DATA 621 Business Analytics & Data Mining

Homework #2 Classification Metrics

Kyle Gilde

3/11/2018

- Overview
- Deliverables
- Instructions


```
##           installed_and_loaded.packages.
## prettydoc                           TRUE
## tidyverse                           TRUE
## caret                               TRUE
## pROC                                TRUE
## zoo                                 TRUE
```

Overview

In this homework assignment, you will work through various classification metrics. You will be asked to create functions in R to carry out the various calculations. You will also investigate some functions in packages that will let you obtain the equivalent results. Finally, you will create graphical output that also can be used to evaluate the output of classification models, such as binary logistic regression.

Deliverables

Upon following the instructions below, use your created R functions and the other packages to generate the classification metrics for the provided data set. A write-up of your solutions submitted in PDF format.

Instructions

Complete each of the following steps as instructed:

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

```
class_df <- read.csv("https://raw.githubusercontent.com/kylegilde/D621-Data-Mi
```

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?

In the confusion matrix below, the rows represent the predicted classes, and the columns represent the actual classes.

```
class_vars <- subset(class_df, select = c(scored.class, class))

table(class_vars)
```

```
##              class
## scored.class   0   1
##             0 119  30
##             1   5  27
```

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.

```r
accuracy_calc <- function(df){
  # Takes a 2-column df where the 1st column is the predicted class
  # The 2nd column is the actual class (0 or 1)
  # Calculates the number of true positives & negatives
  # Returns the accuracy rate, the proportion of true predictions
  TP <- sum(df[, 2] == 1 & df[, 1] == 1)
  TN <- sum(df[, 2] == 0 & df[, 1] == 0)
  (TP + TN)/nrow(df)
}


(accuracy_value <- accuracy_calc(class_vars))
```

```
## [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.

```r
classification_error_rate <- function(df){
  # Takes a 2-column df where the 1st column is the predicted class
  # The 2nd column is the actual class (0 or 1)
  # Calculates the number of false positives & negatives
  # Returns the classification error rate, the proportion of false predictions
  FP <- sum(df[, 2] == 0 & df[, 1] == 1)
  FN <- sum(df[, 2] == 1 & df[, 1] == 0)
  (FP + FN)/nrow(df)
}

(cer <- classification_error_rate(class_vars))
```

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

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

They do sum to 1.

```r
cer + accuracy_value == 1
```

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

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_calc <- function(df){
  # Takes a 2-column df where the 1st column is the predicted class
  # The 2nd column is the actual class (0 or 1)
  # Calculates the number of true & false positives
  # Returns the precision rate, the proportion of the predicted positives that
  TP <- sum(df[, 2] == 1 & df[, 1] == 1)
  FP <- sum(df[, 2] == 0 & df[, 1] == 1)
  TP/(TP + FP)
}

(precision_value <- precision_calc(class_vars))
```

```
## [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_calc <- function(df, threshold = .5){
  # Takes a 2-column df
  # The 1st column is either the predicted class (0 or 1) or the predicted cla
  # The default class threshold is .5
  # & it works with either the predicted class (0 or 1) or the predicted class
  # The 2nd column is the actual class (0 or 1)
  # Calculates the number of true positives & false negatives
  # Returns the sensitivity rate
  # AKA the true positive rate, the recall, or probability of detection,
  # the proportion of correctly identified positives
  TP <- sum(df[, 1] > threshold & df[, 2] == 1)
  FN <- sum(df[, 1] <= threshold & df[, 2] == 1)
  TP/(TP + FN)
}

(sensitivity_value <- sensitivity_calc(class_vars))
```

```
## [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.

```r
specificity_calc <- function(df, threshold = .5){
  # Takes a 2-column df
  # The 1st column is either the predicted class (0 or 1) or the predicted cla
  # The default class threshold is .5
  # & it works with either the predicted class (0 or 1) or the predicted class
  # The 2nd column is the actual class (0 or 1)
  # Calculates the number of true positives & false negatives
  # Returns the specificity rate,
  # AKA the true negative rate & the proportion of correctly identified negati
  TN <- sum(df[, 1] <= threshold & df[, 2] == 0)
  FP <- sum(df[, 1] > threshold & df[, 2] == 0)
  TN/(TN + FP)
}

(specificity_value <- specificity_calc(class_vars))
```

```
## [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.

```r
F1_score_calc <- function(df){
  # Takes a 2-column df where the 1st column is the predicted class
  # The 2nd column is the actual class
  # Calculates the precision & sensitivity
  # Returns the F1 score
  precision_value <- precision_calc(class_vars)
  sensitivity_value <- sensitivity_calc(class_vars)
  (2 * precision_value * sensitivity_value)/(precision_value + sensitivity_valu
}

(F1_score_value <- F1_score_calc(class_vars))
```

```
## [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.

```r
set.seed(5)
n_sims <- 100000

# Create a 3 column matrix representing true positive, false positive & false
some_possible_inputs <- data.frame(
  a = c(runif(n_sims), 0, 1),
  b = c(runif(n_sims), 0, 1),
  c = c(runif(n_sims), 0, 1)
)

F1_score_generator <- function(df){
  # Takes a 3-column df of true positive, false positive & false negative valu
  # Calculates the precision & sensitivity
  # Returns a vector of the F1 scores

  precision_values <- mapply(function(TP, FP) TP/(TP + FP), df[, 1], df[, 2])
  sensitivity_values <- mapply(function(TP, FN) TP/(TP + FN), df[, 1], df[, 3]

  (2 * precision_values * sensitivity_values)/(precision_values + sensitivity_v
}


F1_values <- F1_score_generator(some_possible_inputs)
```

The F1 score simulation produced a minimum & maximum between 0 and 1. Some nans are produced if the sum of the true positive and false negative values or the true positive and false positive values is zero since this divides by zero.

```r
(min_value <- min(F1_values, na.rm = T))
```

```
## [1] 1.723105e-06
```

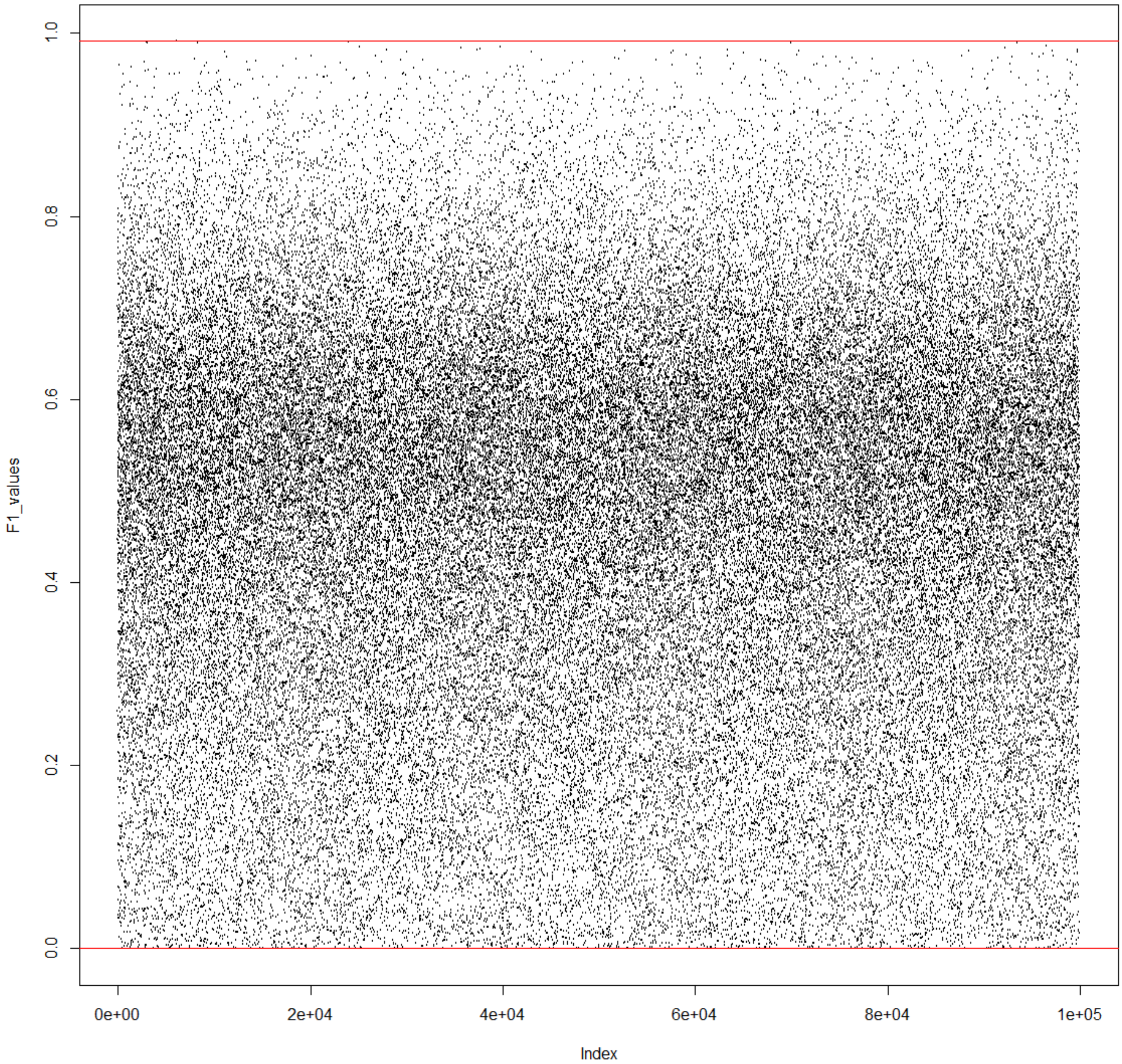
```r
(max_value <- max(F1_values, na.rm = T))
```

```
## [1] 0.9911914
```

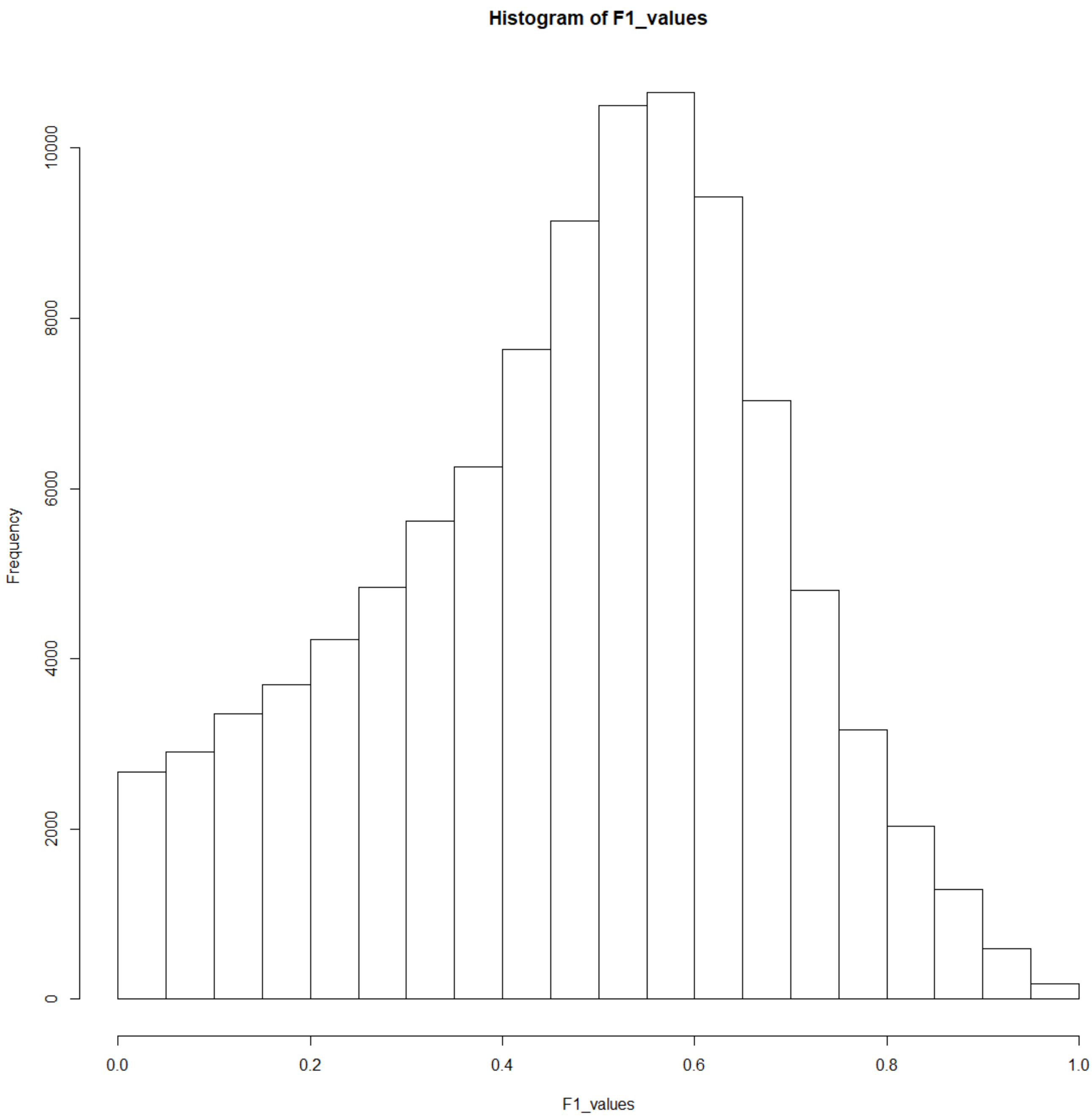
```r
sum(is.nan(F1_values))
```

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

```r
plot(F1_values, cex = .01)
abline(h = min_value, col = "red")
abline(h = max_value, col = "red")
```

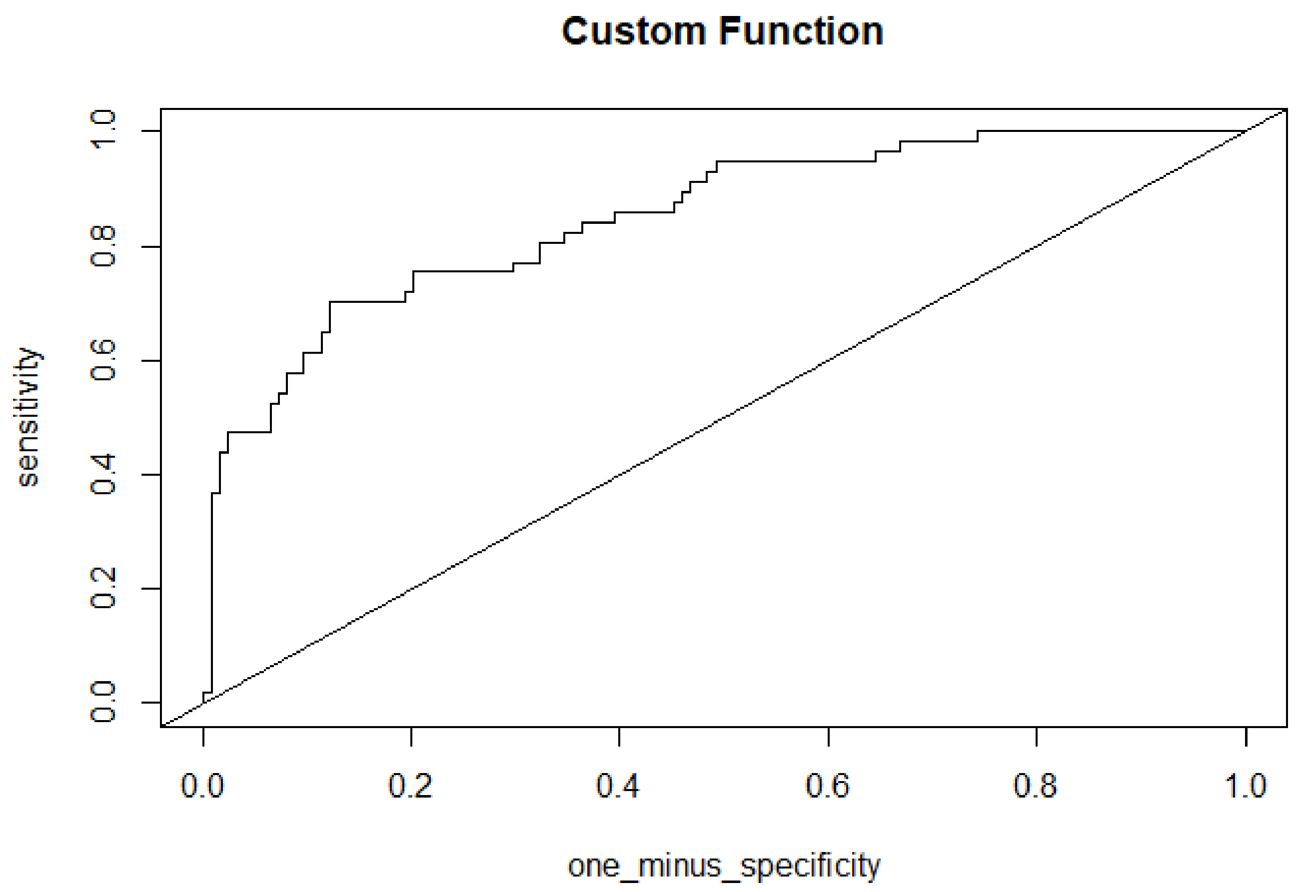



```r
hist(F1_values)
```



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.

```
class_prob_df <- subset(class_df, select = c(scored.probability, class))

receiver_operating_characteristic <- function(df, intervals = 10000){
  # Takes a 2-column df
  # The  1st column is the predicted class probablity
  # The 2nd column is the actual class (0 or 1)
  # intervals creates the number of thresholds to use between 0 and 1
  # Calcalutes the sensitivity & 1-specificity for all thresholds
  # Prints the ROC curve plot & returns the AUC value
  # AUC reference: https://stackoverflow.com/questions/4954507/calculate-the-a

  thresholds <- seq(0, 1, by = 1/intervals)

  sensitivity <- sort(sapply(thresholds, function(x) sensitivity_calc(df, thre

  one_minus_specificity <- sort(1 - sapply(thresholds, function(x) specificity_

  #create plot
  plot(sensitivity ~ one_minus_specificity, type = "s", xlim=c(0, 1), ylim=c(0,
  abline(a = 0, b = 1)


  AUC <- sum(diff(one_minus_specificity) * rollmean(sensitivity, 2))
  AUC
}

(AUC_value <- receiver_operating_characteristic(class_prob_df))
```




```
## [1] 0.8502405
```

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

```
classification_output <- c(accuracy_value, cer, sensitivity_value, specificity_

names(classification_output) <- c("accuracy", "classification error rate", "se


t(t(classification_output))
```

```
##                                [,1]
## accuracy                  0.8066298
## classification error rate 0.1933702
## sensitivity               0.4736842
## specificity               0.9596774
## precision                 0.8437500
## 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.

```
(cMatrix <- confusionMatrix(class_vars$scored.class, class_vars$class, positiv
```

```
## 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
##
```

```
caret::sensitivity(as.factor(class_vars$scored.class), as.factor(class_vars$cl
```

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

```
caret::specificity(as.factor(class_vars$scored.class), as.factor(class_vars$cl
```

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

How do the results compare with your own functions?

They match when rounded to the 8th decimal place.

```
# get the needed metrics
caret_metrics <- c(cMatrix$overall[1],
                   1 -  as.numeric(cMatrix$overall[1]),
                   cMatrix$byClass[c(1, 2, 5, 7)])

t(t(round(classification_output, 8) == round(caret_metrics, 8)))
```

```
##                           [,1]
## accuracy                  TRUE
## classification error rate TRUE
## sensitivity               TRUE
## specificity               TRUE
## precision                 TRUE
## F1_score                  TRUE
```

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?

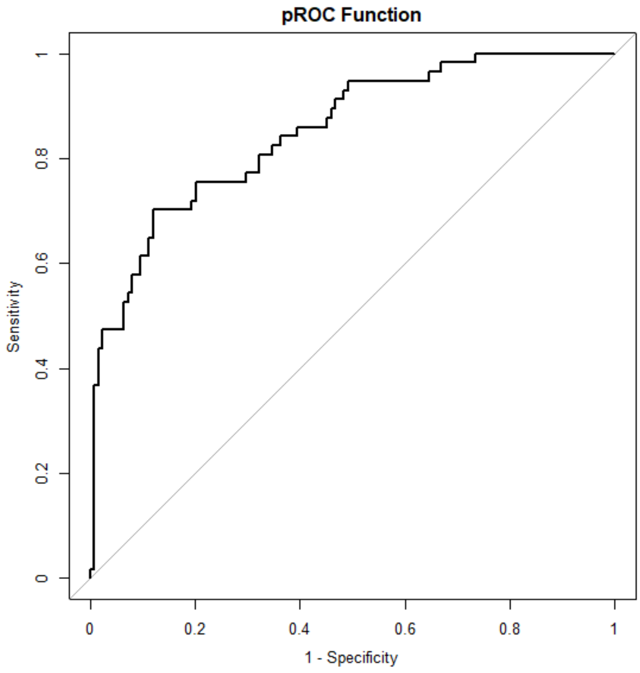
The ROC curve plots look very similar.

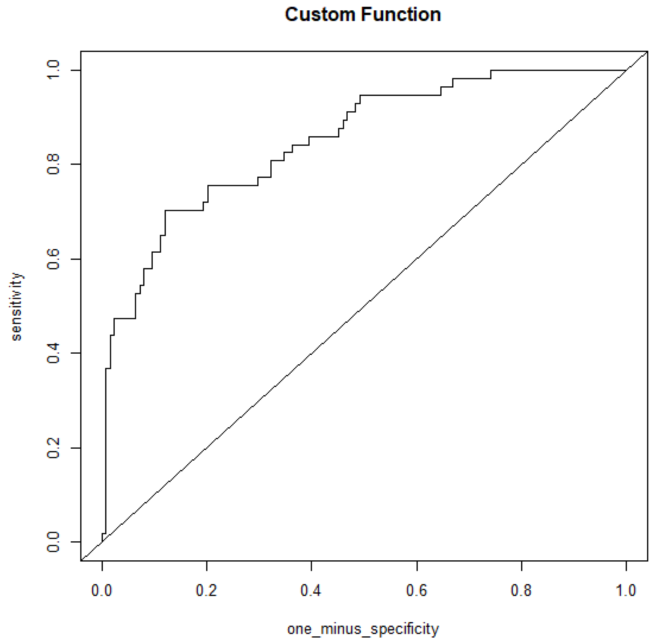
```
(curveROC <- roc(class_prob_df$class, class_prob_df$scored.probability))
```

```
##
## Call:
## roc.default(response = class_prob_df$class, predictor = class_prob_df$score
##
## Data: class_prob_df$scored.probability in 124 controls (class_prob_df$class
## Area under the curve: 0.8503
```

```
par(mfrow=c(2, 2))
plot(curveROC, legacy.axes = T, main = "pROC Function")
receiver_operating_characteristic(class_prob_df)
```

```
## [1] 0.8502405
```





The AUC values are similar but not the same. My AUC value is 7.074137e-05 less. They match when they are rounded to 3 decimal places.

```
as.numeric(curveROC$auc) - AUC_value
```

```
## [1] 7.074137e-05
```

```
round(curveROC$auc, 3) == round(AUC_value, 3)
```

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