621_HW2_SantoshCheruku

```
## Loading required package: lattice

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

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

```r
set.seed(1234)
data <- read.csv('classification-output-data.csv', header=T)
str(data)
```

```
## 'data.frame':    181 obs. of  11 variables:
##  $ pregnant          : int  7 2 3 1 4 1 9 8 1 2 ...
##  $ glucose           : int  124 122 107 91 83 100 89 120 79 123 ...
##  $ diastolic         : int  70 76 62 64 86 74 62 78 60 48 ...
##  $ skinfold          : int  33 27 13 24 19 12 0 0 42 32 ...
##  $ insulin           : int  215 200 48 0 0 46 0 0 48 165 ...
##  $ bmi               : num  25.5 35.9 22.9 29.2 29.3 19.5 22.5 25 43.5 42.1 ...
##  $ pedigree          : num  0.161 0.483 0.678 0.192 0.317 0.149 0.142 0.409 0.678 0.52 ...
##  $ age               : int  37 26 23 21 34 28 33 64 23 26 ...
##  $ class             : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ scored.class      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ scored.probability: num  0.328 0.273 0.11 0.056 0.1 ...
```

2. 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?

Rows represent the predicted values, and the columsn represent the actual values. And I will be considering 1 as positive case.

```r
confusionMatrixTable <- table(data$scored.class, data$class)

confusionMatrixTable
```

```
##
##       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.

```
func_accuracy <- function(data) {
  confusionMatrixTable <- table(data$scored.class, data$class)
  TP <- confusionMatrixTable[4]

  TN <- confusionMatrixTable[1]

  FP <- confusionMatrixTable[2]

  FN <- confusionMatrixTable[3]

  accuracy <- (TP + TN) / (TP + TN + FP + FN)
  return(accuracy)
}
func_accuracy(data)
```

```
## [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.

```
func_class_errorRate <- function(data) {
  confusionMatrixTable <- table(data$scored.class, data$class)
  TP <- confusionMatrixTable[4]

  TN <- confusionMatrixTable[1]

  FP <- confusionMatrixTable[2]

  FN <- confusionMatrixTable[3]


  classificationErrorRate <- (FP+FN)/sum(confusionMatrixTable)
  return(classificationErrorRate)
}
func_class_errorRate(data)
```

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

```
# Verify that you get an accuracy and an error rate that sums to one.
(func_accuracy(data) + func_class_errorRate(data))
```

```
## [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.

```r
  func_precision <- function(data) {
  confusionMatrixTable <- table(data$scored.class, data$class)
  TP <- confusionMatrixTable[4]

  TN <- confusionMatrixTable[1]

  FP <- confusionMatrixTable[2]

  FN <- confusionMatrixTable[3]


  precision <- TP / (TP + FP)
  return(precision)
  }
func_precision(data)
```

```
## [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.

```r
func_sensitivty <- function(data) {
  confusionMatrixTable <- table(data$scored.class, data$class)
  TP <- confusionMatrixTable[4]

  TN <- confusionMatrixTable[1]

  FP <- confusionMatrixTable[2]

  FN <- confusionMatrixTable[3]


  sensitivity <- (TP) / (TP + FN)
  return(sensitivity)
}
func_sensitivty(data)
```

```
## [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
func_specificity <- function(data) {
  confusionMatrixTable <- table(data$scored.class, data$class)

  TP <- confusionMatrixTable[4]

  TN <- confusionMatrixTable[1]

  FP <- confusionMatrixTable[2]

```

```r
  FN <- confusionMatrixTable[3]



  specificity <- (TN) / (TN + FP)
  return(specificity)
}
func_specificity(data)
```

```
## [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
func_f1Score <- function(data) { val <- 2 * func_precision(data) *  func_sensitivty(data) / (func_preci
return(val)
}
```

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 < < 1 and 0 < < 1 then < .)

```r
func_f1Score(data)
```

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

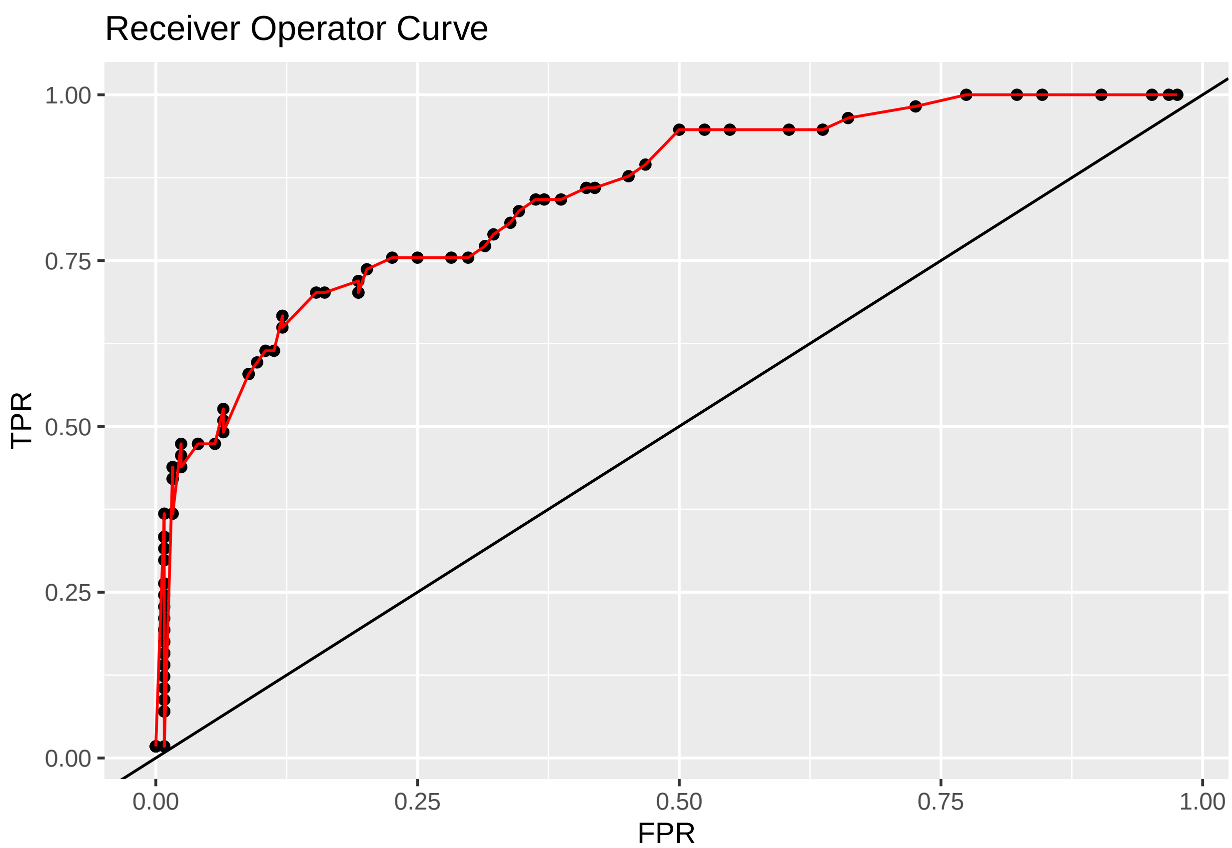
The F1Score is bound within 0 to 1, because the values of precision and sensitivity are always less than 1, hence the multiplication of two lower bound number always results in less than 1 and as well as less than their sum.

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.

```r
func_roc_curve <- function(data) {
  confusionMatrix <- table(data$scored.class, data$class)
  df <- data.frame(i=NA,TPR=NA,FPR=NA)
  val <- 0.0
  for(i in c(1:99)) {
    val <- val + 0.01
    data$score_newsclass <- as.numeric(data$scored.probability>val)
    confusionMatrix <- table(data$score_newsclass, data$class)
    TP <- confusionMatrix[4]

    TN <- confusionMatrix[1]

    FP <- confusionMatrix[2]

    FN <- confusionMatrix[3]
```

```r
    FPR <- FP / (FP + TN)
    TPR <- TP / (TP + FN)
    df <- rbind(df, c(val,TPR, FPR))
  }
  df <- na.omit(df)

  #Plot the values
  plot <- ggplot(df, aes(x=FPR, y=TPR)) +
          geom_point() +
          geom_line(col="red") +
          geom_abline(intercept = 0, slope = 1) +
          labs(title="Receiver Operator Curve", x = "FPR", y = "TPR")


  # Calculate the AUC
  a <- abs(diff(df$FPR))
  b <- df$TPR
  auc <- sum(a*b)

  return(list(plot, auc))
}

func_roc_curve(data)[[1]]
```

```
## Warning in a * b: longer object length is not a multiple of shorter object
## length
```




```r
func_roc_curve(data)[[2]]
```

```
## Warning in a * b: longer object length is not a multiple of shorter object
## length

## [1] 0.8299377
```

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

```r
func_accuracy(data)
```

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

```r
func_class_errorRate(data)
```

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

```r
func_precision(data)
```

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

```r
func_sensitivty(data)
```

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

```r
func_specificity(data)
```

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

```r
func_f1Score(data)
```

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

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?

Both the caret package as well as Rcode written function, have generated similar results for Sensitivity and Specifictiy.

```r
matrix <- confusionMatrix(confusionMatrixTable, positive ='1')
matrix
```

```
## Confusion Matrix and Statistics
##
##
##       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
##
```

```r
matrix$byClass[['Sensitivity']]
```

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

```r
matrix$byClass[['Specificity']]
```

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

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?

Both the functions generated similar ROC curves.

```r
plot(roc(data$class, data$scored.probability), main="ROC Curve")
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```