create the tibble and calculate metrics for each threshold
asymmetric_predictions_results <- tibble(
  p_hat_threshold = seq(from = 0.001, to = 0.999, by = 0.001)
) %>%
  mutate(metrics = map(p_hat_threshold, calculate_metrics, actual = actual, predicted = probabilities))
  unnest(metrics)

# View the results
print(asymmetric_predictions_results)
```

```
## # A tibble: 999 x 7
##    p_hat_threshold misclassification_error precision recall    F1   FDR   FOR
##              <dbl>                   <dbl>     <dbl>  <dbl> <dbl> <dbl> <dbl>
##  1           0.001                     NaN        NA     NA    NA    NA    NA
##  2           0.002                     NaN        NA     NA    NA    NA    NA
##  3           0.003                     NaN        NA     NA    NA    NA    NA
##  4           0.004                     NaN        NA     NA    NA    NA    NA
##  5           0.005                     NaN        NA     NA    NA    NA    NA
##  6           0.006                     NaN        NA     NA    NA    NA    NA
##  7           0.007                     NaN        NA     NA    NA    NA    NA
##  8           0.008                     NaN        NA     NA    NA    NA    NA
##  9           0.009                     NaN        NA     NA    NA    NA    NA
## 10           0.01                      NaN        NA     NA    NA    NA    NA
## # i 989 more rows
```

Calculate the column `total_cost` and append it to this data frame via `mutate`.

```r
#TO-DO
cost_per_misclassification = 1
cost_per_fdr = 2
cost_per_for = 1.5

asymmetric_predictions_results <- asymmetric_predictions_results %>%
  mutate(total_cost = misclassification_error * cost_per_misclassification +
                      FDR * cost_per_fdr +
                      FOR * cost_per_for)

# View the updated results
print(asymmetric_predictions_results)
```

```
## # A tibble: 999 x 8
##    p_hat_threshold misclassification_error precision recall    F1   FDR   FOR
##              <dbl>                   <dbl>     <dbl>  <dbl> <dbl> <dbl> <dbl>
##  1           0.001                     NaN        NA     NA    NA    NA    NA
##  2           0.002                     NaN        NA     NA    NA    NA    NA
##  3           0.003                     NaN        NA     NA    NA    NA    NA
##  4           0.004                     NaN        NA     NA    NA    NA    NA
##  5           0.005                     NaN        NA     NA    NA    NA    NA
##  6           0.006                     NaN        NA     NA    NA    NA    NA
##  7           0.007                     NaN        NA     NA    NA    NA    NA
##  8           0.008                     NaN        NA     NA    NA    NA    NA
##  9           0.009                     NaN        NA     NA    NA    NA    NA
## 10           0.01                      NaN        NA     NA    NA    NA    NA
## # i 989 more rows
## # i 1 more variable: total_cost <dbl>
```

Which is the lowest total cost? What is the "winning" probability threshold value providing that minimum total cost?

```r
#TO-DO
# Find the row with the minimum total cost
min_cost_data <- asymmetric_predictions_results %>%
  filter(total_cost == min(total_cost, na.rm = TRUE)) %>%
  slice(1)  # In case there are multiple minima, take the first
```

```
## Warning: There was 1 warning in `filter()`.
## i In argument: `total_cost == min(total_cost, na.rm = TRUE)`.
## Caused by warning in `min()`:
## ! no non-missing arguments to min; returning Inf
```

```r
# Display the result
print(min_cost_data)
```

```
## # A tibble: 0 x 8
## # i 8 variables: p_hat_threshold <dbl>, misclassification_error <dbl>,
## #   precision <dbl>, recall <dbl>, F1 <dbl>, FDR <dbl>, FOR <dbl>,
## #   total_cost <dbl>
```

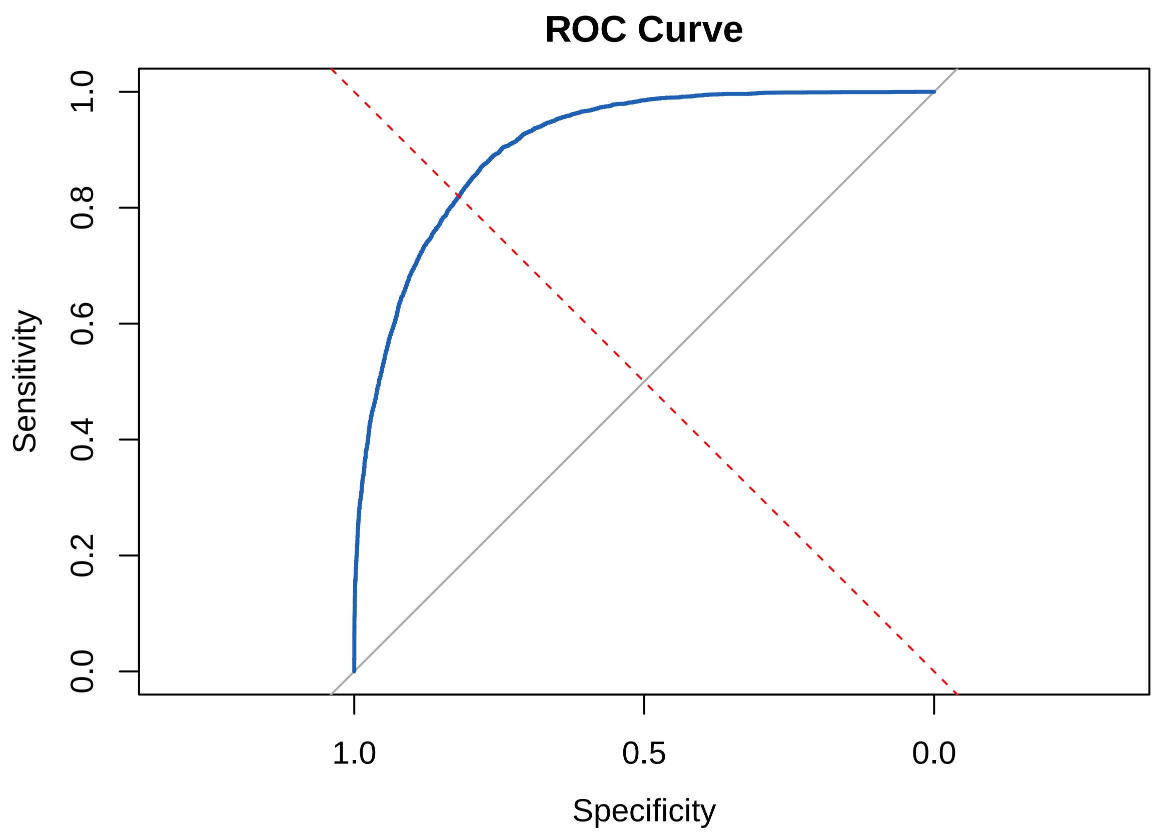
Plot an ROC curve and interpret.

```r
#TO-DO
roc_curve <- roc(response = completed_data$income, predictor = probabilities)
```

```
## Setting levels: control = <=50K, case = >50K
```

```
## Setting direction: controls < cases
```

```r
plot(roc_curve, main = "ROC Curve", col = "#1c61b6", lwd = 2)
abline(a = 0, b = 1, lty = 2, col = "red")  # Adding a diagonal reference line
```



#TO-DO interpretation The ROC curve indicates a good predictive model, as it significantly bows towards the top left corner, suggesting a high true positive rate with a low false positive rate across various thresholds.

Calculate AUC and interpret.

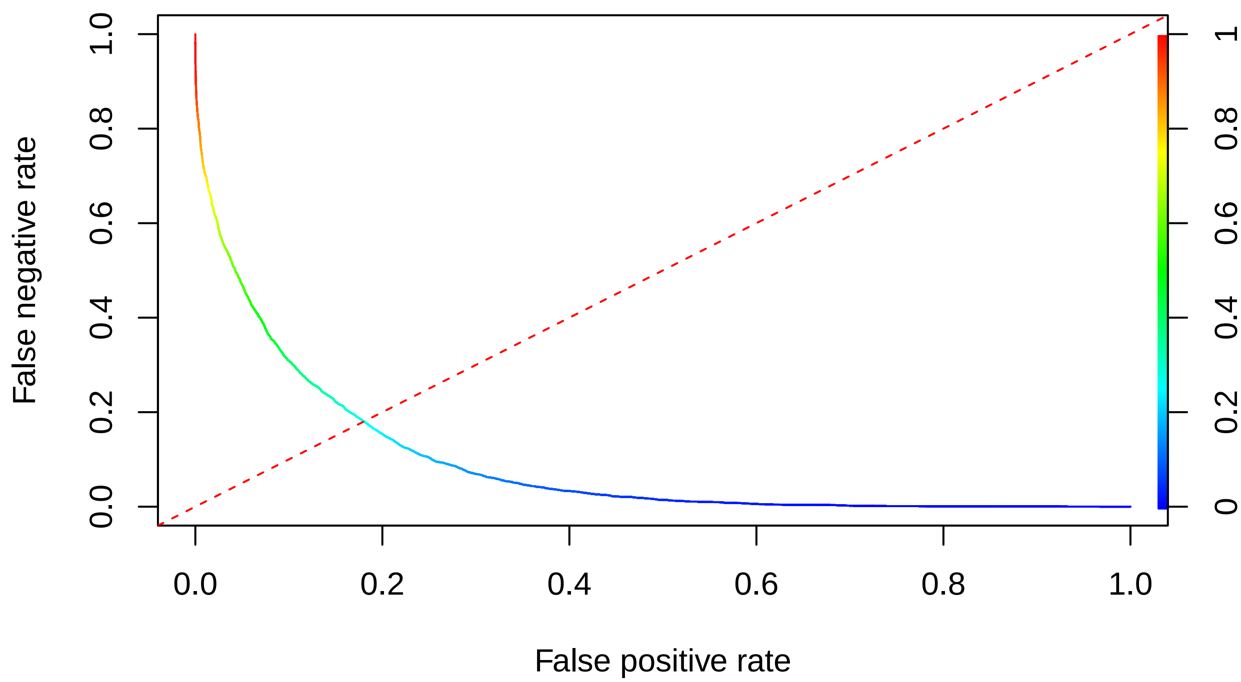
```r
#TO-DO
auc_value <- auc(roc_curve)
print(paste("AUC value:", auc_value))
```

```
## [1] "AUC value: 0.908380715565783"
```

#TO-DO interpretation The AUC value of 0.9095 indicates excellent performance. This suggests that the classifier does very well at distinguishing between the positive class and the negative class.

Plot a DET curve and interpret.

```r
#TO-DOpred <- prediction(probabilities, actual)
perf <- performance(pred, "fnr", "fpr")

# Plot the DET curve
plot(perf, colorize = TRUE)
abline(0, 1, lty = 2, col = "red")  # Adding a reference line
```




```r
# Transforming the axis to normal deviate scale might require additional steps.
```

#TO-DO interpretation The DET curve illustrates a model with strong performance, evidenced by the steep curve towards the lower left, which indicates a low false match rate for most thresholds before the false non-match rate begins to increase significantly.