Data 621 - Homework 2

Group 4 Layla Quinones, Ian Costello, Dmitriy Burtsev & Esteban Aramayo

OCtober 10, 2021

```
library(kableExtra)
library(tidyverse)
library(tidymodels)
library(caret)
library(pROC)
```

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

```
HW2_url = "https://raw.githubusercontent.com/MsQCompSci/Data621Group4/main/HW2/classification-output-da
df <- read.csv(HW2_url)
kable(head(df)) %>%
  kable_styling()
```

2. 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?

The rows represents the *actual* class of for the observation. Given the medical statistics in the data, let's say this relates to the presence (or absence) of a medical condition. In this example, zero represents the absence

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

	0	1
0	119	5
1	30	27

of the condition, one represents the presence. The columns are the scored class, meaning based on the data provided, the prediction of whether (to continue with our example) a condition *may be* present. Zero again representing absence, and one representing presence.

Now, where the actual and scored classes differ are the errors. Errors can be false positives (type I) or false negatives (type II). In table 1 below, the 30 value are the false negatives, where the predictor scored zero, but the actual class is one. And the 5 value is the false positive, where the predictor scored one, but the actual class is zero.

```
kable(table(df$class, df$scored.class)) %>%
  kable_styling()
```

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.

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

```
confmat <- function(df){
  data.frame(TN=nrow(df[df$class==0 & df$scored.class==0,]),
             FN=nrow(df[df$class==1 & df$scored.class==0,]),
             FP=nrow(df[df$class==0 & df$scored.class==1,]),
             TP=nrow(df[df$class==1 & df$scored.class==1,])

  )
}

confmat(df)
```

```
##    TN FN FP TP
## 1 119 30  5 27
```

```
accuracy_score <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)

  # Calculate accuracy from confusion matrix
  ( cm$TP + cm$TN ) / ( cm$TP + cm$FP + cm$TN + cm$FN )

}

# test accuracy_score
print(accuracy_score(df))
```

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

4. 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.

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

```
class_err_rate <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)

  # Calculate Classification Error Rate from confusion matrix
  ( cm$FP + cm$FN ) / ( cm$TP + cm$FP + cm$TN + cm$FN )

}

# test class_err_rate
print(class_err_rate(df))
```

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

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

```
as <- accuracy_score(df)
cer <- class_err_rate(df)
print(as)
```

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

```
print(cer)
```

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

```
print(as + cer)
```

```
## [1] 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.

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

```r
precision_score <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)

  # Calculate Precision from confusion matrix
  cm$TP / ( cm$TP + cm$FP )

}

# test precision_score
print(precision_score(df))
```

```
## [1] 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.

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

```r
sensitivity_score <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)

  # Calculate Sensitivity from confusion matrix
  cm$TP / ( cm$TP + cm$FN )

}

# test sensitivity_score
print(sensitivity_score(df))
```

```
## [1] 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.

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

```r
specificity_score <- function(df){

  # Get confusion matrix from df data
  cm <- confmat(df)
```

```
  # Calculate Specificity from confusion matrix
  cm$TN / ( cm$TN + cm$FP )

}

# test specificity_score
print(specificity_score(df))
```

```
## [1] 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.

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

```
F1_score <- function(df){

  precis <- precision_score(df)
  sensit <- sensitivity_score(df)

  # Calculate F1 Score
  ( 2 * precis * sensit ) / ( precis + sensit )

}

# test F1_score
print(F1_score(df))
```

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

9. 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* $ab < a$.*)*

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.

```r
AUC_ROC <- function(df){

  # Purpose: It returns a list that includes the plot of the ROC curve and a
  # vector that contains the calculated area under the curve (AUC)
  #
  # Note: AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve

  library(ggplot2)

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


  TPR_vector <- c()
  FPR_vector <- c()


  # Build the TPR and FPR vectors using the thresholds vector
  for (i in 1:length(thresholds)){

    scored_class <- ifelse(df$scored.probability >= thresholds[i], 1, 0)

    rev_df <- data.frame(scored.class = scored_class, class = df$class)

    df_table <- with(rev_df, table(scored.class, class))

    TPR <- (df_table[4]) / (df_table[4] + df_table[3])

    FPR <- (df_table[2] / (df_table[2] + df_table[1]))

    TPR_vector[i] <- TPR
    FPR_vector[i] <- FPR

  }

  # define dataframe to be used for the ROC curve
  plot_df <- data.frame(TRUE_POSITIVE = TPR_vector, FALSE_POSITIVE = FPR_vector)




  # In order to roughly calculate the area under the curve, we must remove the NA values
  AUC_df <- plot_df[complete.cases(plot_df),]

  # Now to calculate the AUC
  x <- abs(diff(AUC_df$FALSE_POSITIVE))
  y <- AUC_df$TRUE_POSITIVE

  area_under_curve <- sum(x*y)

  plot_title <- paste0("Receiver Operating Characteristics Curve - ( AUC = ", area_under_curve, " )")

  # store ROC plot in a variable
```

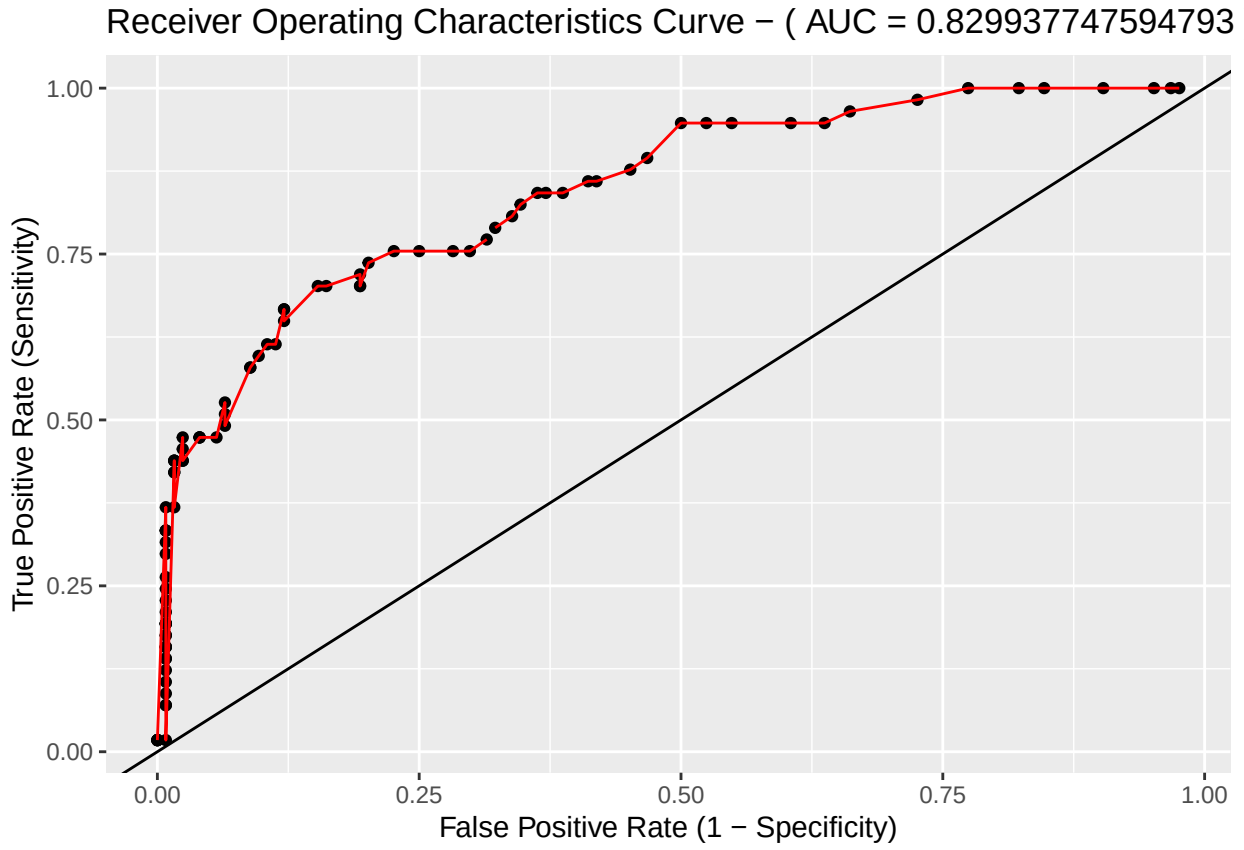
```
  ROC_plot <- ggplot(plot_df, aes( x = FALSE_POSITIVE, y = TRUE_POSITIVE)) +
    geom_point() +
    geom_line(col="red") +
    geom_abline(intercept = 0, slope = 1) +
    labs(title = plot_title,
         x = "False Positive Rate (1 - Specificity)",
         y = "True Positive Rate (Sensitivity)")

  return(list(ROC_plot, area_under_curve))
}

 # ROC_list <- AUC_ROC(df)
 # ROC_plot <- ROC_list[[1]]
 # AUC <- ROC_list[[2]]
 #
 # ROC_plot
 # print(AUC)
```

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

```
metrics_names <- c("Accuracy",
                   "Classification Error Rate",
                   "Precision",
                   "Sensitivity",
                   "Specificity",
                   "F1 Score")

metrics_values <- c(accuracy_score(df),
             class_err_rate(df),
             precision_score(df),
             sensitivity_score(df),
             specificity_score(df),
             F1_score(df))



metrics_df <- data.frame(metrics_names, metrics_values)
colnames(metrics_df) <- c("Metric","Value")

kable(metrics_df) %>%
  kable_styling()
```

```
ROC_list <- AUC_ROC(df)
ROC_plot <- ROC_list[[1]]
AUC <- ROC_list[[2]]

 ROC_plot
```

Metric	Value
Accuracy	0.8066298
Classification Error Rate	0.1933702
Precision	0.8437500
Sensitivity	0.4736842
Specificity	0.9596774
F1 Score	0.6067416




```
print(AUC)
```

```
## [1] 0.8299377
```

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?

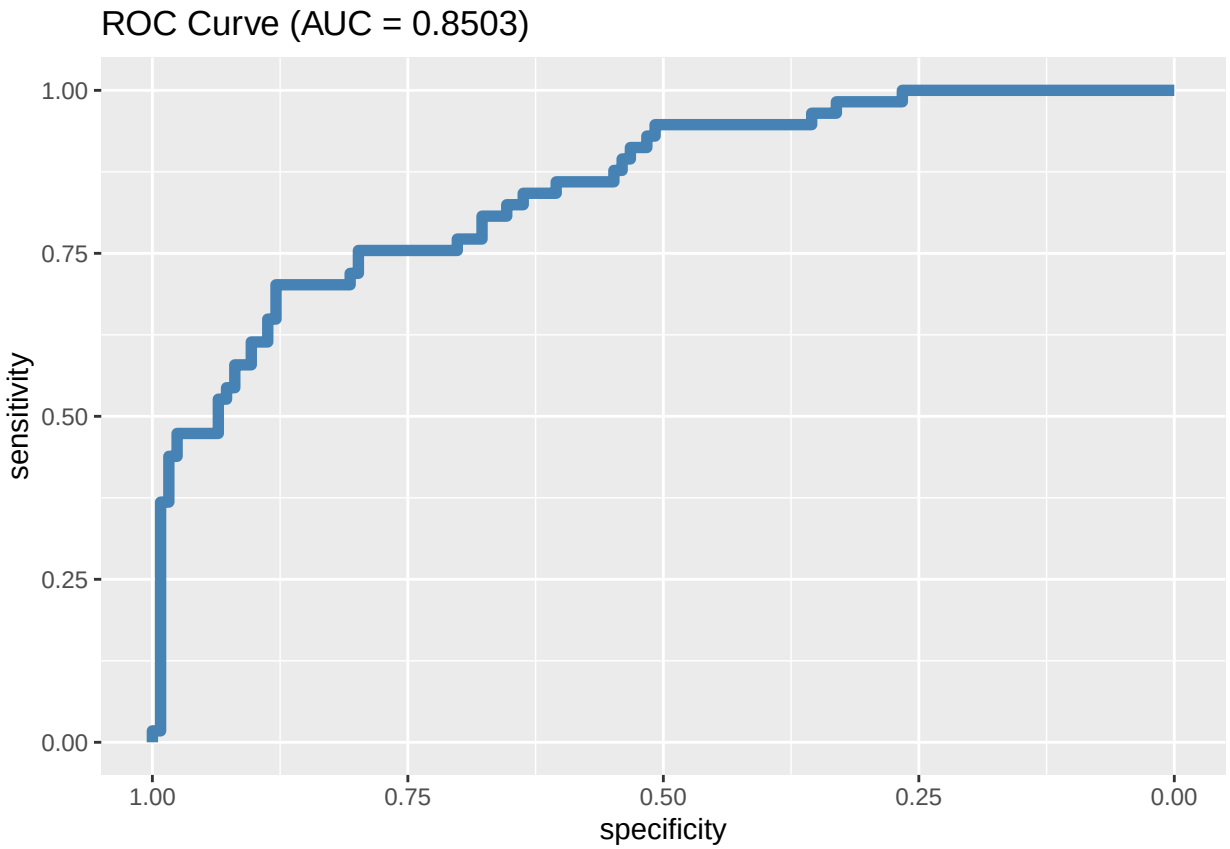
[IC Added: I really liked this overview of the `caret` package and its capabilities. https://www.machinelearningplus.com/machine-learning/caret-package/]

	Performance Measure	Caret function result	Custom function result	Difference
Accuracy	Accuracy	0.8066298	0.8066298	0
Sensitivity	Sensitivity	0.4736842	0.4736842	0
Specificity	Specificity	0.9596774	0.9596774	0

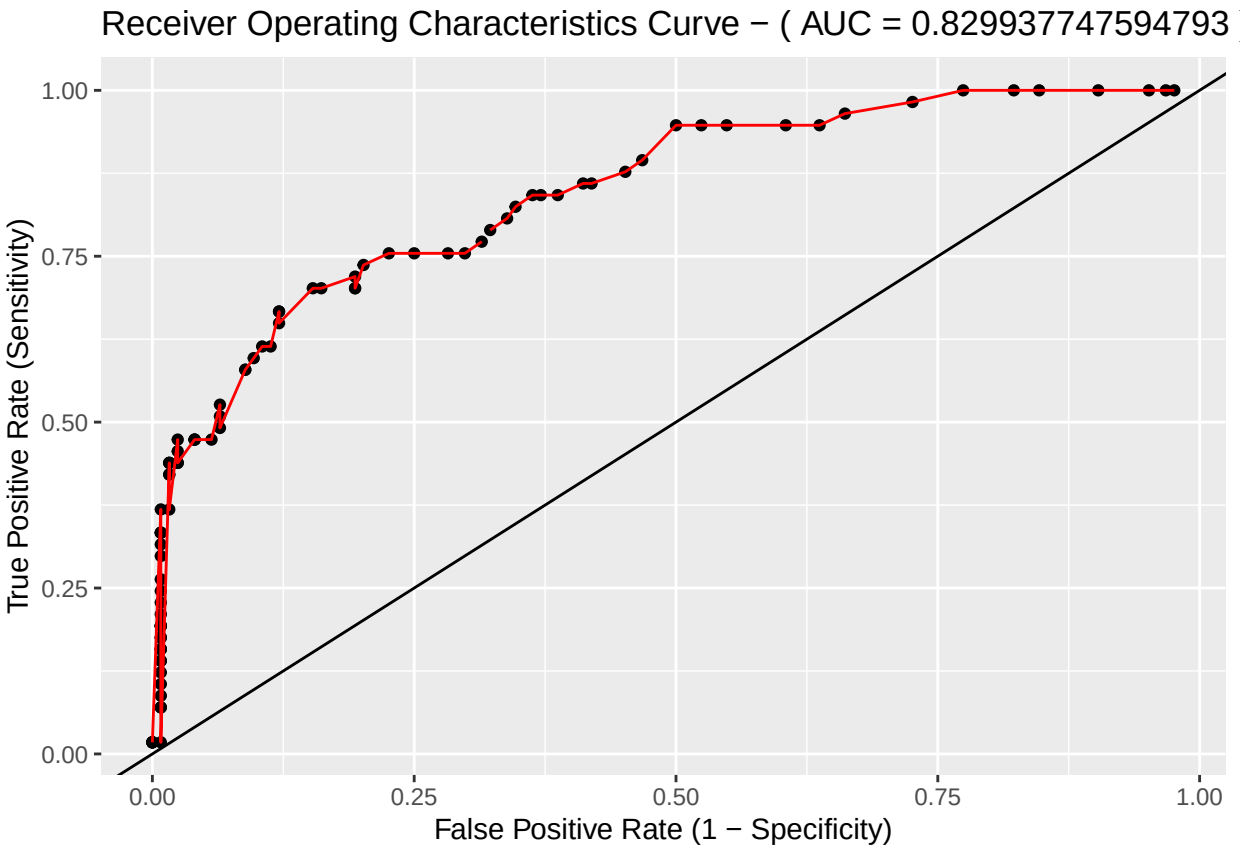
```
# Calculate the confusion matrix using Caret function
caret_cm <- caret::confusionMatrix(as.factor(df$scored.class),
                                   as.factor(df$class), positive = "1")


measure_names <- c("Accuracy", "Sensitivity", "Specificity")

# Calculate performance measures using
caret_funcs_values <- c(caret_cm$overall["Accuracy"],
                    caret_cm$byClass["Sensitivity"],
                    caret_cm$byClass["Specificity"])

custom_funcs_values <- c(accuracy_score(df),
                     sensitivity_score(df),
                     specificity_score(df))

comparison_df <- data.frame(measure_names, caret_funcs_values, custom_funcs_values)


comparison_df <- comparison_df %>%
  mutate(Difference = caret_funcs_values - custom_funcs_values)

colnames(comparison_df) <- c("Performance Measure",
                             "Caret function result",
                             "Custom function result",
                             "Difference")


# Show results comparison
kable(comparison_df) %>%
  kable_styling()
```

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?

```
#define object to plot and calculate AUC
rocobj <- pROC::roc(df$class, df$scored.probability)
auc <- round(pROC::auc(df$class, df$scored.probability),4)

#create ROC plot
ggroc(rocobj, colour = 'steelblue', size = 2) +
  ggtitle(paste0('ROC Curve ', '(AUC = ', auc, ')'))
```




```
ROC_list <- AUC_ROC(df)
ROC_plot <- ROC_list[[1]]
AUC <- ROC_list[[2]]

ROC_plot
```


Receiver Operating Characteristics Curve − (AUC = 0.829937747594793
True Positive Rate (Sensitivity)
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
False Positive Rate (1 − Specificity)