DATA621Assignment2

Erick Hadi, Marco Castro, Mubashira Qari, Puja Roy, Zach Rose

2025-03-15

Load the Data

Download the classification output data set (attached in Blackboard to the assignment)

```
classification_df <- read.csv("https://raw.githubusercontent.com/uzmabb182/Data_621/refs/heads/main/Ass
head(classification_df)
```

```
##   pregnant glucose diastolic skinfold insulin  bmi pedigree age class
## 1        7     124        70       33     215 25.5    0.161  37     0
## 2        2     122        76       27     200 35.9    0.483  26     0
## 3        3     107        62       13      48 22.9    0.678  23     1
## 4        1      91        64       24       0 29.2    0.192  21     0
## 5        4      83        86       19       0 29.3    0.317  34     0
## 6        1     100        74       12      46 19.5    0.149  28     0
##   scored.class scored.probability
## 1            0         0.32845226
## 2            0         0.27319044
## 3            0         0.10966039
## 4            0         0.05599835
## 5            0         0.10049072
## 6            0         0.05515460
```

Confusion Matrix

The data set has three key columns we will use:

- class: the actual class for the observation
- scored.class: the predicted class for the observation (based on a threshold of 0.5)
- scored.probability: the predicted probability of success for the observation

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?

Confusion Matrix: The table() function in R will give you the raw confusion matrix. In this table:

- The rows generally represent the actual class
- The columns generally represent the predicted class

For example: Predicted Positive Predicted Negative Actual Positive True Positive (TP) False Negative (FN) Actual Negative False Positive (FP) True Negative (TN)

```
confusion_matrix <- table(classification_df$class, classification_df$scored.class)
confusion_matrix
```

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

This can be interpreted as:

- True Negatives (TN): 119 – These are the instances where the actual class was 0, and the predicted class was also 0
- False Positives (FP): 5 – These are the instances where the actual class was 0, but the predicted class was 1
- False Negatives (FN): 30 – These are the instances where the actual class was 1, but the predicted class was 0
- True Positives (TP): 27 – These are the instances where the actual class was 1, and the predicted class was also 1

Accuracy

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

Accuracy: Represents the proportion of correct predictions. For instance, if accuracy is 90%, it means the model correctly classified 90% of the observations as either success or failure.

Formula Accuracy = (TP + TN) / (TP + FP + TN + FN)

```r
calculate_accuracy <- function(data) {
  confusion_matrix <- table(data$class, data$scored.class)
  TP <- confusion_matrix[2, 2]
  TN <- confusion_matrix[1, 1]
  FP <- confusion_matrix[1, 2]
  FN <- confusion_matrix[2, 1]
  accuracy <- (TP + TN) / (TP + FP + TN + FN)
  return(accuracy)
}
```

Classification Error

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.

Error Rate: Complements accuracy and indicates the proportion of incorrect predictions. If it's 10%, it tells us the model misclassified 10% of observations.

Formula Classification Error Rate == (FP + FN) / (TP + FP + TN + FN)

```r
calculate_error_rate <- function(data) {
  confusion_matrix <- table(data$class, data$scored.class)
  FP <- confusion_matrix[1, 2]
  FN <- confusion_matrix[2, 1]
  TP <- confusion_matrix[2, 2]
  TN <- confusion_matrix[1, 1]
  error_rate <- (FP + FN) / (TP + FP + TN + FN)
  return(error_rate)
}
```

Precision

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

Precision: Focuses on the quality of positive predictions. For example, in a healthcare setting, a high precision rate means most patients predicted to have a condition actually do have it, reducing the risk of unnecessary interventions.

Formula Precision = TP / (TP + FP)

```r
calculate_precision <- function(data) {
  confusion_matrix <- table(data$class, data$scored.class)
  TP <- confusion_matrix[2, 2]
  FP <- confusion_matrix[1, 2]
  precision <- TP / (TP + FP)
  return(precision)
}
```

Sensitivity (Recall / True Positive Rate)

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.

Sensitivity measures the ability of the model to correctly identify actual positives. This tells us how well the model is capturing positive cases. For example, if this is a medical dataset, sensitivity indicates the percentage of patients correctly identified as having the condition.

Formula Sensitivity = TP / (TP + FN)

```r
calculate_sensitivity <- function(data) {
  confusion_matrix <- table(data$class, data$scored.class)
  TP <- confusion_matrix[2, 2]
  FN <- confusion_matrix[2, 1]
  sensitivity <- TP / (TP + FN)
  return(sensitivity)
}
```

Specificity (True Negative Rate)

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

Specificity measures the ability of the model to correctly identify actual negatives. In the context of the data, specificity shows how well the model avoids falsely predicting positives for negative cases.

Formula Specificity = TN / (TN + FP)

```r
calculate_specificity <- function(data) {
  confusion_matrix <- table(data$class, data$scored.class)
  TN <- confusion_matrix[1, 1]
  FP <- confusion_matrix[1, 2]
  specificity <- TN / (TN + FP)
  return(specificity)
}
```

F1 Score

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

The F1 score is the harmonic mean of Precision and Sensitivity. It balances the trade-off between precision (quality of positive predictions) and sensitivity (coverage of actual positives). This metric is useful when there is an imbalance between positive and negative classes in the dataset.

Formula F1 Score = 2 * (precision * sensitivity) / (precision + sensitivity)

```r
calculate_f1_score <- function(data) {
  confusion_matrix <- table(data$class, data$scored.class)
  TP <- confusion_matrix[2, 2]
  FP <- confusion_matrix[1, 2]
  FN <- confusion_matrix[2, 1]

  precision <- TP / (TP + FP)
  sensitivity <- TP / (TP + FN)

  f1_score <- 2 * (precision * sensitivity) / (precision + sensitivity)
  return(f1_score)
}
```

What are the bounds on the F1 score?

Before we move on, let's consider a question that was asked: 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 $ab < a$.)

F1 Score is always between 0 and 1

1. Both Precision and Sensitivity are proportions, and thus range between 0 and 1
2. When multiplying two values between 0 and 1, the result is also between 0 and 1. For example, If $0 < a < 1$ and $0 < b < 1$ then $a\ b < a$ and $a\ b < b$
3. Dividing a number between 0 and 1 (numerator) by another number greater than it (denominator) ensures the result remains between 0 and 1.

Thus, since Precision and Sensitivity are bounded by 0 and 1, their harmonic mean (F1 Score) must also be within this range.

Function that generates an ROC curve

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.

```r
generate_roc <- function(data) {
  true_class <- data$class
  predicted_probabilities <- data$scored.probability

  thresholds <- seq(0, 1, by = 0.01)

  TPR <- numeric(length(thresholds))
  FPR <- numeric(length(thresholds))

  for (i in seq_along(thresholds)) {
    threshold <- thresholds[i]

    predicted_class <- ifelse(predicted_probabilities >= threshold, 1, 0)

    TP <- sum(predicted_class == 1 & true_class == 1)
    TN <- sum(predicted_class == 0 & true_class == 0)
    FP <- sum(predicted_class == 1 & true_class == 0)
    FN <- sum(predicted_class == 0 & true_class == 1)

    TPR[i] <- TP / (TP + FN) # Sensitivity
    FPR[i] <- FP / (FP + TN) # 1 - Specificity
  }

  auc <- sum(diff(FPR) * (TPR[-1] + TPR[-length(TPR)]) / 2)
  auc <- abs(auc)

  roc_data <- data.frame(FPR = FPR, TPR = TPR)

  roc_plot <- ggplot(roc_data, aes(x = FPR, y = TPR)) +
    geom_line(color = "blue") +
    geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "red") +
    labs(title = "ROC Curve", x = "False Positive Rate (FPR)",
         y = "True Positive Rate (TPR)") +
         theme_minimal()

  return(list(plot = roc_plot, auc = auc))
}
```

Use your created R functions and the provided classification output data set to produce all of the classification metrics discussed above

Consider the functions confusionMatrix, sensitivity, and specificity

```
print(paste("Confusion Matrix: "))
```

```
## [1] "Confusion Matrix: "
```

```
confusion_matrix
```

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

```
print(paste("Accuracy: ", calculate_accuracy(classification_df)))
```

```
## [1] "Accuracy:  0.806629834254144"
```

```
print(paste("Error rate: ", calculate_error_rate(classification_df)))
```

```
## [1] "Error rate:  0.193370165745856"
```

```
print(paste("Precision: ", calculate_precision(classification_df)))
```

```
## [1] "Precision:  0.84375"
```

```
print(paste("Sensitivity: ", calculate_sensitivity(classification_df)))
```

```
## [1] "Sensitivity:  0.473684210526316"
```

```
print(paste("Specifivity: ", calculate_specificity(classification_df)))
```

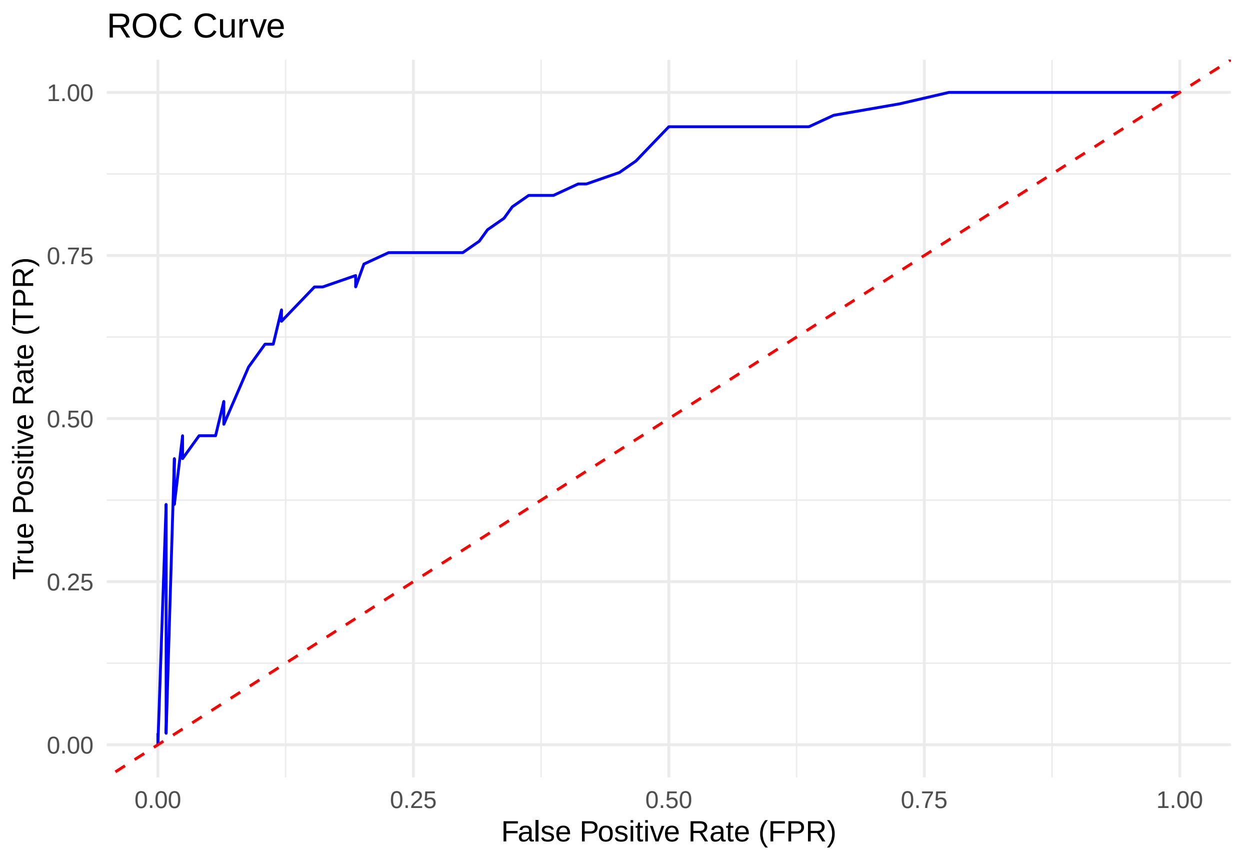
```
## [1] "Specifivity:  0.959677419354839"
```

```
print(paste("Classification: ", calculate_f1_score(classification_df)))
```

```
## [1] "Classification:  0.606741573033708"
```

```
roc_gen <- generate_roc(classification_df)

print(roc_gen$plot)
```



```r
print(paste("AUC:", roc_gen$auc))
```

```
## [1] "AUC: 0.848896434634975"
```

Investigate the caret package

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?

```r
caret_results <- confusionMatrix(as.factor(classification_df$scored.class),
                                 as.factor(classification_df$class), positive = "1")

print(caret_results)
```

```
## 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
##
```

The results from using the caret function compared to our own function are nearly identical.

Investigate the pROC package

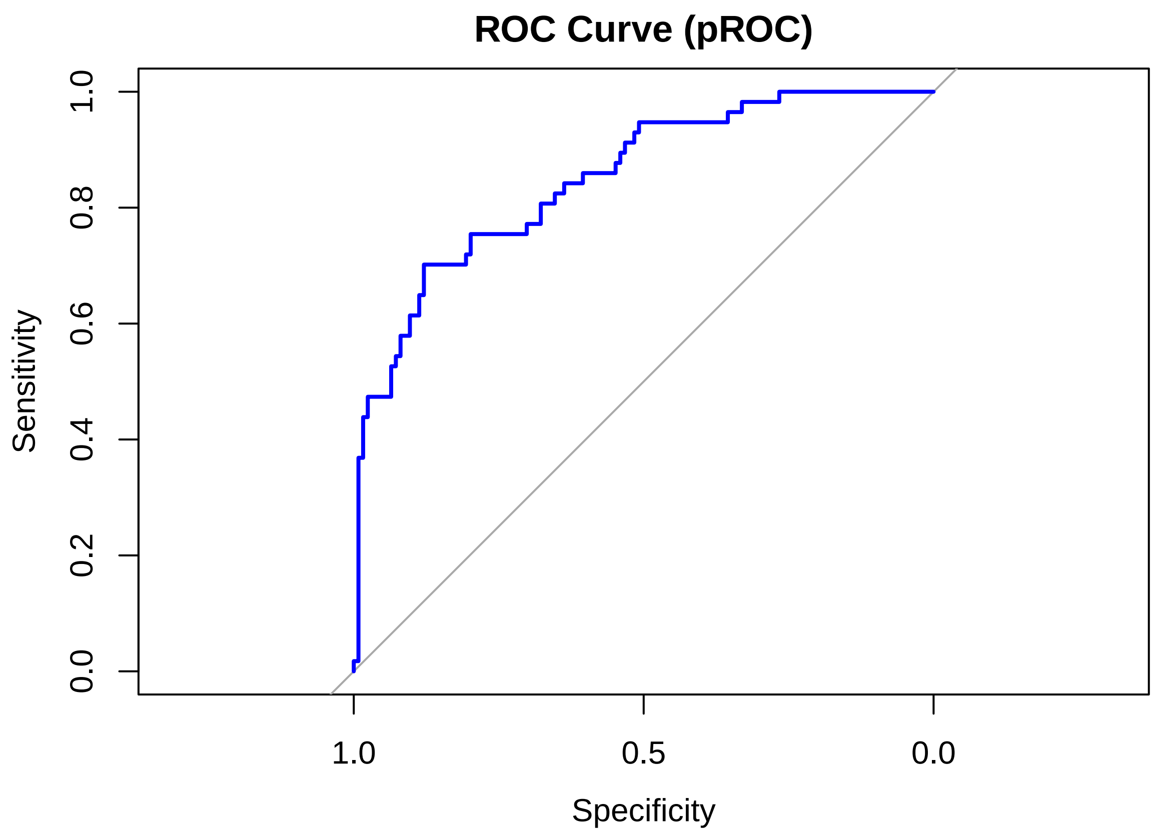
Investigate the pROC package. Use it to generate an ROC curve for the data set. How do the results compare with your own functions?

```r
roc_result <- roc(classification_df$class, classification_df$scored.probability)
```

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

## Setting direction: controls < cases
```

```r
plot(roc_result, col = "blue", main = "ROC Curve (pROC)")
```



```r
auc_value <- auc(roc_result)
print(paste("AUC (pROC):", auc_value))
```

```
## [1] "AUC (pROC): 0.850311262026033"
```

The curve produced by pROC is nearly identical to the one generated by the manual function, and the AUC values were nearly the same. While writing the function by hand was valuable for learning the underlying mechanics of ROC analysis, using pROC is clearly more efficient and practical for real-world work.