Homework 02

CUNY MSDS DATA 621

Duubar Villalobos Jimenez mydvtech@gmail.com

October 14, 2018

Homework #2

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).

Steps

For reproducibility purposes, I have put the original data sets in my git-hub account and then I will read it as a data frame from that location.

```
git_user <- 'https://raw.githubusercontent.com/dvillalobos/'
git_dir <- 'MSDS/master/621/Homeworks/assignment-02/data/'
classification.df = read.csv(paste(git_user,
                                   git_dir,
                                   "classification-output-data.csv",
                                   sep = ""))
```

2.

The data set has three key columns we will use:

- **class:** the actual class for the observation
- **cored.class:** the predicted class for the observation (based on a threshold of 0.5)
- **cored.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?

Steps

table() uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels [1].

First, let's visualize the counts by creating individual reports.

- **Actual class for the observations**

```
table(classification.df$class, dnn = "Actual class for the observations")
```

```
## Actual class for the observations
##   0   1
## 124  57
```

- **Predicted class for the observations**

```
table(classification.df$scored.class, dnn = "Predicted class for the observations")
```

```
## Predicted class for the observations
##   0   1
## 149  32
```

- **Raw confusion matrix for this scored dataset**

```
table(classification.df$scored.class,
      classification.df$class,
      dnn = c("Predicted", "Target"))
```

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

A confusion matrix shows the number of correct and incorrect predictions made by the classification model compared to the actual outcomes (target value) in the data [2].

From the above table, we can describe as follows:

- **Columns** represent the true values **= Target**.
- **Rows** represent the predicted values **= Model**.

An easy way to corroborate the above assertion, is by performing as follows:

Column 1: Represents "0", if we add $119 + 5 = 124$ as seeing in the first summary report for the actual class for the observations.

Column 2: Represents "1", if we add $30 + 27 = 57$ as seeing in the first summary report for the actual class for the observations.

Also, from the above table, we can see the **predicted values in the rows** as follows:

Row 1: Represents "0", if we add $119 + 30 = 149$ as seeing in the second summary report for the predicted class for the observations.

Row 2: Represents "1", if we add $5 + 27 = 32$ as seeing in the second summary report for the predicted class for the observations.

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.

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

Steps

Below, you will find the R function that will take the data set as a data frame with actual and predicted classifications identified and will return the accuracy of the predictions.

```
getAccuracy <- function(df){

  confusion_matrix <- table(df$scored.class,
                            df$class,
                            dnn = c("Predicted", "Target"))

  a <- confusion_matrix[2,2]
  b <- confusion_matrix[2,1]
  c <- confusion_matrix[1,2]
  d <- confusion_matrix[1,1]

  Accuracy <-  (a+d)/(a+b+c+d)

  return(Accuracy)
}
```

Let's run the above function in order to obtain our model's accuracy.

```
getAccuracy(classification.df)
```

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

From the above results, we obtained that the accuracy is about 80.7% accurate.

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.

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

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

Steps

Below, you will find the R function that will take the data set as a data frame with actual and predicted classifications identified and will return the classification error rate of the predictions.

```
getClassificationErrorRate <- function(df){

  confusion_matrix <- table(df$scored.class,
                            df$class,
```

```
                            dnn = c("Predicted", "Target"))

  a <- confusion_matrix[2,2]
  b <- confusion_matrix[2,1]
  c <- confusion_matrix[1,2]
  d <- confusion_matrix[1,1]

  ClassificationErrorRate <-  (b+c)/(a+b+c+d)

  return(ClassificationErrorRate)
}
```

Let's run the above function in order to obtain our model's classification error rate.

```
getClassificationErrorRate(classification.df)
```

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

From the above results, we obtained that the classification error rate is about 19.3%.

Observation

Something interesting to note is that the sum of the two previous results equals 1, that is:

```
0.8066298 + 0.1933702
```

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

Or, just by running our previously build functions as well we can see that our final result equals to 1.

```
getAccuracy(classification.df) + getClassificationErrorRate(classification.df)
```

```
## [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.

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

Steps

Please note that the **Precision** is the same as the **Positive Predictive Value**.

Below, you will find the R function that will take the data set as a data frame with actual and predicted classifications identified and will return the precision of the predictions.

```
getPrecision <- function(df){

  confusion_matrix <- table(df$scored.class,
                            df$class,
                            dnn = c("Predicted", "Target"))

  a <- confusion_matrix[2,2]
  b <- confusion_matrix[2,1]
  c <- confusion_matrix[1,2]
```

```
  d <- confusion_matrix[1,1]

  Precision <-  a/(a+b)

  return(Precision)
}
```

Let's run the above function in order to obtain our model's Positive Predictive Value.

```
getPrecision(classification.df)
```

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

From the above results, we obtained that the Positive Predictive Value is about 84.4%.

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.

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

Steps

Below, you will find the R function that will take the data set as a data frame with actual and predicted classifications identified and will return the sensitivity of the predictions.

```
getSensitivity <- function(df){

  confusion_matrix <- table(df$scored.class,
                            df$class,
                            dnn = c("Predicted", "Target"))

  a <- confusion_matrix[2,2]
  b <- confusion_matrix[2,1]
  c <- confusion_matrix[1,2]
  d <- confusion_matrix[1,1]

  Sensitivity <-  a/(a+c)

  return(Sensitivity)
}
```

Let's run the above function in order to obtain our model's Sensitivity.

```
getSensitivity(classification.df)
```

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

From the above results, we obtained that the Sensitivity is about 47.4%.

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.**

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

Steps

Below, you will find the R function that will take the data set as a data frame with actual and predicted classifications identified and will return the Specificity of the predictions.

```
getSpecificity <- function(df){

  confusion_matrix <- table(df$scored.class,
                            df$class,
                            dnn = c("Predicted", "Target"))

  a <- confusion_matrix[2,2]
  b <- confusion_matrix[2,1]
  c <- confusion_matrix[1,2]
  d <- confusion_matrix[1,1]

  Specificity <-  d/(b+d)

  return(Specificity)
}
```

Let's run the above function in order to obtain our model's Specificity.

```
getSpecificity(classification.df)
```

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

From the above results, we obtained that the Specificity is about 95.9%.

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.

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

Steps

Below, you will find the R function that will take the data set as a data frame with actual and predicted classifications identified and will return the F1 score of the predictions.

```
getF1Score <- function(df){

  confusion_matrix <- table(df$scored.class,
                            df$class,
                            dnn = c("Predicted", "Target"))

  a <- confusion_matrix[2,2]
  b <- confusion_matrix[2,1]
  c <- confusion_matrix[1,2]
  d <- confusion_matrix[1,1]
```

```
  Precision <-  a/(a+b)
  Sensitivity <-  a/(a+c)

  F1Score <- (2 * Precision * Sensitivity)/(Precision + Sensitivity)

  return(F1Score)
}
```

Let's run the above function in order to obtain our model's F1 score.

```
getF1Score(classification.df)
```

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

From the above results, we obtained that the F1 score is about 60.7%.

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 $ab < a$.)

Steps

In order to show that the bounds on the $F1$ score are always between 0 and 1, I will proceed to solve it as follows:

For simplicity reasons, let's define as follows:

P = Precision

S = Sensitivity

Hence, we can rewrite our given formula as follows:

$$F1\ Score = \frac{2 \times P \times S}{P + S}$$

From previous results, it has been demonstrated that $0 \leq P \leq 1$ and $0 \leq S \leq 1$; hence we could conclude that $PS \leq S$ and $PS \leq P$, this implies that $0 \leq PS \leq P \leq 1$ and $0 \leq PS \leq S \leq 1$; from this result and given that we have a numerator in the $[0, 1]$, the resulting division by any number will result in a value pertaining to the $[0, 1]$ range.

Another way of performing these calculations, will be by employing calculus as follows:

First, let's rearrange our equation as follows:

$$F1\ Score = \frac{2 \times P \times S}{P + S}$$

We could rewrite it as follows:

$$F1\ Score = \frac{\frac{2}{1}}{\frac{P+S}{PS}}$$

From here, we could rewrite our equation as follows:

$$F1\ Score = \frac{\frac{2}{1}}{\frac{P}{PS} + \frac{S}{PS}}$$

And by simplifying, we obtain:

$$F1\ Score = \frac{2}{\frac{1}{S} + \frac{1}{P}}$$

In order to determine the values, we could calculate as follows:

$$a)\ lim_{P->0}\left(lim_{S->0}\left(\frac{2}{\frac{1}{S} + \frac{1}{P}}\right)\right)$$

$$b)\ lim_{P->0}\left(lim_{S->1}\left(\frac{2}{\frac{1}{S} + \frac{1}{P}}\right)\right)$$

$$c)\ lim_{P->1}\left(lim_{S->0}\left(\frac{2}{\frac{1}{S} + \frac{1}{P}}\right)\right)$$

$$d)\ lim_{P->1}\left(lim_{S->1}\left(\frac{2}{\frac{1}{S} + \frac{1}{P}}\right)\right)$$

By solving the above limits, we conclude as follows:

$$a)\ \frac{2}{\infty + \infty} = 0$$

$$b)\ \frac{2}{1 + \infty} = 0$$

$$c)\ \frac{2}{\infty + 1} = 0$$

$$d)\ \frac{2}{1 + 1} = 1$$

From the above results, is demonstrated that the $F1$ Score will always be between 0 and 1.

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.

Steps

Let's group and count how many classes are in each 0.01 threshold intervals calculated by rounding to 2 decimals the scored.probability column, if a threshold interval is missing from the list, we will assume that the number of classes for both 0 and 1 in that missing interval is non existent.

```
reduced.df <- classification.df[c('class','scored.probability')]
reduced.df$scored.probability <- round(reduced.df$scored.probability,2)

table <- data.frame(table(reduced.df$class, reduced.df$scored.probability))
table$Var1 <- as.numeric(levels(table$Var1))
```

Let's visualize our given table in the defined thresholds:

```
names(table) <- c('class', 'threshold', 'counts')
table
```

```
##     class threshold counts
## 1       0      0.02      1
## 2       1      0.02      0
## 3       0      0.03      3
## 4       1      0.03      0
## 5       0      0.05      5
## 6       1      0.05      0
## 7       0      0.06      6
## 8       1      0.06      0
## 9       0      0.07      4
## 10      1      0.07      0
## 11      0      0.08      6
## 12      1      0.08      0
## 13      0      0.09      4
## 14      1      0.09      0
## 15      0       0.1      9
## 16      1       0.1      1
## 17      0      0.11      4
## 18      1      0.11      1
## 19      0      0.12      3
## 20      1      0.12      1
## 21      0      0.13      6
## 22      1      0.13      0
## 23      0      0.14      7
## 24      1      0.14      0
## 25      0      0.15      4
## 26      1      0.15      0
## 27      0      0.16      2
## 28      1      0.16      1
## 29      0      0.17      3
## 30      1      0.17      3
## 31      0      0.18      3
## 32      1      0.18      1
## 33      0      0.19      2
## 34      1      0.19      0
## 35      0       0.2      2
## 36      1       0.2      0
## 37      0      0.21      3
## 38      1      0.21      1
```

```
## 39      0      0.22      2
## 40      1      0.22      0
## 41      0      0.23      0
## 42      1      0.23      1
## 43      0      0.24      2
## 44      1      0.24      0
## 45      0      0.25      1
## 46      1      0.25      1
## 47      0      0.26      2
## 48      1      0.26      1
## 49      0      0.27      3
## 50      1      0.27      2
## 51      0      0.28      1
## 52      1      0.28      0
## 53      0      0.29      2
## 54      1      0.29      0
## 55      0       0.3      6
## 56      1       0.3      0
## 57      0      0.31      1
## 58      1      0.31      0
## 59      0      0.32      2
## 60      1      0.32      1
## 61      0      0.33      1
## 62      1      0.33      1
## 63      0      0.34      0
## 64      1      0.34      1
## 65      0      0.35      2
## 66      1      0.35      0
## 67      0      0.36      3
## 68      1      0.36      0
## 69      0      0.37      4
## 70      1      0.37      0
## 71      0      0.38      0
## 72      1      0.38      2
## 73      0       0.4      1
## 74      1       0.4      1
## 75      0      0.41      1
## 76      1      0.41      2
## 77      0      0.42      1
## 78      1      0.42      1
## 79      0      0.43      0
## 80      1      0.43      1
## 81      0      0.44      1
## 82      1      0.44      0
## 83      0      0.45      1
## 84      1      0.45      1
## 85      0      0.46      2
## 86      1      0.46      2
## 87      0      0.47      0
## 88      1      0.47      1
## 89      0      0.48      0
## 90      1      0.48      1
## 91      0      0.49      2
## 92      1      0.49      1
```

```
## 93      0       0.5      1
## 94      1       0.5      0
## 95      0      0.52      2
## 96      1      0.52      0
## 97      0      0.53      0
## 98      1      0.53      1
## 99      0      0.55      0
## 100     1      0.55      1
## 101     0      0.56      1
## 102     1      0.56      0
## 103     0      0.59      0
## 104     1      0.59      1
## 105     0      0.62      0
## 106     1      0.62      3
## 107     0      0.63      1
## 108     1      0.63      0
## 109     0      0.64      0
## 110     1      0.64      2
## 111     0      0.66      0
## 112     1      0.66      1
## 113     0      0.68      0
## 114     1      0.68      2
## 115     0      0.69      0
## 116     1      0.69      1
## 117     0       0.7      0
## 118     1       0.7      1
## 119     0      0.71      0
## 120     1      0.71      1
## 121     0      0.72      0
## 122     1      0.72      2
## 123     0      0.76      0
## 124     1      0.76      1
## 125     0      0.78      0
## 126     1      0.78      1
## 127     0      0.81      0
## 128     1      0.81      1
## 129     0      0.83      0
## 130     1      0.83      1
## 131     0      0.85      0
## 132     1      0.85      2
## 133     0      0.86      0
## 134     1      0.86      1
## 135     0      0.88      0
## 136     1      0.88      2
## 137     0      0.89      1
## 138     1      0.89      1
## 139     0      0.95      0
## 140     1      0.95      1
```

Let's reshape this table in order to have respective group counts by column for 0 and for 1.

```
table <- reshape(table, timevar = "class", idvar = "threshold", direction = 'wide')
table
```

```
##     threshold counts.0 counts.1
```

```
## 1       0.02       1       0
## 3       0.03       3       0
## 5       0.05       5       0
## 7       0.06       6       0
## 9       0.07       4       0
## 11      0.08       6       0
## 13      0.09       4       0
## 15       0.1       9       1
## 17      0.11       4       1
## 19      0.12       3       1
## 21      0.13       6       0
## 23      0.14       7       0
## 25      0.15       4       0
## 27      0.16       2       1
## 29      0.17       3       3
## 31      0.18       3       1
## 33      0.19       2       0
## 35       0.2       2       0
## 37      0.21       3       1
## 39      0.22       2       0
## 41      0.23       0       1
## 43      0.24       2       0
## 45      0.25       1       1
## 47      0.26       2       1
## 49      0.27       3       2
## 51      0.28       1       0
## 53      0.29       2       0
## 55       0.3       6       0
## 57      0.31       1       0
## 59      0.32       2       1
## 61      0.33       1       1
## 63      0.34       0       1
## 65      0.35       2       0
## 67      0.36       3       0
## 69      0.37       4       0
## 71      0.38       0       2
## 73       0.4       1       1
## 75      0.41       1       2
## 77      0.42       1       1
## 79      0.43       0       1
## 81      0.44       1       0
## 83      0.45       1       1
## 85      0.46       2       2
## 87      0.47       0       1
## 89      0.48       0       1
## 91      0.49       2       1
## 93       0.5       1       0
## 95      0.52       2       0
## 97      0.53       0       1
## 99      0.55       0       1
## 101     0.56       1       0
## 103     0.59       0       1
## 105     0.62       0       3
## 107     0.63       1       0
```

```
## 109      0.64        0        2
## 111      0.66        0        1
## 113      0.68        0        2
## 115      0.69        0        1
## 117       0.7        0        1
## 119      0.71        0        1
## 121      0.72        0        2
## 123      0.76        0        1
## 125      0.78        0        1
## 127      0.81        0        1
## 129      0.83        0        1
## 131      0.85        0        2
## 133      0.86        0        1
## 135      0.88        0        2
## 137      0.89        1        1
## 139      0.95        0        1
```

Let's do some calculations for specificity and sensitivity.

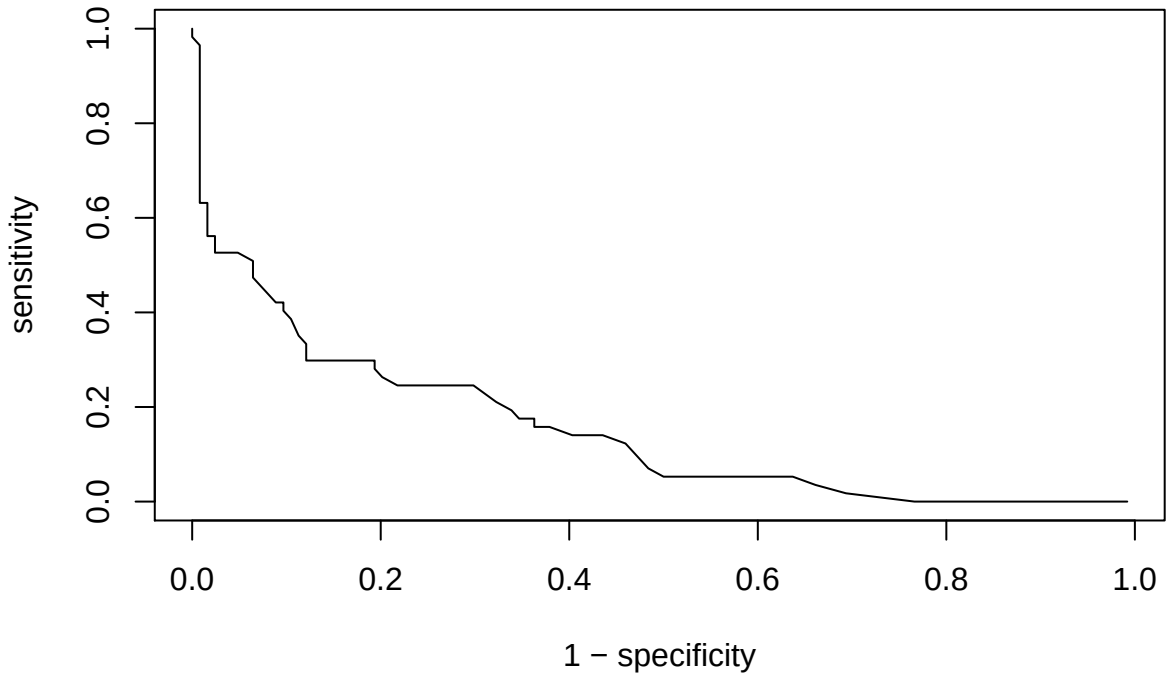
```
table$specificity <- cumsum(table$counts.0)/sum(table$counts.0)
table$sensitivity <- cumsum(table$counts.1)/sum(table$counts.1)
table
```

```
##     threshold counts.0 counts.1 specificity sensitivity
## 1        0.02        1        0 0.008064516  0.00000000
## 3        0.03        3        0 0.032258065  0.00000000
## 5        0.05        5        0 0.072580645  0.00000000
## 7        0.06        6        0 0.120967742  0.00000000
## 9        0.07        4        0 0.153225806  0.00000000
## 11       0.08        6        0 0.201612903  0.00000000
## 13       0.09        4        0 0.233870968  0.00000000
## 15        0.1        9        1 0.306451613  0.01754386
## 17       0.11        4        1 0.338709677  0.03508772
## 19       0.12        3        1 0.362903226  0.05263158
## 21       0.13        6        0 0.411290323  0.05263158
## 23       0.14        7        0 0.467741935  0.05263158
## 25       0.15        4        0 0.500000000  0.05263158
## 27       0.16        2        1 0.516129032  0.07017544
## 29       0.17        3        3 0.540322581  0.12280702
## 31       0.18        3        1 0.564516129  0.14035088
## 33       0.19        2        0 0.580645161  0.14035088
## 35        0.2        2        0 0.596774194  0.14035088
## 37       0.21        3        1 0.620967742  0.15789474
## 39       0.22        2        0 0.637096774  0.15789474
## 41       0.23        0        1 0.637096774  0.17543860
## 43       0.24        2        0 0.653225806  0.17543860
## 45       0.25        1        1 0.661290323  0.19298246
## 47       0.26        2        1 0.677419355  0.21052632
## 49       0.27        3        2 0.701612903  0.24561404
## 51       0.28        1        0 0.709677419  0.24561404
## 53       0.29        2        0 0.725806452  0.24561404
## 55        0.3        6        0 0.774193548  0.24561404
## 57       0.31        1        0 0.782258065  0.24561404
## 59       0.32        2        1 0.798387097  0.26315789
## 61       0.33        1        1 0.806451613  0.28070175
```

```
## 63       0.34        0        1 0.806451613  0.29824561
## 65       0.35        2        0 0.822580645  0.29824561
## 67       0.36        3        0 0.846774194  0.29824561
## 69       0.37        4        0 0.879032258  0.29824561
## 71       0.38        0        2 0.879032258  0.33333333
## 73        0.4        1        1 0.887096774  0.35087719
## 75       0.41        1        2 0.895161290  0.38596491
## 77       0.42        1        1 0.903225806  0.40350877
## 79       0.43        0        1 0.903225806  0.42105263
## 81       0.44        1        0 0.911290323  0.42105263
## 83       0.45        1        1 0.919354839  0.43859649
## 85       0.46        2        2 0.935483871  0.47368421
## 87       0.47        0        1 0.935483871  0.49122807
## 89       0.48        0        1 0.935483871  0.50877193
## 91       0.49        2        1 0.951612903  0.52631579
## 93        0.5        1        0 0.959677419  0.52631579
## 95       0.52        2        0 0.975806452  0.52631579
## 97       0.53        0        1 0.975806452  0.54385965
## 99       0.55        0        1 0.975806452  0.56140351
## 101      0.56        1        0 0.983870968  0.56140351
## 103      0.59        0        1 0.983870968  0.57894737
## 105      0.62        0        3 0.983870968  0.63157895
## 107      0.63        1        0 0.991935484  0.63157895
## 109      0.64        0        2 0.991935484  0.66666667
## 111      0.66        0        1 0.991935484  0.68421053
## 113      0.68        0        2 0.991935484  0.71929825
## 115      0.69        0        1 0.991935484  0.73684211
## 117       0.7        0        1 0.991935484  0.75438596
## 119      0.71        0        1 0.991935484  0.77192982
## 121      0.72        0        2 0.991935484  0.80701754
## 123      0.76        0        1 0.991935484  0.82456140
## 125      0.78        0        1 0.991935484  0.84210526
## 127      0.81        0        1 0.991935484  0.85964912
## 129      0.83        0        1 0.991935484  0.87719298
## 131      0.85        0        2 0.991935484  0.91228070
## 133      0.86        0        1 0.991935484  0.92982456
## 135      0.88        0        2 0.991935484  0.96491228
## 137      0.89        1        1 1.000000000  0.98245614
## 139      0.95        0        1 1.000000000  1.00000000
```

Let's plot our Receiver Operating Characteristic (ROC) Curve for our true classification model.

```
plot(1 - table$specificity, table$sensitivity, type = 'l',
     xlab = '1 - specificity',
     ylab = 'sensitivity')
```



Let's create a function that return the Area Under the Curve (AUC). For this process, I will employ Riemann approximation approach in order to calculate the area [3].

```
table$x <- 0
table$y <- 0
table$auc <- 0
table$`1 - specificity` <- 1 - table$specificity
for(i in 1:(dim(table)[1]-1)){
  table$x[i] <- abs(table$`1 - specificity`[i+1] - table$`1 - specificity`[i])
  table$y[i] <- abs(table$sensitivity[i+1] - table$sensitivity[i])
  table$auc[i] <- table$x[i] * (table$sensitivity[i] + table$sensitivity[i+1])/2
  #table$auc[i] <- table$x[i] * table$sensitivity[i] + (table$x[i] * table$y[i])/2 # Alternate
}
```

Let's see what's the AUC result.

```
sum(table$auc)
```

```
## [1] 0.1494765
```

As you can see our calculated AUC is about 0.1495.

11.

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

Steps

All have been answered on a one by one basis; please review the previous answers.

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?

Steps

Let's compare our previous results with the use of the function **confusionMatrix()** from the **caret** library.

```
cMatrix <- confusionMatrix(data = as.factor(classification.df$scored.class),
                           reference = as.factor(classification.df$class),
                           positive = '1')
cMatrix
```

```
## 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
##
```

In particular, let's expand the class:

```
data.frame(cMatrix$byClass)
```

```
##                      cMatrix.byClass
## Sensitivity                0.4736842
## Specificity                0.9596774
## Pos Pred Value             0.8437500
## Neg Pred Value             0.7986577
## Precision                  0.8437500
## Recall                     0.4736842
## F1                         0.6067416
## Prevalence                 0.3149171
## Detection Rate             0.1491713
```

```
## Detection Prevalence       0.1767956
## Balanced Accuracy          0.7166808
```

From the above function, we can notice that all our results and setups match accordingly with our individual results from all previous questions; that is:

- Confusion matrix generated with the **table()** function looks the same .
- The accuracy match.
- The classification error also match since the accuracy match.
- The Pos Pred Value also known as precision also match.
- The sensitivity also match.
- The specificity also match.
- The F1 Score also match.

Now, let's compare sensitivity and specificity.

```
sensitivity(data = as.factor(classification.df$scored.class),
            reference = as.factor(classification.df$class),
            positive = '1')
```

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

As we can see this sensitivity result also match our previous results.

```
specificity(data = as.factor(classification.df$scored.class),
            reference = as.factor(classification.df$class),
            negative = '0')
```

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

Also, if we compare the values, we notice that the results match accordingly.

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?

Steps

First, let's prepare our function

```
rocCurve <- roc(classification.df$class, classification.df$scored.probability)
```

Let's see the area under the curve.

```
auc(rocCurve)
```

```
## Area under the curve: 0.8503
```

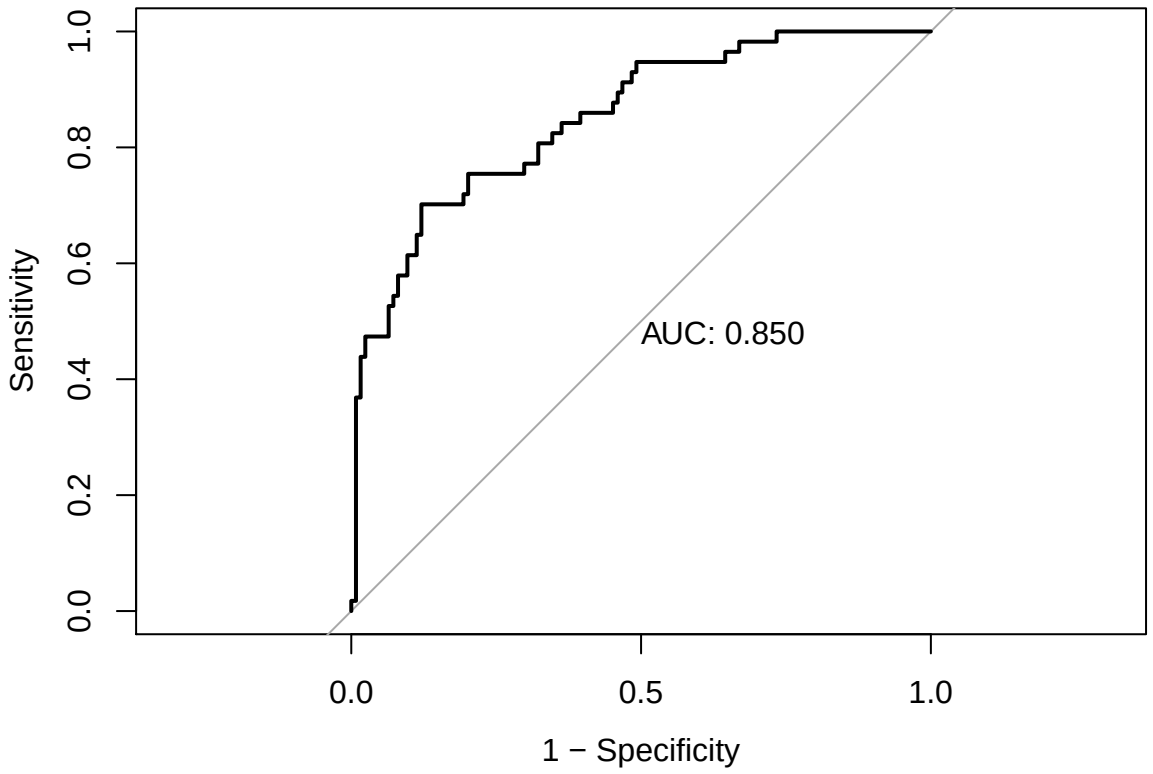
Let's see the confidence interval.

```
ci(rocCurve)
```

```
## 95% CI: 0.7905-0.9101 (DeLong)
```

Let's plot our RCO curve.

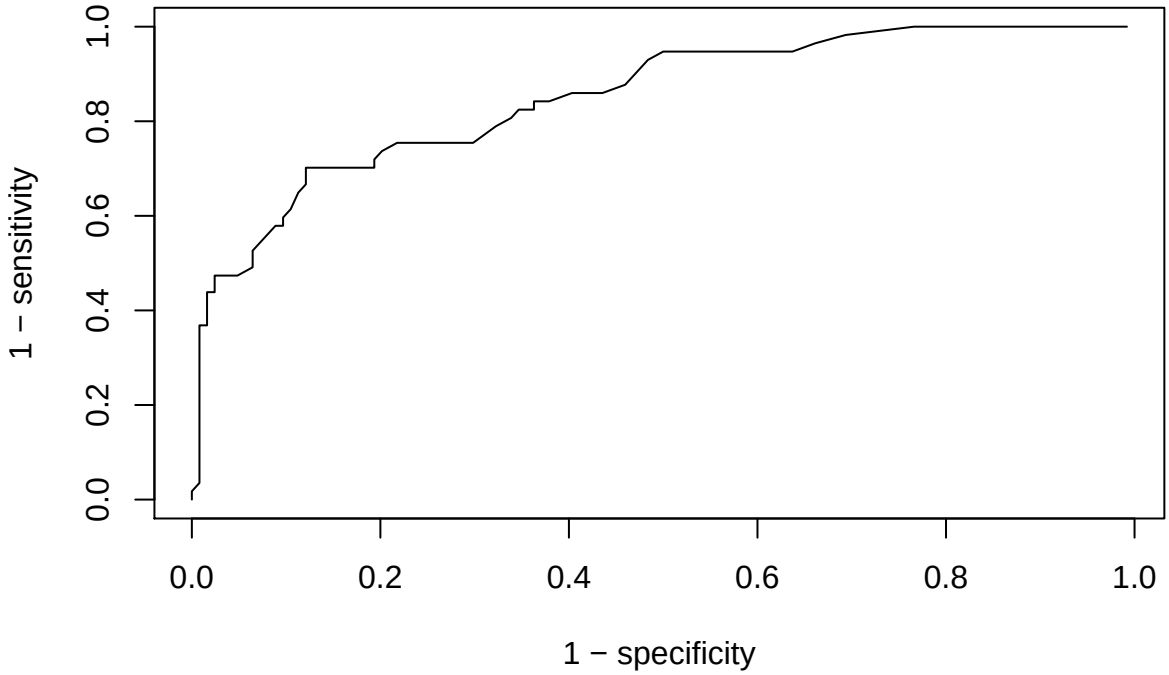
```
plot(rocCurve, print.auc=TRUE, legacy.axes = TRUE)
```



Notes

Something interesting to note from above, is that our manually provided ROC curve and graph are not aligned with our pROC plot; after some research, I have noticed that the **sensitivity** values generated by the ROC function seem to be **1 - sensitivity** values generated from the manual form; thus making both graphs and areas not matching; however; below is the modification for the manually created chart by adjusting the sensitivity value for **1 - sensitivity**.

```
plot(1 - table$specificity, 1 - table$sensitivity, type = 'l',
     xlab = '1 - specificity',
     ylab = '1 - sensitivity')
```

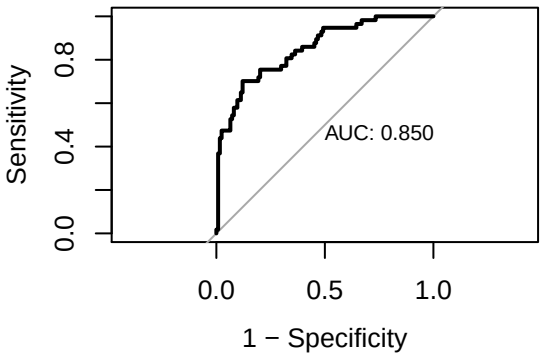


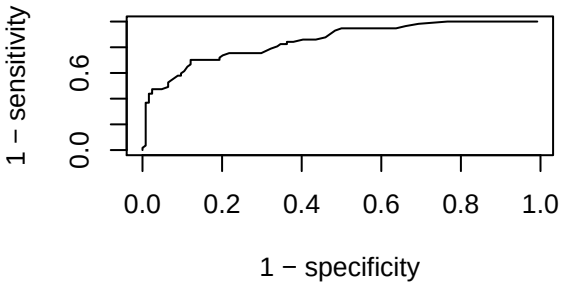
Thus producing the following AUC result that matches up to three decimals with our automatically generated values from the pROC function.

```
1 - sum(table$auc)
```

```
## [1] 0.8505235
```

```
par(mfrow=c(2,2))
plot(rocCurve, print.auc=TRUE, legacy.axes = TRUE)
plot(1 - table$specificity, 1 - table$sensitivity, type = 'l',
     xlab = '1 - specificity',
     ylab = '1 - sensitivity')
```





References

[1] https://www.rdocumentation.org/packages/base/versions/3.5.1/topics/table

[2] http://www.saedsayad.com/model_evaluation_c.htm

[3] https://en.wikipedia.org/wiki/Riemann_sum