Data 621: Assignment 2

Group 5: Christina Valore, Henry Vasquez, Chunhui Zhu, Chunmei Zhu, Yuen Chun Wong

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.

RMD file can be accessed here: https://github.com/ChristinaValore/Business-Analytics-and-Data-Mining-621/blob/master/Homework2/Data621-Homework2.Rmd

Q1.

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

We uploaded the classification dataset to a GitHub repo: https://raw.githubusercontent.com/hvasquez81/DATA621/master/classification-output-data.csv to be easily accessed by our entire team.

Here is the head() of the data using kable styling:

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

Q2.

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?

After generating the confusion matrix using the table() function, we printed the output in kable styling below. The rows are predicted values and the columns are actual values.

	0	1
0	119	30
1	5	27

Q3.

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

$$Accuracy = TP + TN\ TP + FP + TN + FN$$

Our function takes in the data, predicted column and actual column and then add them to the confusion matrix. We then return the accuracy equation. Our results show that our classifier has an accuracy of .8066 or 80.66%.

```
```{r accuracy}
accuracy = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #Accurary = (TP + TN) / (TP + FP +TN +FN)
  return(round((TP+TN)/(TP + FP + TN + FN), 4))
}
print(paste0("Accuracy: ", accuracy(data, 'scored.class', 'class')))
```
```

```
[1] "Accuracy: 0.8066"
```

Q4.

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.

$$Classification\ Error\ Rate = FP + FN\ TP + FP + TN + FN$$

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

We use the same procedure as above; except this time, we return an error rate which is .1934 or 19.34%. We also check to that the accuracy and error rate sum to 1.

```{r errorRate}
errorRate = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #Classification Error Rate = ( FP + FN )/(TP + FP +TN +FN)
  return(round((FP+FN)/(TP + FP + TN + FN), 4))
}

print(paste0("Error rate: ", errorRate(data, 'scored.class', 'class')))

#accuracy + error rate
print(paste0("Accuracy + Error rate = ", accuracy(data, 'scored.class', 'class'), " + ", errorRate(data, 'scored.class', 'class'), " = ",
(accuracy(data, 'scored.class', 'class') + errorRate(data, 'scored.class', 'class'))))
```

```
[1] "Error rate: 0.1934"
[1] "Accuracy + Error rate = 0.8066 + 0.1934 = 1"
```

Q5.

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

$$Precision = TP\ TP + FP$$

We use the same procedure as above; except this time, we return the precision, which is .8438 or 84.38%.

```{r precision}
precision = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #Precision = TP / (TP + FP)
  return(round((TP)/(TP + FP), 4))
}

print(paste0("Precision: ", precision(data, 'scored.class', 'class')))
```

```
[1] "Precision: 0.8438"
```

Q6.

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.

$$Sensitivity = TP\ TP + FN$$

We use the same procedure as above; except this time, we return the sensitivity, which is .4737 or 47.37%.

```
```{r sensitivity}
sensitivity = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #Sensitivity = TP / (TP + FN)
  return(round((TP)/(TP + FN), 4))
}

print(paste0("Sensitivity: ", sensitivity(data, 'scored.class', 'class')))
```
```

```
[1] "Sensitivity: 0.4737"
```

Q7.

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

$$Specificity = TN\ TN + FP$$

We use the same procedure as above; except this time, we return the specificty, which is .9597 or 95.97%.

```r specificity
specificity = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #Specificity = TN / (TN+FP)
  return(round((TN)/(TN + FP), 4))
}

print(paste0("Specificity: ", specificity(data, 'scored.class', 'class')))
```

```
[1] "Specificity: 0.9597"
```

Q8.

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.

$$F1\ Score = 2 \times Precision \times Sensitivity\ Precision + Sensitivity$$

We use the same procedure as above; except this time, we compute the precision and sensitivity and then return the F1 score using those values. Our F1 score is .6068.

```r f1 score
f1_score = function(data, predicted_col_name, actual_col_name) {

  Precision = precision(data, predicted_col_name, actual_col_name)
  Sensitivity = sensitivity(data, predicted_col_name, actual_col_name)

  #F1 Score = 2 * Precision * Sensitivity / (Precision + Sensitivity)
  return(round((2*Precision*Sensitivity)/(Precision + Sensitivity), 4))

}

print(paste0("F1 score: ", f1_score(data, 'scored.class', 'class')))
```

```
[1] "F1 score: 0.6068"
```

Q9.

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 $ab < a$.)

Using the hint above, we say a = precision and b = sensitivity and both numbers are between 0 and 1. So if a = .5 and b =.5 then a*b = .25 and so a*b < a or a*b < b. To prove that the F1 score is always between 0 and 1, we can run a simulation using the idea above with the p and s values also between 0 and 1.

By generating random numbers and plugging them into the F1 score equation, we see that as we increase the amount of random numbers generated from 10 to 100 to 1000, we see that the max value gets close to 1 as the min value gets close 0 but never reaches either value. If we continue to generate more random numbers, we will see the max value continues to rise closer to 1, while the min closer to 0, however the max/min will never be equal to 1/0. Thus, the F1 value will always be between 0 and 1.

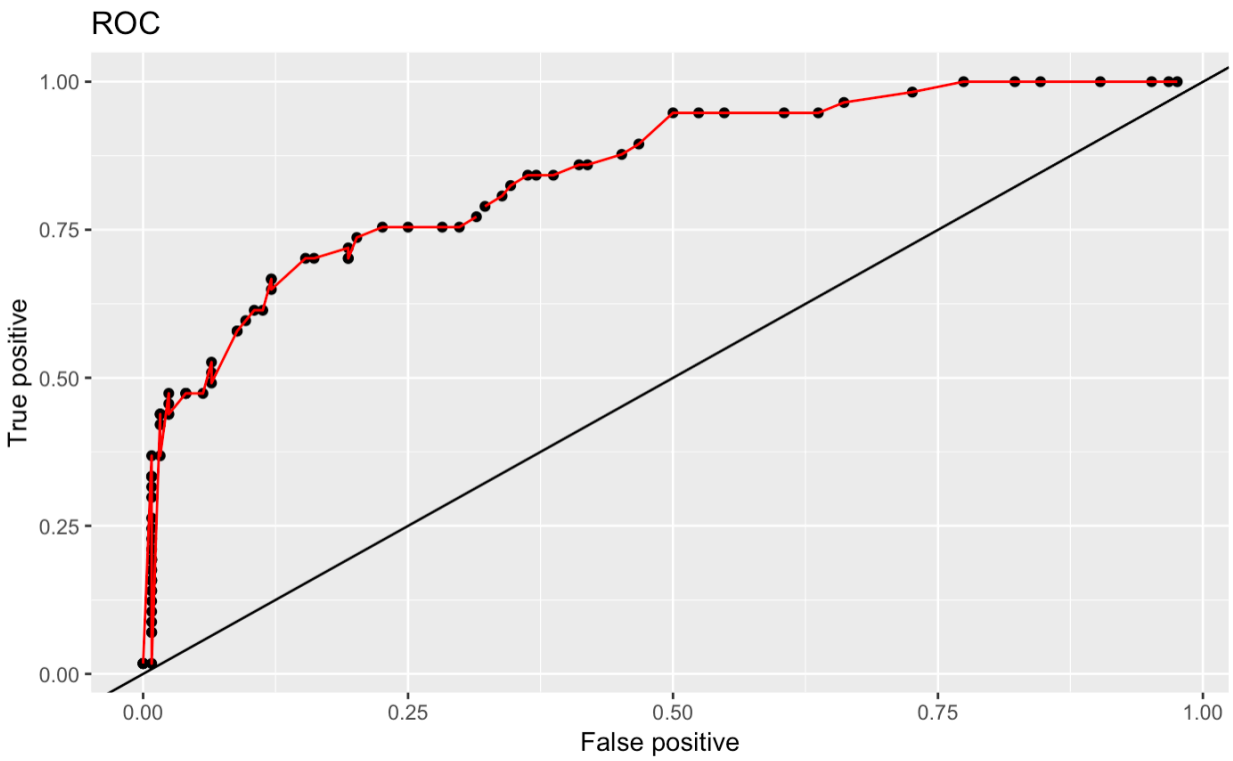
See the summary output below for random numbers generated at 10, 100 and 1000, taking note of min and max values approaching 0 and 1:

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
0.04301 0.21419 0.47869 0.46052 0.67632 0.88619 
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
0.004924 0.213325 0.346562 0.404481 0.578541 0.960901 
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max. 
0.0005588 0.1889674 0.3670928 0.3981011 0.5990581 0.9950490 
```

Q10.

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.

We created a function that takes in the data and created the AUC segments using the scored probability and class variables. Then we took those segments and added them into two vectors, and we added both of those vectors to a data frame.



Q11.

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

Using our created functions, this was our total classification output:

```
[1] "Accuracy: 0.8066"
[1] "Error rate: 0.1934"
[1] "Accuracy + Error rate = 0.8066 + 0.1934 = 1"
[1] "Precision: 0.8438"
[1] "Sensitivity: 0.4737"
[1] "Specificity: 0.9597"
[1] "F1 score: 0.6068"
```

The accuracy level is pretty high indicating a good model; however, sensitivity is also a bit high. The sensitivity could indicate that the model is easily manipulated with small changes.

Q12.

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?

Comparing the results to the functions produced previously, all results are the same. Here is the output from the caret package:

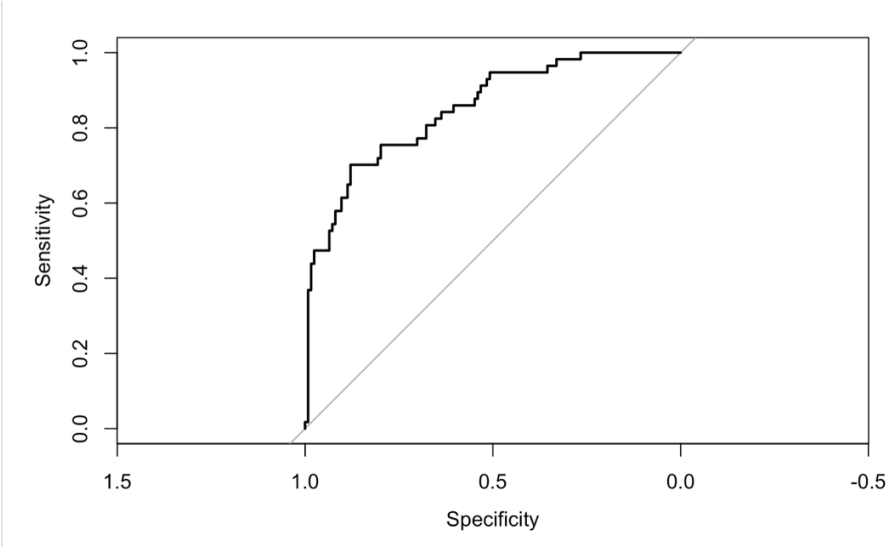
```
Confusion Matrix and Statistics

          Actual
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

[1] 0.4736842
[1] 0.9596774
```

Q13.

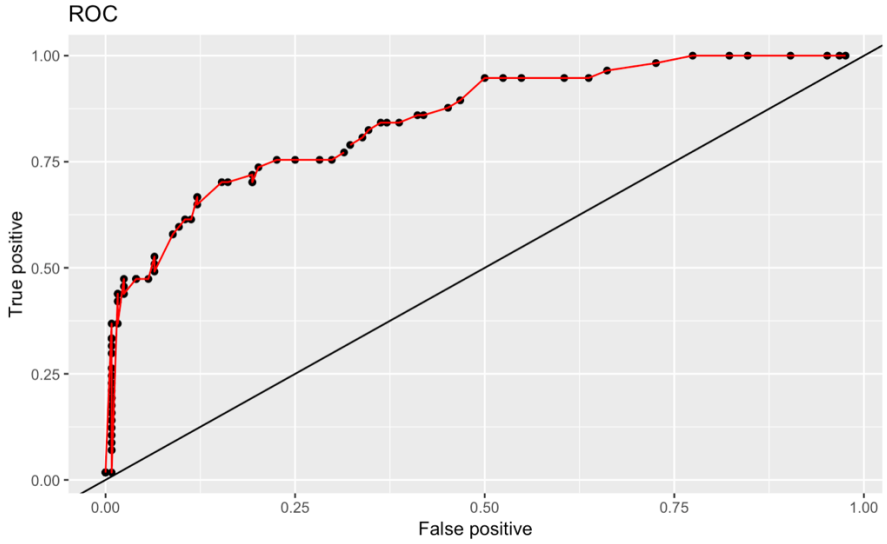
Investigate the pROC package. Use it to generate an ROC curve for the data set. How do the results compare with your own functions?

The curve shape is very similar from our generated graph. The pROC curve (on top) looks to be more steep then our curve, we may have to adjust our calculations in the function to account for this.

pROC package graph:



ROC function curve:



RMD file can be accessed here: https://github.com/ChristinaValore/Business-Analytics-and-Data-Mining-621/blob/master/Homework2/Data621-Homework2.Rmd