HW2

Selina Noori, Gavriel Steinmitz-Silber, John Cruz, Shaya Engelman, Daniel Craig

2024-03-04

1 - Get the Data

The data provided is the result of a classification model applied to a dataset related to diabetes. The actual class, whether the patient had diabetes, is stored in column “class” and the model’s predicted class is stored in column “scored.class”

pregnant	glucose	diastolic	skinfold	insulin	bmi	pedigree	age	class	scored.class	scored.probability
7	124	70	33	215	25.5	0.161	37	0	0	0.3284523
2	122	76	27	200	35.9	0.483	26	0	0	0.2731904
3	107	62	13	48	22.9	0.678	23	1	0	0.1096604
1	91	64	24	0	29.2	0.192	21	0	0	0.0559984
4	83	86	19	0	29.3	0.317	34	0	0	0.1004907
1	100	74	12	46	19.5	0.149	28	0	0	0.0551546

2- 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

Make sure you understand the output. In particular, do the rows represent the actual or predicted class? The columns?

```r
conf_matrix <- table(data$class, data$scored.class)
dimnames(conf_matrix) <- list(Actual = c("0", "1"), Predicted = c("0", "1"))
print(conf_matrix)
```

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

The above code creates a confusion matrix of the predicted class versus the actual class and renames the dimensions with the values they represent. The output shows us the accuracy of the predictions on both positive and negative values separately. This can be very useful for unbalanced data where overall accuracy results can be very misleading. The rows of the confusion table represent the actual values of the data, while the columns represent the predicted values. The main diagonal (upper-left corner to lower-right corner) represents to the correct values, in this case 119 True Positives (TPs) and 27 True Negatives (TNs) and the other fields represent the incorrect predictions, in this case 30 False Positives (FPs) and 5 False Negatives (FNs).

Calculate key statistics

First, we will write a function to generate key statistical measures used for later calculations. We will use the returned values in all the following functions

```r
key_stats <- function(actual, predicted) {
  TN <- sum(actual == 0 & predicted == 0)
  FP <- sum(actual == 0 & predicted == 1)
  FN <- sum(actual == 1 & predicted == 0)
  TP <- sum(actual == 1 & predicted == 1)

  stats <- list(TN = TN, FP = FP, FN = FN, TP = TP)
  return(stats)
}
```

3 - Calculate 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.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

```r
calc_acc <- function(df, actual, predicted) {
  actual <- df[[actual]]
  predicted <- df[[predicted]]

  stats <- key_stats(actual, predicted)

  acc <- (stats$TP + stats$TN) / (stats$TP + stats$FP + stats$TN + stats$FN)

  return(acc)
}
```

The above code initializes a function to calculate the accuracy of the predictions.

Now let's calculate the accuracy rate for our particular case:

```r
acc <- calc_acc(data, 'class','scored.class')
print(acc)
```

```
## [1] 0.8066298
```

The provided classification model achieved an accuracy score of 80%.

4 - Classification Error Rate

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

$$\text{Classification Error Rate} = \frac{FP + FN}{TP + TN + FP + FN}$$

```
calc_cer <- function(df, actual, predicted) {
  actual <- df[[actual]]
  predicted <- df[[predicted]]
  stats <- key_stats(actual, predicted)

  cer <- (stats$FP + stats$FN) / (stats$TP + stats$TN + stats$FP + stats$FN)
  return(cer)
}
```

The above code initializes a function to calculate the classification error rate.

Now let's calculate the classification error rate for our particular case:

```
cer <- calc_cer(data, 'class', 'scored.class')
print(cer)
```

```
## [1] 0.1933702
```

The provided classification model has a classification error rate of 19%.

5 - 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.

$$Precision = \frac{TP}{TP + FP}$$

```
calc_prec <- function(df, actual,predicted){
  actual <- df[[actual]]
  predicted <- df[[predicted]]

  stats <- key_stats(actual, predicted)

  prec <- (stats$TP) / (stats$TP + stats$FP)
  return(prec)
}
```

The above code initializes a function to calculate the precision rate.

Now let's calculate the precision rate for our particular case:

```
prec <- calc_prec(data, 'class','scored.class')
print(prec)
```

```
## [1] 0.84375
```

The provided classification model achieved a precision rate of 84%.

6 - Sensitivity

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.

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

```r
calc_sens <- function(df, actual,predicted){
  actual <- df[[actual]]
  predicted <- df[[predicted]]

  stats <- key_stats(actual, predicted)

  sens <- (stats$TP) / (stats$TP + stats$FN)

  return(sens)
}
```

The above code initializes a function to calculate the sensitivity rate.

Now let's calculate the sensitivity rate for our particular case:

```r
sens <- calc_sens(data, 'class','scored.class')
print(sens)
```

```
## [1] 0.4736842
```

The provided classification model achieved a sensitivity rate of 47%.

7 - Specificity

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

$$\text{Specificity} = \frac{TN}{TN + FP}$$

```r
calc_spec <- function(df, actual, predicted){
  actual <- df[[actual]]
  predicted <- df[[predicted]]

  stats <- key_stats(actual, predicted)

  spec<- stats$TN / (stats$TN + stats$FP)
  return(spec)
}
```

The above code initializes a function to calculate the accuracy rate of a model.

Now let's calculate the accuracy rate for our particular case:

```
calc_spec(data, 'class', 'scored.class')
```

```
## [1] 0.9596774
```

The returned specificity of our predictions is 96%

8. 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.

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

A quick breakdown on F1_Score, Precision, and Sensitivity. Precision is useful when there is a high cost to having a false positive, for instance a false positive cancer diagnosis, and is used to evaluate how often a false positive is occurring for calculating costs. The priority is to capture as few false positives as possible, it would be awful and expensive to families if the patient went through chemo-therapy and never had cancer to begin with.

Sensitivity is used when positive classifications are a high priority and the cost of a false positive is low in comparison, such as malicious network behavior. The priority in sensitivity is to capture as many true positives as possible, even if it means picking up a few false positives.

F1 Score is a combination of the twe metrics, useful for when we want a well-balanced model.

To accomplish this, we use the functions for Precision and Sensitivity created in sections 5 and 6 respectively to create a new function to calculate the F1 Score:

```
calc_F1 <- function(data, actual,predicted){
  prec <- calc_prec(data, actual,predicted)
  sens <- calc_sens(data, actual,predicted)

  F1 <- (2 * prec * sens) / (prec + sens)

  return(F1)
}
```

```
F1_score <- calc_F1(data, 'class','scored.class')

print(F1_score)
```

```
## [1] 0.6067416
```

Using the functions, we find an F1 Score of 606.

9. F1 Bounds

What are the bounds on the F1 score? Show that the F1 score will always be between 0 and 1.

As long as the F1 is defined, its value must be greater than zero and less than or equal to 1. First, let's consider the formula for F1:

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

The key is that both precision and sensitivity are between 0 and 1.

This is because

$$\text{Precision} = \frac{TP}{TP + FP}$$

In the best case scenario, there are no false positives, the numerator equals the denominator, and the precision score is therefore 1. In the worst case scenario, there are no true positives, the numerator is 0, and the precision score is therefore 0.

Similarly,

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

In the best case scenario, there are no false negatives, the numerator equals the denominator, and the sensitivity score is therefore 1. In the worst case scenario, there are no true positives, the numerator is 0, and the sensitivity score is therefore 0.

I ignore the case where both precision and sensitivity are 0, since then there would be no F1 score defined at all.

Let's see how small we can make the F1 score: Well, we can have either the precision score or the sensitivity score equal 0. In that case, the numerator equals 0, and so the F1 score likewise equals 0. However, we can never get an F1 score less than 0 since if the precision score and sensitivity score are both positive then their product (even multiplied by 2) is positive, as is their sum. It follows that the lower bound for the F1 score is 0.

Now let's see how large we can make the F1 score: The product of two numbers between 0 and 1 is always less than both numbers, but the sum of those two numbers is always greater than either number. The difference between the product and sum is greatest closest to 0. So let's see what happens when we try to maximize the F1 score by making *both* the precision score and the sensitivity score 1:

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

$$\text{F1 Score} = \frac{2 \times 1 \times 1}{1 + 1} = 1$$

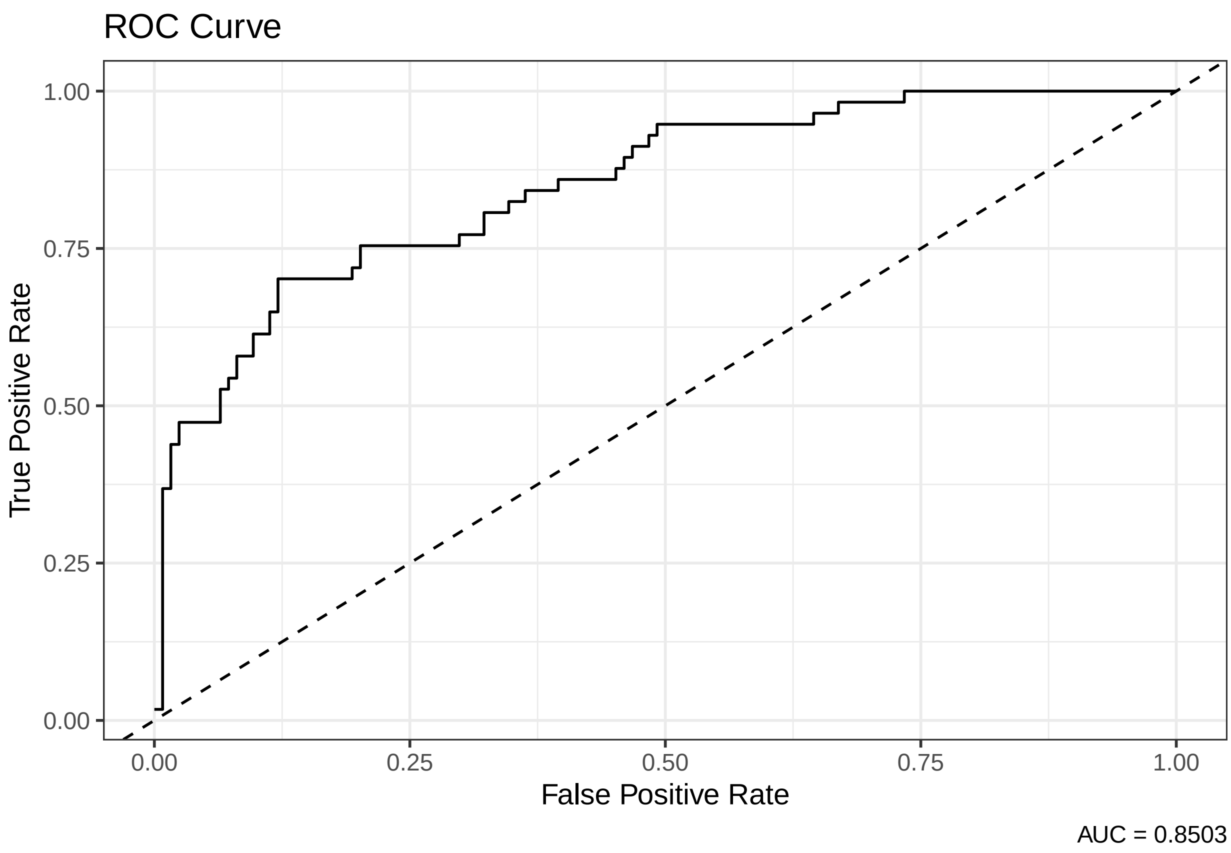
And so the bounds on the F1 score (as long as it's defined) are 0 and 1.

10 - 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.

```
plot_ROC <-
  function(actual, probability){
    actual <-
      actual[order(probability, decreasing = TRUE)]

    tpr <-
```

```
      cumsum(actual) / sum(actual)
    fpr <-
      cumsum(!actual) / sum(!actual)

    tpr_fpr_df <-
      data.frame(tpr, fpr, actual)

    fpr_df <-
      c(diff(tpr_fpr_df$fpr), 0)

    tpr_df <-
      c(diff(tpr_fpr_df$tpr), 0)

    auc <-
      round(sum(tpr_fpr_df$tpr * fpr_df) + sum(tpr_df * fpr_df)/2, 4)

    roc_plot <- ggplot() +
    geom_line(data = tpr_fpr_df, mapping = aes(x=fpr, y=tpr)) +
    geom_abline(linetype=2) +
    labs(title = "ROC Curve",
         caption = paste("AUC =", auc),
         x = "False Positive Rate",
         y = "True Positive Rate") +
    theme_bw()

    list(roc_plot = roc_plot, auc = auc)
}

result <- plot_ROC(data$class, data$scored.probability)

roc_plot <- result$roc_plot
auc <- result$auc
print(roc_plot)
```



```r
print(auc)
```

```
## [1] 0.8503
```

The resulting auc is 0.85.

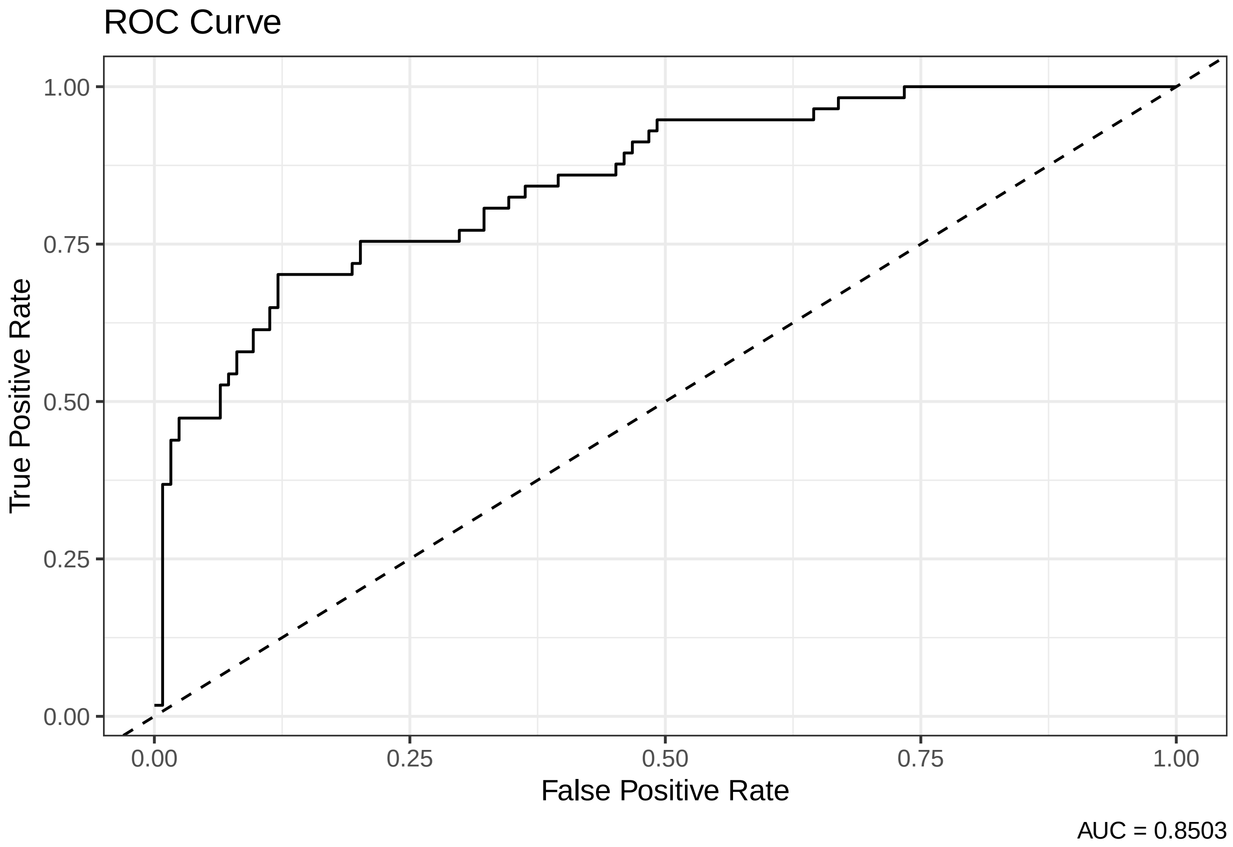
11 - Run Functions

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

Instead of running them separately, we created a wrapper function to run all the above functions on the given data and return the metrics as a list.

```r
run_all_functions <- function(df, actual, predicted, probability) {
  stats <- key_stats(df[[actual]], df[[predicted]])

  acc <- calc_acc(df, actual, predicted)

  cer <- calc_cer(df, actual, predicted)

  prec <- calc_prec(df, actual, predicted)

  sens <- calc_sens(df, actual, predicted)
  spec <- calc_spec(df, actual, predicted)

  f1_score <- calc_F1(df, actual, predicted)

  roc_result <- plot_ROC(df[[actual]], df[[probability]])
  roc_plot <- roc_result$roc_plot
  auc <- roc_result$auc

  result <- list(
    #key_stats = stats,
    accuracy = acc,
    classification_error_rate = cer,
    precision = prec,
    sensitivity = sens,
    specificity = spec,
    f1_score = f1_score,
    roc_plot = roc_plot,
    auc = auc
  )

  return(result)
}

result_summary <- run_all_functions(data, 'class', 'scored.class', 'scored.probability')

print(result_summary)
```

```
## $accuracy
## [1] 0.8066298
##
## $classification_error_rate
## [1] 0.1933702
##
## $precision
## [1] 0.84375
##
## $sensitivity
## [1] 0.4736842
##
## $specificity
## [1] 0.9596774
##
## $f1_score
## [1] 0.6067416
##
## $roc_plot
```



```
##
## $auc
## [1] 0.8503
```

12 - 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?

We investigated the caret package, specifically utilizing its functions like confusionMatrix(), sensitivity(), and specificity() to assess our classification model's performance. By applying these functions to our dataset, we compared the results with our own functions, ensuring consistency and accuracy in evaluating the model's predictive power and class-specific metrics.

The results obtained using the `caret` package align perfectly with our previous calculations. Specifically, the confusion matrix, accuracy, sensitivity, and specificity values from the `caret` package match the results we derived earlier, confirming the consistency and accuracy of our analysis.

```
cm <- confusionMatrix(data = as.factor(data$scored.class),
                      reference = as.factor(data$class),
                      positive = "1")
cm
```

```
## 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
##
```

```r
accuracy_equal <- result_summary$accuracy == cm$overall["Accuracy"]
print(accuracy_equal)
```

```
## Accuracy
##     TRUE
```

```r
specificity_equal <- result_summary$specificity == cm$byClass["Specificity"]
print(specificity_equal)
```

```
## Specificity
##        TRUE
```

```r
sensitivity_equal <- result_summary$sensitivity == cm$byClass["Sensitivity"]
print(sensitivity_equal)
```

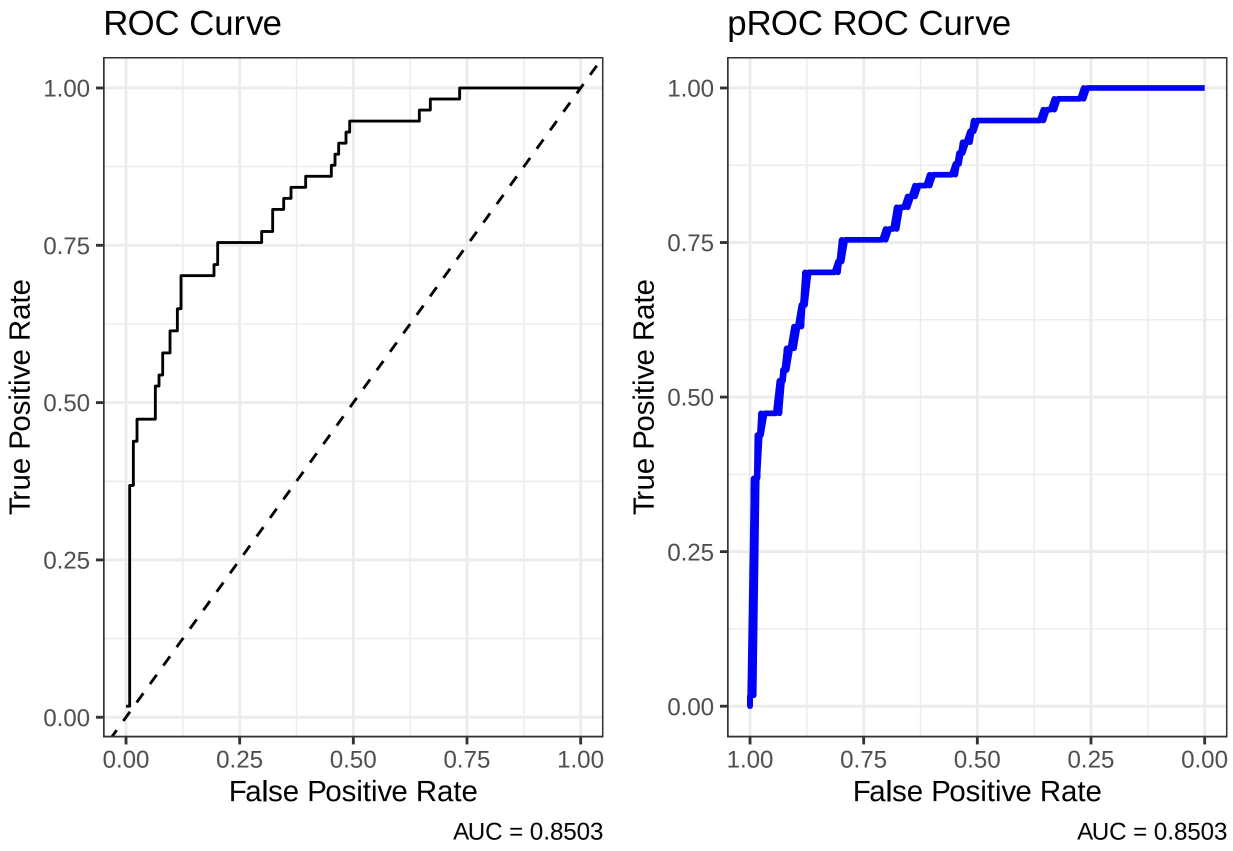
```
## Sensitivity
##        TRUE
```

13 - 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?

Using the pROC package returned the exact results as our functions. The AUC is identical at 0.8503 and the plots look to have the exact same shape. This further confirms the accuracy of our calculations. The pROC package does seem to return a plot with the opposite orientation than our plot, but since the AUC is the same, we can write that off as a difference of style.

```
roc_curve <- roc(data$class, data$scored.probability)

proc_auc <- round(auc(roc_curve), 4)

roc_data <- data.frame(fpr = roc_curve$specificities,
                       tpr = roc_curve$sensitivities)

proc_plot <- ggplot(roc_data, aes(x = fpr, y = tpr)) +
  geom_line(color = "blue", linetype = 1, size = 1) +
  labs(title = "pROC ROC Curve",
       caption = paste("AUC =", auc),
       x = "False Positive Rate",
       y = "True Positive Rate") +
  theme_bw() +
  scale_x_reverse() # proc uses a different orientation

grid.arrange(roc_plot, proc_plot, ncol = 2)
```



```
auc_equal <- result_summary$auc == proc_auc
print(auc_equal)
```

```
## [1] TRUE
```

Summary

In this analysis, we explored the performance metrics of a diabetes classification model. Our methodology involved the following steps:

- Custom Functions: We developed custom functions to calculate essential metrics such as accuracy, precision, sensitivity, and specificity. These functions allowed us to evaluate the model's performance based on its predictions and actual class labels.
- Cross-Checking with Prebuilt Packages: To ensure accuracy and reliability, we cross-checked our results using prebuilt packages. Specifically, we utilized the caret package for the confusion matrix and accuracy calculation. Additionally, the pROC package helped us generate the ROC curve and compute the AUC.