Crystal Mosley
Predict 422
Project #2

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

```
library(readr)
library(ROCR)

#read in classification data
class.data<-read.csv("classification-output-data.csv")
class.data <- class.data[,c("class","scored.class","scored.probability")]

confusion.matrix <- table(class.data$scored.class, class.data$class)
con.mat <- apply(apply(confusion.matrix, 1, rev), 1, rev)

#set column and row names for con.mat
colnames(con.mat) <- c("Class (Actual) 1", "Class (Actual) 0"); rownames(con.mat) <- c("Scored (Predicted) 1",
"Scored (Predicted) 0"); con.mat
```

OUTPUT:

```
                      Class (Actual) 1   Class (Actual) 0
 Scored (Predicted) 1        27                 5
 Scored (Predicted) 0        30               119
```

The rows represent the predicted class, and the columns represent the actual class.

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

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

```
#dt = datatable
#df = dataframe
class.accuracy <- function(df, predicted, actual) {
  dt <- table(df[ ,predicted], df[ ,actual])
  confusion.matrix2 <- apply(apply(dt, 1, rev), 1, rev)
  TP <- confusion.matrix2[1,1]; FP <- confusion.matrix2[1,2]
  FN <- confusion.matrix2[2,1]; TN <- confusion.matrix2[2,2]
  accuracy <- (TP + TN) / (TP + FP + TN + FN)
  return (accuracy)
}
```

3. Write a function that take three inputs, the data set as a dataframe with actual and predicted classifications identified and returns the classification error rate of the predictions. Verify that you get an accuracy and an error rate that sums to one.

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

```
#dt = datatable
#df = dataframe
class.error <- function(df, predicted, actual) {
  dt <- table(df[ ,predicted], df[ ,actual])
  confusion.matrix2 <- apply(apply(dt, 1, rev), 1, rev)
  TP <- confusion.matrix2[1,1]; FP <- confusion.matrix2[1,2]
  FN <- confusion.matrix2[2,1]; TN <- confusion.matrix2[2,2]
  error <- (FP + FN) / (TP + FP + TN + FN)
  return (error)
}
```

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

```
class.accuracy(class.data, "scored.class", "class") + class.error(class.data, "scored.class", "class")
```

OUTPUT: 1 – yes, the accuracy and error rate sums to one.

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

Precision = TP / TP + FP

```
#dt = datatable
#df = dataframe
class.precision <- function(df, predicted, actual) {
  dt <- table(df[ ,predicted], df[ ,actual])
  confusion.matrix2 <- apply(apply(dt, 1, rev), 1, rev)
  TP <- confusion.matrix2[1,1]; FP <- confusion.matrix2[1,2]
  FN <- confusion.matrix2[2,1]; TN <- confusion.matrix2[2,2]
  precision <- TP / (TP + FP)
  return (precision)
}
```

5. Write a function that take three inputs, 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 = TP / TP + FN

```
#dt = datatable
#df = dataframe
class.sensitivity <- function(df, predicted, actual) {
  dt <- table(df[ ,predicted], df[ ,actual])
  confusion.matrix2 <- apply(apply(dt, 1, rev), 1, rev)
TP <- confusion.matrix2[1,1]; FP <- confusion.matrix2[1,2]
  FN <- confusion.matrix2[2,1]; TN <- confusion.matrix2[2,2]
  sensitivity <- TP / (TP + FN)
  return (sensitivity)
}
```

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

Specificity = TN / TN + FP

```
#dt = datatable
#df = dataframe
class.specificity <- function(df, predicted, actual) {
  dt <- table(df[ ,predicted], df[ ,actual])
  confusion.matrix2 <- apply(apply(dt, 1, rev), 1, rev)
  TP <- confusion.matrix2[1,1]; FP <- confusion.matrix2[1,2]
  FN <- confusion.matrix2[2,1]; TN <- confusion.matrix2[2,2]
  specificity <- TN / (TN + FP)
  return (specificity)
}
```

7. Write a function that take three inputs, the data set as a dataframe with actual and predicted classifications identified and returns the F1 score of the predictions.

F1 Score = 2 x Precision x Sensitivity / Precision + Sensitivity

```
#dt = datatable
#df = dataframe
class.f1.score <- function(df, predicted, actual) {
  dt <- table(df[ ,predicted], df[ ,actual])
  confusion.matrix2 <- apply(apply(dt, 1, rev), 1, rev)
  TP <- confusion.matrix2[1,1]; FP <- confusion.matrix2[1,2]
  FN <- confusion.matrix2[2,1]; TN <- confusion.matrix2[2,2]
  f1.score <- (2 *(TP / (TP + FP)) * (TP / (TP + FN))) / ((TP / (TP + FP)) + (TP / (TP + FN)))
  return (f1.score)
}
```

8. Let's consider the following question: 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 = 2 x Precision x Sensitivity / Precision + Sensitivity

$$= 2\left(\frac{(\frac{A}{A+B})(\frac{A}{A+C})}{\left(\frac{A}{A+B}\right)+\left(\frac{A}{A+C}\right)}\right)$$

$$= \frac{2A^2}{2A^2+AB+AC}$$

$$= \frac{4A}{4A+B+C}$$

If A=0, then $= \frac{4*0}{(4*0)+B+C} = \frac{0}{B+C} = 0$

9. Write a function that generates an ROC curve from a data set using two inputs, a true classification column (i.e., class) and a probability column (i.e., scored.probability). Your function should return the plot of the ROC curve and the calculated area under the ROC curve (AUC). Note that I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.

```
ROC.curve <- function(df, probabilities, actual) {
  thresholds <- seq(0, 1, 1/180) #0-180 data points

  true.pos.rate <- false.pos.rate <- numeric(length(thresholds))
  pos <- sum(df[,actual] == 1) #positive sum
  neg <- sum(df[,actual] == 0) #negative sum

  #for loop
  for (i in 1:length(thresholds)) {

    #Upper bound probability threshold
    class.subset <- subset(df, df[ ,probabilities] <= thresholds[i])

    #Condition numerator
    true.pos <- sum(class.subset[class.subset[,actual] == 1, probabilities] > 0.1) #TP
    true.neg <- sum(class.subset[class.subset[,actual] == 0, probabilities] <= 0.1) #TN
    false.pos <- sum(class.subset[class.subset[,actual] == 0, probabilities] > 0.1) #FP
    false.neg <- sum(class.subset[class.subset[,actual] == 1, probabilities] <= 0.1) #FN

    # true.pos <- sum(class.subset[class.subset[,actual] == 1, probabilities] > 0.3) #TP
    # true.neg <- sum(class.subset[class.subset[,actual] == 0, probabilities] <= 0.3) #TN
    # false.pos <- sum(class.subset[class.subset[,actual] == 0, probabilities] > 0.3) #FP
    # false.neg <- sum(class.subset[class.subset[,actual] == 1, probabilities] <= 0.3) #FN

    # true.pos <- sum(class.subset[class.subset[,actual] == 1, probabilities] > 0.5) #TP
    # true.neg <- sum(class.subset[class.subset[,actual] == 0, probabilities] <= 0.5) #TN
    # false.pos <- sum(class.subset[class.subset[,actual] == 0, probabilities] > 0.5) #FP
    # false.neg <- sum(class.subset[class.subset[,actual] == 1, probabilities] <= 0.5) #FN

    # true.pos <- sum(class.subset[class.subset[,actual] == 1, probabilities] > 0.1) #TP
    # true.neg <- sum(class.subset[class.subset[,actual] == 0, probabilities] <= 0.3) #TN
    # false.pos <- sum(class.subset[class.subset[,actual] == 0, probabilities] > 0.5) #FP
    # false.neg <- sum(class.subset[class.subset[,actual] == 1, probabilities] <= 0.7) #FN

    # true.pos <- sum(class.subset[class.subset[,actual] == 1, probabilities] > 0.7) #TP
    # true.neg <- sum(class.subset[class.subset[,actual] == 0, probabilities] <= 0.7) #TN
    # false.pos <- sum(class.subset[class.subset[,actual] == 0, probabilities] > 0.7) #FP
    # false.neg <- sum(class.subset[class.subset[,actual] == 1, probabilities] <= 0.7) #FN
```

#When running the ROC.curve function that I created, using 0.1/0.3/0.5/0.7 as the thresholds all held the same output of 0.8504527, as well as the same plotted ROC curve. I then wanted to see if having different thresholds within the same function would cause a fluctuation, so I tried all 0.1-0.7 (odd nums only) at the same time and the output was different: 0.6465054 as well as a different ROC curve. I chose 0.1 as the final since it looks like it doesn't matter as long as the ROC output and curve curved plot look good.

```
    true.pos.rate[i] <- 1 - (true.pos + false.neg) / pos  #sensitivity
    false.pos.rate[i] <- 1 - (true.neg + false.pos) / neg  #specificity
  }

#plot ROC
  plot(false.pos.rate, true.pos.rate, type = "l", lwd = 2,
       xlab = "False.Pos.Rate (1-specificity)",
       ylab = "True.Pos.Rate (sensitivity)")

  #add straight line through plot
  abline(0, 1)

  false.pos.rate.diff <- c(abs(diff(false.pos.rate)), 0); true.pos.rate.diff <- c(abs(diff(true.pos.rate)), 0) #diff() returns n-1
suitably lagged and iterated differences


  #calculate the area under the ROC curve using the formula for the area of a trapezoid
  AUC <- sum(true.pos.rate * false.pos.rate.diff) - sum(true.pos.rate.diff * false.pos.rate.diff) / 2
  return (AUC)
}
```

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

#error
class.error(class.data, "scored.class", "class")

OUTPUT: 0.1933702

#accuracy
class.accuracy(class.data, "scored.class", "class")

OUTPUT: 0.8066298

#precision
class.precision(class.data, "scored.class", "class")

OUTPUT: 0.84375

#sensitivity
class.sensitivity(class.data, "scored.class", "class")

OUTPUT: 0.4736842

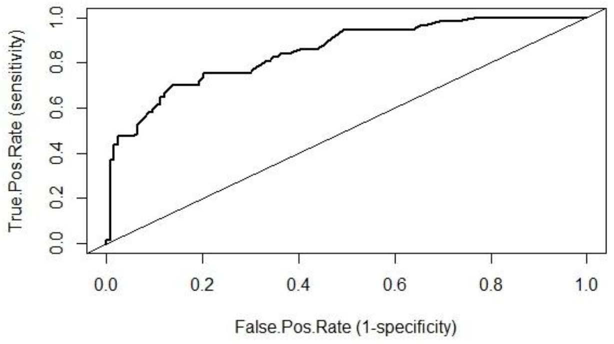
#specificity
class.specificity(class.data, "scored.class", "class")

OUTPUT: 0.9596774

#f1 score
class.f1.score(class.data, "scored.class", "class")

OUTPUT: 0.6067416

#ROC curve
ROC.curve(class.data, "scored.probability", "class")



OUTPUT: 0.8504527

11. 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) *#Classification and Regression Training package*
#drop missing data
data = na.omit(class.data)

#confusionMatrix() calculates a cross tabulation of obs. and pred. classes with assoc. stats
confusionMatrix(class.data$scored.class, class.data$class, class.data$scored.probability)
#receiving error: `data` and `reference` should be factors with the same levels.

#caret package using sensitivity function
sensitivity(factor(class.data$scored.class), factor(class.data$class), positive = "1")

Caret OUTPUT: 0.4736842
My OUTPUT: 0.4736842

#caret package using specificity function
specificity(factor(class.data$scored.class), factor(class.data$class), negative = "0")

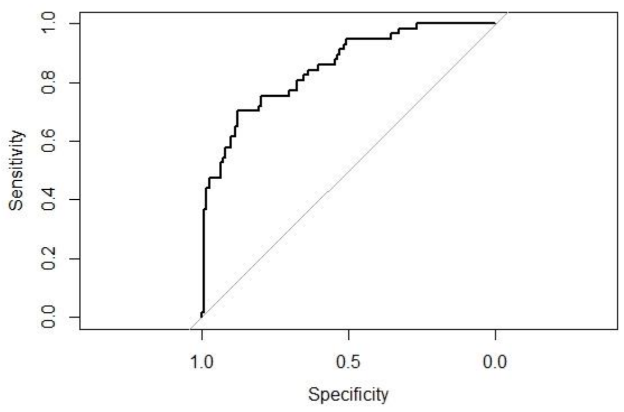
Caret OUTPUT: 0.9596774
My OUTPUT: 0.9596774

I could not get the confusionMatrix() to work without errors, so I’m not sure how it matched up. Both the sensitivity an d specificity functions matched equally.

12. 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)

    plot(roc(class.data$class, class.data$scored.probability, smooth=F))



    auc(class.data$class, class.data$scored.probability)

    pROC OUTPUT: Area under the curve: 0.8503

References

Horton, Bob. “Calculating AUC: the Area under a ROC Curve.” *Revolutions*, blog.revolutionanalysics.com/2016/11/calculating-auc.html.

Model Evaluation - Classification. www.saedsayad.com/model_evaluation_c.htm.

Numerical Integration. http://calculus.seas.upenn.edu/?n=Main.NumericalIntegration.

Plotting and Interpreting an ROC Curve. http://gim.unmc.edu/dxtests/roc2.htm.