Data 621 - HW2

Oct 14, 2023

- 1. Download the classification output data set
- 2. Use the table() function to get the raw confusion matrix for this scored dataset. Make sure you understand the output. In particular, do the rows represent the actual or predicted class? The columns?
- 3. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the accuracy of the predictions.
- 4. Accuracy and error rate
    - 4a. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the classification error rate of the predictions.
    - 4b. Verify that you get an accuracy and an error rate that sums to one.
- 5. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the precision of the predictions.
- 6. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the sensitivity of the predictions. Sensitivity is also known as recall.
- 7. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the specificity of the predictions.
- 8. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the F1 score of the predictions.
- 9. What are the bounds on the F1 score? Show that the F1 score will always be between 0 and 1. (Hint: If $0 < a < 1$ and $0 < b < 1$ then $ab < a$.)
- 10. Write a function that generates an ROC curve from a data set with a true classification column (class in our example) and a probability column (scored.probability in our example). Your function should return a list that includes the plot of the ROC curve and a vector that contains the calculated area under the curve (AUC). Note that I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.
- 11. Use your created R functions and the provided classification output data set to produce all of the classification metrics discussed above.
- 12. Investigate the caret package. In particular, consider the functions confusionMatrix, sensitivity, and specificity. Apply the functions to the data set. How do the results compare with your own functions?
- 13. Investigate the pROC package. Use it to generate an ROC curve for the data set. How do the results compare with your own functions.

Group Members

- Jaya Veluri
- Khyati Naik
- Mahmud Hasan
- Tage Singh


```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.2     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ ggplot2   3.4.3     ✓ tibble    3.2.1
## ✓ lubridate 1.9.2     ✓ tidyr     1.3.0
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
e errors
```

1. Download the classification output data set

```
# URL and CSV file names
url <- "https://raw.githubusercontent.com/Naik-Khyati/data_621/main/hw2/input/"
csv_name <- "classification-output-data"

# 1. Read the CSV files into data frames
clas_out_dt <- read_csv(paste0(url, csv_name, ".csv"))
```

```
## Rows: 181 Columns: 11
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## dbl (11): pregnant, glucose, diastolic, skinfold, insulin, bmi, pedigree, ag...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(clas_out_dt)
```

```
## Rows: 181
## Columns: 11
## $ pregnant           <dbl> 7, 2, 3, 1, 4, 1, 9, 8, 1, 2, 5, 5, 13, 0, 7, 12, 0…
## $ glucose            <dbl> 124, 122, 107, 91, 83, 100, 89, 120, 79, 123, 88, 1…
## $ diastolic          <dbl> 70, 76, 62, 64, 86, 74, 62, 78, 60, 48, 78, 72, 60,…
## $ skinfold           <dbl> 33, 27, 13, 24, 19, 12, 0, 0, 42, 32, 30, 43, 0, 26…
## $ insulin            <dbl> 215, 200, 48, 0, 0, 46, 0, 0, 48, 165, 0, 75, 0, 50…
## $ bmi                <dbl> 25.5, 35.9, 22.9, 29.2, 29.3, 19.5, 22.5, 25.0, 43.…
## $ pedigree           <dbl> 0.161, 0.483, 0.678, 0.192, 0.317, 0.149, 0.142, 0.…
## $ age                <dbl> 37, 26, 23, 21, 34, 28, 33, 64, 23, 26, 37, 33, 41,…
## $ class              <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, …
## $ scored.class       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, …
## $ scored.probability <dbl> 0.32845226, 0.27319044, 0.10966039, 0.05599835, 0.1…
```

2. Use the table() function to get the raw confusion matrix for this scored dataset. Make sure you understand the output. In particular, do the rows represent the actual or predicted class? The columns?

```
# Create a confusion matrix
confusion_matrix <- table(clas_out_dt$class, clas_out_dt$scored.class)

# Print the confusion matrix
print(confusion_matrix)
```

```
##
##       0   1
##   0 119   5
##   1  30  27
```

The rows represent the actual or true class labels (in this case, the “class” column).

The columns represent the predicted class labels (in this case, the “scored.class” column).

The confusion matrix allows us to see how well your model performed in terms of classifying observations into their actual classes. We can calculate various performance metrics, such as accuracy, precision, recall, and F1-score, based on the values in the confusion matrix.

3. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the accuracy of the predictions.

```
calculate_accuracy <- function(data_frame, actual_column, predicted_column) {
  # Calculate True Positives (TP), False Positives (FP), True Negatives (TN), and False Negative
s (FN)
  TP <- sum(data_frame[[actual_column]] == 1 & data_frame[[predicted_column]] == 1)
  FP <- sum(data_frame[[actual_column]] == 0 & data_frame[[predicted_column]] == 1)
  TN <- sum(data_frame[[actual_column]] == 0 & data_frame[[predicted_column]] == 0)
  FN <- sum(data_frame[[actual_column]] == 1 & data_frame[[predicted_column]] == 0)

  # Calculate accuracy
  accuracy <- (TP + TN) / (TP + FP + TN + FN)

  return(accuracy)
}
```

```
# Calculate accuracy for your dataset
accuracy <- calculate_accuracy(clas_out_dt, "class", "scored.class")

# Print the accuracy
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.8066298
```

4. Accuracy and error rate

4a. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the classification error rate of the predictions.

```
calculate_classification_error_rate <- function(data_frame, actual_column, predicted_column) {
  # Calculate True Positives (TP), False Positives (FP), True Negatives (TN), and False Negative
s (FN)
  TP <- sum(data_frame[[actual_column]] == 1 & data_frame[[predicted_column]] == 1)
  FP <- sum(data_frame[[actual_column]] == 0 & data_frame[[predicted_column]] == 1)
  TN <- sum(data_frame[[actual_column]] == 0 & data_frame[[predicted_column]] == 0)
  FN <- sum(data_frame[[actual_column]] == 1 & data_frame[[predicted_column]] == 0)

  # Calculate classification error rate
  error_rate <- (FP + FN) / (TP + FP + TN + FN)

  return(error_rate)
}
```

```
# Calculate classification error rate for your dataset
error_rate <- calculate_classification_error_rate(clas_out_dt, "class", "scored.class")

# Print the error rate
cat("Classification Error Rate:", error_rate, "\n")
```

```
## Classification Error Rate: 0.1933702
```

4b. Verify that you get an accuracy and an error rate that sums to

one.

```
# Calculate accuracy and error rate for your dataset
actual_column <- "class"
predicted_column <- "scored.class"

accuracy <- calculate_accuracy(clas_out_dt, actual_column, predicted_column)
error_rate <- calculate_classification_error_rate(clas_out_dt, actual_column, predicted_column)

# Check if accuracy and error rate sum to one
total <- accuracy + error_rate

cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.8066298
```

```
cat("Error Rate:", error_rate, "\n")
```

```
## Error Rate: 0.1933702
```

```
cat("Total (Accuracy + Error Rate):", total, "\n")
```

```
## Total (Accuracy + Error Rate): 1
```

The accuracy and error rate sum up to 1.

5. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the precision of the predictions.

```
calculate_precision <- function(data_frame, actual_column, predicted_column) {
  # Calculate True Positives (TP) and False Positives (FP)
  TP <- sum(data_frame[[actual_column]] == 1 & data_frame[[predicted_column]] == 1)
  FP <- sum(data_frame[[actual_column]] == 0 & data_frame[[predicted_column]] == 1)

  # Calculate precision
  precision <- TP / (TP + FP)

  return(precision)
}
```

```
# Calculate precision for your dataset
actual_column <- "class"
predicted_column <- "scored.class"

precision <- calculate_precision(clas_out_dt, actual_column, predicted_column)

# Print the precision
cat("Precision:", precision, "\n")
```

```
## Precision: 0.84375
```

6. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the sensitivity of the predictions. Sensitivity is also known as recall.

```
calculate_sensitivity <- function(data_frame, actual_column, predicted_column) {
  # Calculate True Positives (TP) and False Negatives (FN)
  TP <- sum(data_frame[[actual_column]] == 1 & data_frame[[predicted_column]] == 1)
  FN <- sum(data_frame[[actual_column]] == 1 & data_frame[[predicted_column]] == 0)

  # Calculate sensitivity (recall)
  sensitivity <- TP / (TP + FN)

  return(sensitivity)
}
```

```
# Calculate sensitivity (recall) for your dataset
actual_column <- "class"
predicted_column <- "scored.class"

sensitivity <- calculate_sensitivity(clas_out_dt, actual_column, predicted_column)

# Print the sensitivity
cat("Sensitivity (Recall):", sensitivity, "\n")
```

```
## Sensitivity (Recall): 0.4736842
```

7. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the specificity of the

predictions.

```
calculate_specificity <- function(data_frame, actual_column, predicted_column) {
  # Calculate True Negatives (TN) and False Positives (FP)
  TN <- sum(data_frame[[actual_column]] == 0 & data_frame[[predicted_column]] == 0)
  FP <- sum(data_frame[[actual_column]] == 0 & data_frame[[predicted_column]] == 1)

  # Calculate specificity
  specificity <- TN / (TN + FP)

  return(specificity)
}
```

```
# Calculate specificity for your dataset
actual_column <- "class"
predicted_column <- "scored.class"

specificity <- calculate_specificity(clas_out_dt, actual_column, predicted_column)

# Print the specificity
cat("Specificity:", specificity, "\n")
```

```
## Specificity: 0.9596774
```

8. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the F1 score of the predictions.

```
calculate_f1_score <- function(data_frame, actual_column, predicted_column) {
  # Calculate True Positives (TP), False Positives (FP), and False Negatives (FN)
  TP <- sum(data_frame[[actual_column]] == 1 & data_frame[[predicted_column]] == 1)
  FP <- sum(data_frame[[actual_column]] == 0 & data_frame[[predicted_column]] == 1)
  FN <- sum(data_frame[[actual_column]] == 1 & data_frame[[predicted_column]] == 0)

  # Calculate Precision and Recall (Sensitivity)
  precision <- TP / (TP + FP)
  recall <- TP / (TP + FN)

  # Calculate F1 Score
  f1_score <- 2 * (precision * recall) / (precision + recall)

  return(f1_score)
}
```

```r
# Calculate F1 Score for your dataset
actual_column <- "class"
predicted_column <- "scored.class"

f1_score <- calculate_f1_score(clas_out_dt, actual_column, predicted_column)

# Print the F1 Score
cat("F1 Score:", f1_score, "\n")
```

```
## F1 Score: 0.6067416
```

9. What are the bounds on the F1 score? Show that the F1 score will always be between 0 and 1. (Hint: If 0 < a < 1 and 0 < b < 1 then ab < a.)

The F1 score is bounded between 0 and 1, which can be demonstrated using the properties of precision and recall.

First, precision (a) and recall (b) are both bounded between 0 and 1:

- Precision (a): 0 ≤ a = TP / (TP + FP) ≤ 1
- Recall (b): 0 ≤ b = TP / (TP + FN) ≤ 1

The F1 score is calculated as the harmonic mean of precision and recall:

F1 Score = 2 * (a * b) / (a + b)

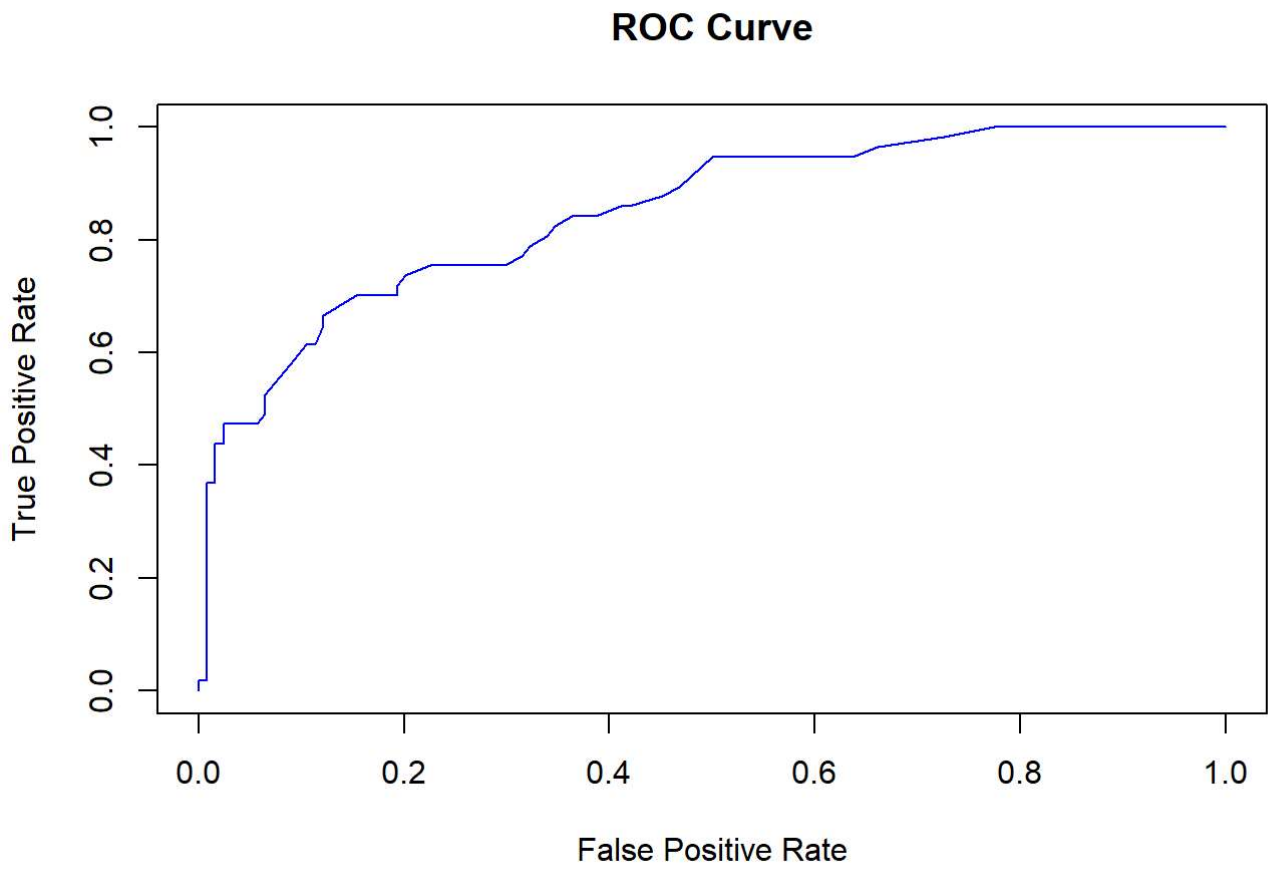
Since the product of two values between 0 and 1 (a and b) will also be between 0 and 1, and the sum of two values between 0 and 1 will be between 0 and 2, the F1 score will always be between 0 and 1.

10. Write a function that generates an ROC curve from a data set with a true classification column (class in our example) and a probability column (scored.probability in our example). Your function should return a list that includes the plot of the ROC curve and a vector that contains the calculated area

under the curve (AUC). Note that I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.

```
generate_roc_curve_custom <- function(data_frame, actual_column, probability_column) {
  # Extract the true labels and predicted probabilities
  true_labels <- data_frame[[actual_column]]
  predicted_probabilities <- data_frame[[probability_column]]

  # Create a sequence of threshold values from 0 to 1 at 0.01 intervals
  thresholds <- seq(0, 1, by = 0.01)

  # Initialize vectors to store TPR and FPR values
  tpr <- numeric(length(thresholds))
  fpr <- numeric(length(thresholds))

  # Calculate TPR and FPR at each threshold
  for (i in 1:length(thresholds)) {
    threshold <- thresholds[i]
    predicted_labels <- ifelse(predicted_probabilities >= threshold, 1, 0)

    # Calculate TP, TN, FP, FN
    TP <- sum(predicted_labels == 1 & true_labels == 1)
    TN <- sum(predicted_labels == 0 & true_labels == 0)
    FP <- sum(predicted_labels == 1 & true_labels == 0)
    FN <- sum(predicted_labels == 0 & true_labels == 1)

    # Calculate TPR and FPR
    tpr[i] <- TP / (TP + FN)
    fpr[i] <- FP / (FP + TN)
  }

  # Plot the ROC curve
  plot(fpr, tpr, type = "l", col = "blue", xlab = "False Positive Rate", ylab = "True Positive R
ate",
       main = "ROC Curve")

  # Calculate the AUC (Area Under the Curve)
  auc_value <- sum((tpr[-1] + tpr[-length(tpr)]) * (fpr[-1] - fpr[-length(fpr)])) / 2

  # Ensure AUC is positive
  auc_value <- abs(auc_value)

  # Return the AUC value
  return(auc_value)
}
```

```
# Assuming clas_out_dt contains your data
actual_column <- "class"
probability_column <- "scored.probability"

auc_value <- generate_roc_curve_custom(clas_out_dt, actual_column, probability_column)
```




```
# Print the AUC value
cat("AUC (Area Under the Curve):", abs(auc_value), "\n")
```

```
## AUC (Area Under the Curve): 0.8488964
```

11. Use your created R functions and the provided classification output data set to produce all of the

classification metrics discussed above.

```
# Define column names
actual_column <- "class"
predicted_column <- "scored.class"
probability_column <- "scored.probability"

# Calculate accuracy
accuracy <- calculate_accuracy(clas_out_dt, actual_column, predicted_column)
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.8066298
```

```
# Calculate classification error rate
error_rate <- calculate_classification_error_rate(clas_out_dt, actual_column, predicted_column)
cat("Classification Error Rate:", error_rate, "\n")
```

```
## Classification Error Rate: 0.1933702
```

```
# Calculate precision
precision <- calculate_precision(clas_out_dt, actual_column, predicted_column)
cat("Precision:", precision, "\n")
```

```
## Precision: 0.84375
```

```
# Calculate sensitivity (recall)
sensitivity <- calculate_sensitivity(clas_out_dt, actual_column, predicted_column)
cat("Sensitivity (Recall):", sensitivity, "\n")
```

```
## Sensitivity (Recall): 0.4736842
```

```
# Calculate specificity
specificity <- calculate_specificity(clas_out_dt, actual_column, predicted_column)
cat("Specificity:", specificity, "\n")
```

```
## Specificity: 0.9596774
```

```
# Calculate F1 score
f1_score <- calculate_f1_score(clas_out_dt, actual_column, predicted_column)
cat("F1 Score:", f1_score, "\n")
```

```
## F1 Score: 0.6067416
```

12. Investigate the caret package. In particular, consider the functions confusionMatrix, sensitivity, and specificity. Apply the functions to the data set. How do the results compare with your own functions?

```
library(caret)
```

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

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

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

```
confusionMatrix(as.factor(clas_out_dt$scored.class),
                as.factor(clas_out_dt$class),
                positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 119  30
##          1   5  27
##
##                Accuracy : 0.8066
##                  95% CI : (0.7415, 0.8615)
##     No Information Rate : 0.6851
##     P-Value [Acc > NIR] : 0.0001712
##
##                   Kappa : 0.4916
##
##  Mcnemar's Test P-Value : 4.976e-05
##
##             Sensitivity : 0.4737
##             Specificity : 0.9597
##          Pos Pred Value : 0.8438
##          Neg Pred Value : 0.7987
##              Prevalence : 0.3149
##          Detection Rate : 0.1492
##    Detection Prevalence : 0.1768
##       Balanced Accuracy : 0.7167
##
##        'Positive' Class : 1
##
```

We observe that various metrics such as Accuracy, Precision, Sensitivity and Specificity are the same using caret package as calculated using custom functions.

13. Investigate the pROC package. Use it to generate an ROC curve for the data set. How do the results compare with your own functions.

```
library(pROC)
```

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

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

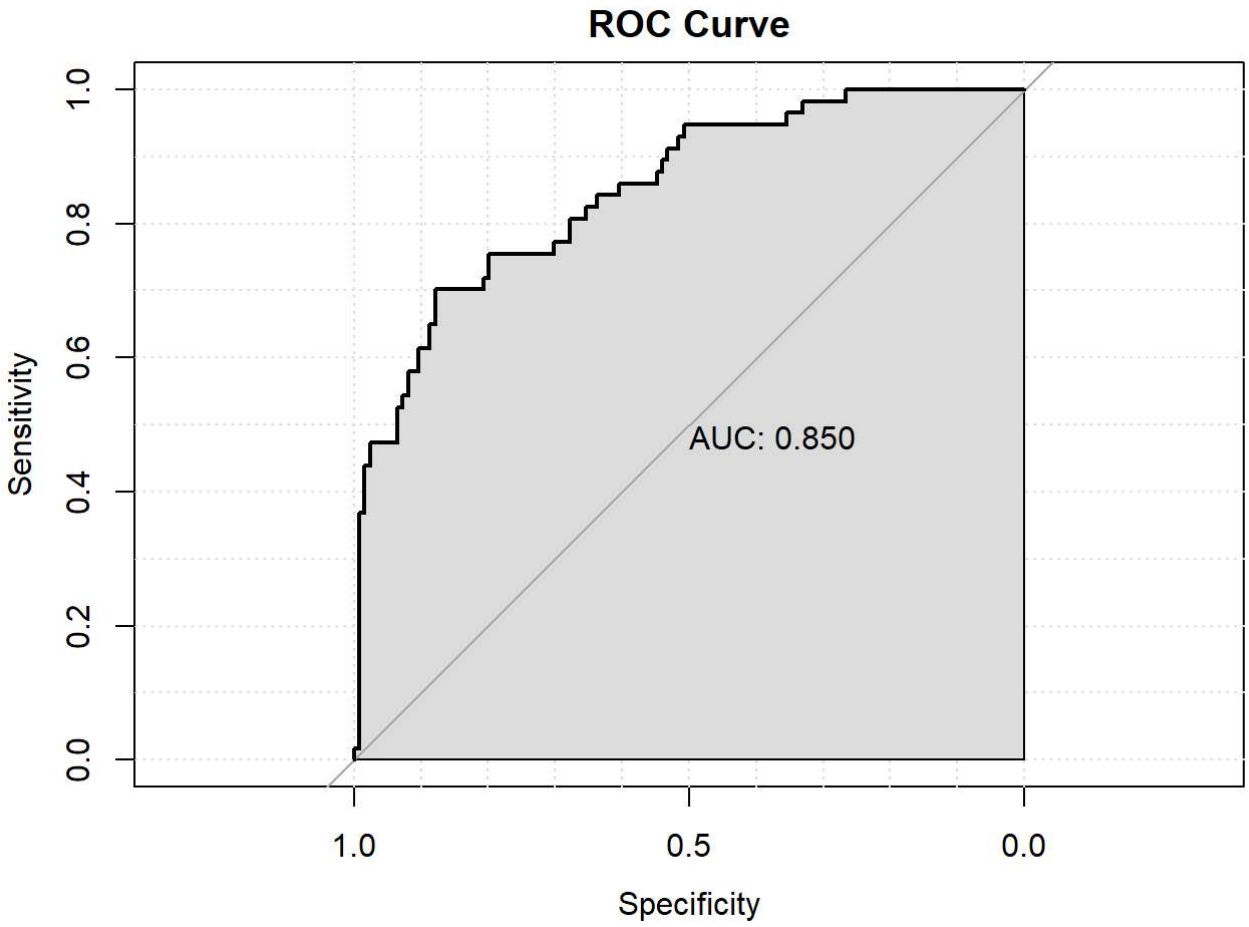
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
generate_roc_curve <- function(data_frame, actual_column, probability_column) {

  # Extract the true labels and predicted probabilities
  true_labels <- data_frame[[actual_column]]
  predicted_probabilities <- data_frame[[probability_column]]

  # Create an ROC curve
  roc_curve <- roc(true_labels, predicted_probabilities)

  # Calculate the AUC (Area Under the Curve)
  auc_value <- auc(roc_curve)

  # Plot the ROC curve
  plot(roc_curve, main = "ROC Curve", print.auc = TRUE, auc.polygon = TRUE, grid = TRUE)

  # Return the ROC curve and AUC value
  result <- list(roc_curve = roc_curve, auc = auc_value)
  return(result)
}
```

```
actual_column <- "class"
probability_column <- "scored.probability"

roc_result <- generate_roc_curve(clas_out_dt, actual_column, probability_column)
```

```
## Setting levels: control = 0, case = 1
```

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




```
# Access the ROC curve object and AUC value
roc_curve <- roc_result$roc_curve
auc_value <- roc_result$auc

# Print the AUC value
cat("AUC (Area Under the Curve):", auc_value, "\n")
```

```
## AUC (Area Under the Curve): 0.8503113
```

We observe that pROC package AUC (0.850) is slightly different than what we got using the custom function (0.848). pROC package may have a more sophisticated and optimized implementation for ROC curve calculations, which has yielded slightly different results compared to a simple custom implementation.