DATA621_Home_Work_2

Dilip Ganesan

6/22/2018

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

```
classdata = read.csv('classification-output-data.csv')
```

Loaded the classification output. It has 181 observations and 11 variables. Out of 11 variables 3 are output variables from the classifier.

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?

```
#table(classdata$scored.class,classdata$class)

knitr::kable(table(classdata$scored.class,classdata$class))
```

	0	1
0	119	30
1	5	27

The Rows are Actual and The Columns are predicted.Per the defenition Confusion Matrix the row values are Actual and Column values are Predicted.

3 - 8 and 11.

For the questions 3 - 8 and 11, we are going to write a single function, which will take input as a dataframe and it will print all the Classification Metrics.

1. Accuracy
2. Classification Error Rate
3. Precision
4. Senstivity or Recall.
5. Specificity
6. F1 Score.

For the calculation of above metrics, we need to use the Confusion Matrix, that we created using table() function in question 2.

```
conmat = table(classdata$scored.class,classdata$class)
```

```
ClassificationMetrics = function(conmat){
  truepositive = conmat[1,1]
  falsepositive = conmat[1,2]
  falsenegative = conmat[2,1]
  truenegative = conmat[2,2]
  totalprediction = truepositive + falsenegative + falsepositive + truenegative

  accuracy = (truepositive + truenegative) / (totalprediction)
  classerrorrate = (falsepositive + falsenegative) / (totalprediction)
  precision = truepositive / (truepositive + falsepositive)
  recall = truepositive / (truepositive + falsenegative)
  specificity = truenegative / (truenegative + falsepositive)
  f1 = (2 * precision * recall) / (precision + recall)

  dataframe = data.frame(Accurarcy = accuracy, ClassErrorRate = classerrorrate, Precision = precision, S
  return(dataframe)

}

metoutput = ClassificationMetrics(conmat)

knitr::kable(metoutput)
```

Accurarcy	ClassErrorRate	Precision	Senstivity	Specificity	F1
0.8066298	0.1933702	0.7986577	0.9596774	0.4736842	0.8717949

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

From the above output we have to see whether the summation of Accuracy and Classification Error Rate equal to 1.

```
sum = metoutput$Accurarcy + metoutput$ClassErrorRate
#sum
```

From the above we can see the summation of Accuracy and ClassErrorRate is 1.

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.

F1-score is the harmonic mean between precision and recall. Since Precision and Recall values are bounded by 0 and 1, the F1 Score will always 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.

```
newROCandAUC = function(label, scores){
    label = label[order(scores, decreasing=TRUE)]
    Sensitivity=cumsum(label)/sum(label)
    Specificity=cumsum(!label)/sum(!label)
    df = data.frame(Sensitivity,
                    Specificity,
                    label)

    dSpecificity=c(diff(Specificity), 0)
    dSensitivity=c(diff(Sensitivity), 0)
    AUC=sum(Sensitivity * dSpecificity) + sum(dSensitivity * dSpecificity) / 2

    output=list(df, AUC)
    return(output)
}

ROCandAUCOutput = newROCandAUC(classdata$class, classdata$scored.probability)
rocResults = ROCandAUCOutput[[1]]
AUC = ROCandAUCOutput[[2]]
```

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?

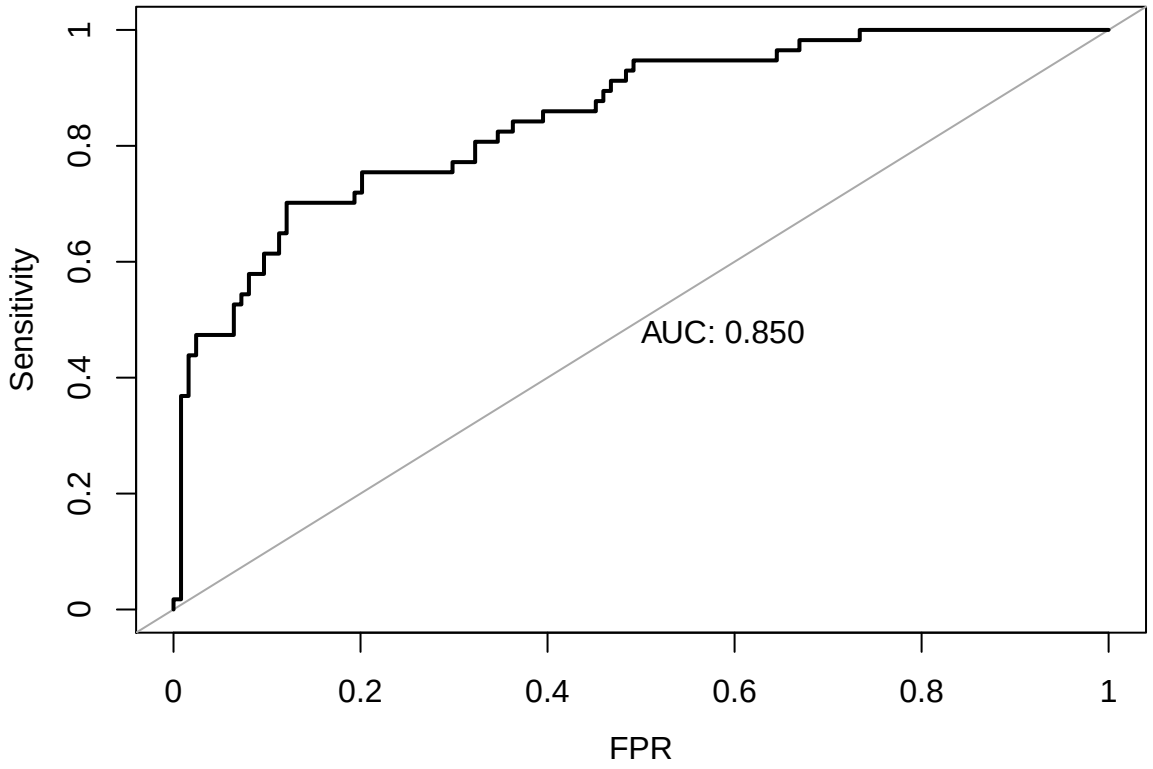
```
caretconmat = confusionMatrix(classdata$scored.class,classdata$class)
caretOutput = data.frame(t(caretconmat$byClass))

caretResults = data.frame(Accurarcy = caretconmat$overall[['Accuracy']],
                          ClassErrorRate = 1 - caretconmat$overall[['Accuracy']],
                          Precision = caretOutput$Precision,
                          Senstivity = caretOutput$Sensitivity,
                          Specificity = caretOutput$Specificity,
                          F1 = caretOutput$F1)

compareMetrics = rbind(metoutput,caretResults )
knitr::kable(compareMetrics)
```

Accurarcy	ClassErrorRate	Precision	Senstivity	Specificity	F1
0.8066298	0.1933702	0.7986577	0.9596774	0.4736842	0.8717949
0.8066298	0.1933702	0.7986577	0.9596774	0.4736842	0.8717949

The output of the comparison are same

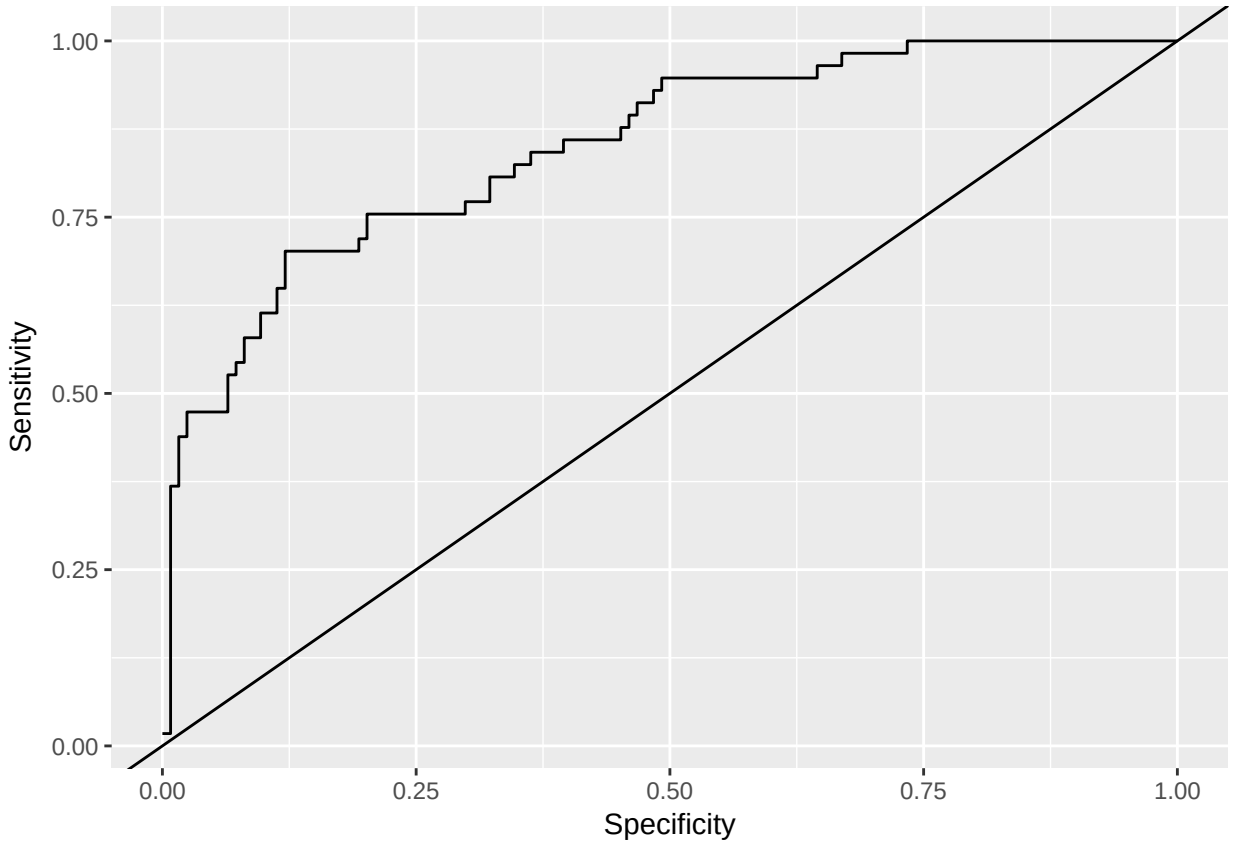
13. Investigate the pROC package. Use it to generate an ROC curve for the data set. How do the results compare with your own functions?

```
rocPlot = roc(classdata$class, classdata$scored.probability)
#rocPlot
plot(rocPlot, asp=NA, legacy.axes = TRUE, print.auc=TRUE, xlab="FPR")
```



Comparing the Plot of pROC Package with the plot generated by custom function. The AUC of custom function is 0.8503113.

```
ggplot(rocResults, aes(Specificity, Sensitivity)) +
    geom_line() +
    geom_abline()
```



References

https://datascientia.blog/2017/09/18/dss-p10-class-models/

http://blog.revolutionanalytics.com/2016/08/roc-curves-in-two-lines-of-code.html

http://blog.revolutionanalytics.com/2016/11/calculating-auc.html