Data 621 Project 02
Section 01 Group 5

Farhana Zahir, Habib Khan, Vijaya Cherukuri, Scott Reed, Shovon Biswas

10/02/2020

Contents

```r
# loading libraries
library(tidyverse)
library(caret)
library(pROC)
library(kableExtra)
```

Description

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.

Supplemental Material

Applied Predictive Modeling, Ch. 11 (provided as a PDF file). Web tutorials: http://www.saedsayad.com/model_evaluation_c.htm

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.

Solution 01

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

```r
#read in the data
data<- read.csv("https://raw.githubusercontent.com/zahirf/Data621/master/HW02/classification-output-data
```

Let us take a peak at the data first. At a glance it looks like the dependent variable class was regressed against several independent variables. The Scored class is the predicted variable, and the scored.probability shows the probability that the scored.class belongs to a class of 1. A further description of the variables is given below:

pregnant: no of times pregnant glucose: plasma glucose concentration diastolic: diastolic blod pressure skinfold: triceps skin fold thickness insulin: serum insulin test bmi: body mass index pedigree: diabetes pedigree function age: age in years class: (1: positive for diabetes, 0 negative for diabetes)

(Ref: https://www.kaggle.com/kumargh/pimaindiansdiabetescsv)

```r
head(data)
```

```
## # A tibble: 6 x 11
##   pregnant glucose diastolic skinfold insulin   bmi pedigree   age class
##      <int>   <int>     <int>    <int>   <int> <dbl>    <dbl> <int> <int>
## 1        7     124        70       33     215  25.5    0.161    37     0
## 2        2     122        76       27     200  35.9    0.483    26     0
## 3        3     107        62       13      48  22.9    0.678    23     1
## 4        1      91        64       24       0  29.2    0.192    21     0
## 5        4      83        86       19       0  29.3    0.317    34     0
## 6        1     100        74       12      46  19.5    0.149    28     0
## # ... with 2 more variables: scored.class <int>, scored.probability <dbl>
```

Solution 02

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?

Let us look at the actual class and predicted class separately.

Actual class

```
table(data$class, dnn = "Actual class")
```

```
## Actual class
##   0   1
## 124  57
```

Predicted class

```
table(data$scored.class, dnn = "Predicted class")
```

```
## Predicted class
##   0   1
## 149  32
```

Raw confusion matrix for the data

```
table(data$scored.class, data$class,
      dnn = c("Predicted", "Actual"))
```

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

A confusion matrix shows the number of correct and incorrect predictions made by the model compared to the actual outcomes.

Following the classification laid down in https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative,

We can see that:

TP True Positive Row1Col1: 119 correct predictions were made about class 0 (Actual 0 and Predicted 0)

TN True Positive Row2Col2: 27 correct predictions were made about class 1 (Actual 1 and Predicted 1)

FN False Positive Row2Col1: 5 of the observations had an actual value of 0 but predicted as 1.

FP False Negative Row1Col2: 30 of the observations had an actual value of 1 but predicted as 0

Solution 03

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

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

The function is below:

```r
getAccuracy <- function(df){
   confusion_matrix <- table(df$scored.class, df$class,
                             dnn = c("Predicted", "Actual"))
   TN <- confusion_matrix[2,2]
   FN <- confusion_matrix[2,1]
   FP <- confusion_matrix[1,2]
   TP <- confusion_matrix[1,1]
   Accuracy <- (TP+TN)/(TP+FP+TN+FN)
   #print(paste0("The Accuracy rate is ", sprintf("%1.2f%%", 100*Accuracy)))
   return(Accuracy)
}
```

We run the function on our data and find an accuracy rate of 80.7%.

```r
getAccuracy(data)
```

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

We can do the same using the caret package and it gives us the same result.

```r
acc<-confusionMatrix(table(data$scored.class, data$class))
acc$overall['Accuracy']
```

```
##  Accuracy
## 0.8066298
```

Solution 04

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.

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

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

WE have created the function below to calculate Classification error rate.

```r
getClassError <- function(df){
   confusion_matrix <- table(df$scored.class, df$class,
                             dnn = c("Predicted", "Actual"))
   TN <- confusion_matrix[2,2]
```

```
    FN <- confusion_matrix[2,1]
    FP <- confusion_matrix[1,2]
    TP <- confusion_matrix[1,1]
    ClassificationErrorRate <- (FP+FN)/(TP+FP+TN+FN)
    return(ClassificationErrorRate)

}
```

We run it on our data

```
getClassError(data)
```

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

The Accuracy and Error rates sum to 1.

```
print(paste0("The sum is ", (getClassError(data) + getAccuracy(data))))
```

```
## [1] "The sum is 1"
```

Solution 05

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

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

The R function for Precision, also known as Positive Predictive value (PPV), is as follows:

```
getPrecision <- function(df){
    confusion_matrix <- table(df$scored.class, df$class,
                              dnn = c("Predicted", "Actual"))
    TN <- confusion_matrix[2,2]
    FN <- confusion_matrix[2,1]
    FP <- confusion_matrix[1,2]
    TP <- confusion_matrix[1,1]
    Precision <- (TP)/(TP+FP)
    return(Precision)
}
```

Running it on our data

```
getPrecision(data)
```

```
## [1] 0.7986577
```

Verify using caret

```r
posPredValue(table(data$scored.class, data$class))
```

```
## [1] 0.7986577
```

Solution 06

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.

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

Sensitivity is also known as Recall, Hit rate or True Positive Rate (TPR).

The R function to calculate recall is as follows:

```r
getSensitivity <- function(df){
   confusion_matrix <- table(df$scored.class, df$class,
                             dnn = c("Predicted", "Actual"))
   TN <- confusion_matrix[2,2]
   FN <- confusion_matrix[2,1]
   FP <- confusion_matrix[1,2]
   TP <- confusion_matrix[1,1]
   Sensitivity <- (TP)/(TP+FN)
   return(Sensitivity)
}
```

Running it on our data

```r
getSensitivity(data)
```

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

Verify using caret

```r
sensitivity(table(data$scored.class, data$class))
```

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

Solution 07

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

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

Specificity is also called selectivity or True Negative Rate (TNR). The R fundtion to calculate this is as follows:

```r
getSpecificity <- function(df){
   confusion_matrix <- table(df$scored.class, df$class,
                        dnn = c("Predicted", "Actual"))
   TN <- confusion_matrix[2,2]
   FN <- confusion_matrix[2,1]
   FP <- confusion_matrix[1,2]
   TP <- confusion_matrix[1,1]
   Specificity <- (TN)/(TN+FP)
   return(Specificity)
}
```

Running on our data

```r
getSpecificity(data)
```

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

Verify using caret

```r
specificity(table(data$scored.class, data$class))
```

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

Solution 08

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.

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

F1 Score is the harmonic mean of precision and sensitivity. The highest possible value of F1 is 1, indicating perfect precision and recall, and the lowest possible value is 0, if either the precision or the recall is zero.

The R function is below:

```r
getF1Score <- function(df){
   confusion_matrix <- table(df$scored.class, df$class,
                        dnn = c("Predicted", "Actual"))
   TN <- confusion_matrix[2,2]
   FN <- confusion_matrix[2,1]
   FP <- confusion_matrix[1,2]
   TP <- confusion_matrix[1,1]
   Sensitivity <- (TP)/(TP+FN)
   Precision <- (TP)/(TP+FP)
   F1Score <- (2 * Precision * Sensitivity)/(Precision + Sensitivity)
   return(F1Score)
}
```

Running on our data

```r
getF1Score(data)
```

```
## [1] 0.8717949
```

Verify using caret

```r
acc$byClass['F1']
```

```
##        F1
## 0.8717949
```

Solution 09

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 a b < a)

Precision values can range from 0 to 1

$$0 \geq P \geq 1$$

Sensitivity values can also range from 0 to 1

$$0 \geq S \geq 1$$

Using If 0 < a < 1 and 0 < b < 1 then a b < a, we get

$$PS \leq S$$
$$PS \leq P$$

This implies that

$$0 \leq PS \leq P \leq 1$$
$$0 \leq PS \leq S \leq 1$$

The numerator in the equation ranges from 0 to 1 The denominator ranges from 0 to 2 Any reaulting quotient will range from 0 to 1.

Solution 10

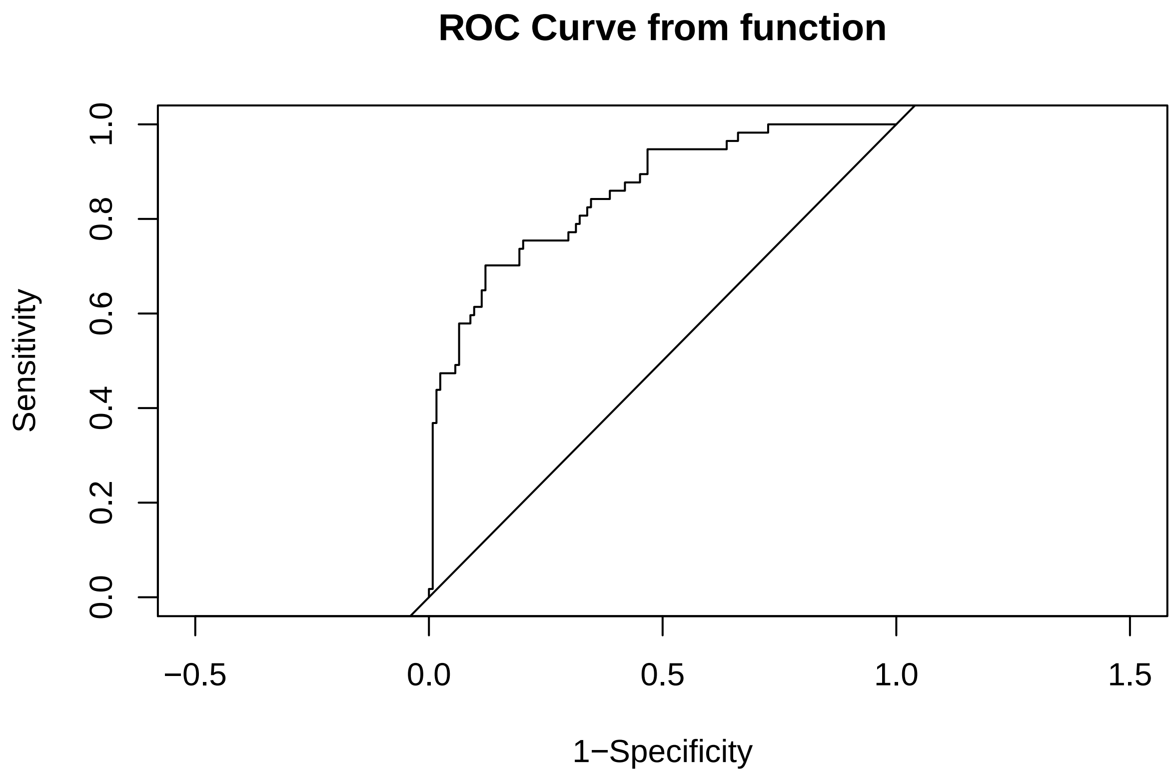
Write a function that generates an ROC curve from a data set with a true classification column (class in our example) and a probability column (scored.probability in our example). Your function should return a list that includes the plot of the ROC curve and a vector that contains the calculated area under the curve (AUC). Note that I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.

The function is given below

```r
Roc_function<- function(d){ #d stands for dataframe that you will pass in function
  #Create a count
  temp <- table(d[ ,'class'], d[ ,"scored.probability"])
  #Calculate frequency
  allPos <- sum(data$class == 1, na.rm=TRUE)
  allNeg <- sum(data$class == 0, na.rm=TRUE)
  #Set threshold
  threshold <- seq(0,1,0.01)
  #Calculating probability for threshold
  x <- c()
  y <- c()
  for (i in 1:length(threshold)) {
    TP <- sum(data$scored.probability >= threshold[i] & data$class == 1, na.rm=TRUE)
    TN <- sum(data$scored.probability < threshold[i] & data$class == 0, na.rm=TRUE)
    y[i] <- TP / allPos
    x[i] <- 1-TN / allNeg
  }

  rocPlot <- plot(x,y,type = "s", xlim=c(-0.5,1.5),
              main = "ROC Curve from function",
              xlab = "1-Specificity",
              ylab = "Sensitivity")
  fPlot <- abline(0,1); fPlot

  xd <- c(0, abs(diff(x)))
  fAuc <- sum(xd*y); fAuc

  print(paste0("Area under the curve: ", fAuc))
}
```

Let us call the function on our data

```r
Roc_function(data)
```




```
## [1] "Area under the curve: 0.843803056027165"
```

Solution 11

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

The classification metrics can be found in the table below:

```
df1 <- c(getAccuracy(data),
         getClassError(data),
         getPrecision(data),
         getSensitivity(data),
         getSpecificity(data),
         getF1Score(data))
names(df1) <- c("Accuracy", "Classification Error", "Precision",
                "Sensitivity", "Specificity", "F1 Score")
df1<-as.data.frame(df1)
names(df1)[1]<-'Scores'
kbl(df1)%>%
   kable_classic("hover", full_width = F, html_font = "Cambria")
```

	Scores
Accuracy	0.8066298
Classification Error	0.1933702
Precision	0.7986577
Sensitivity	0.9596774
Specificity	0.4736842
F1 Score	0.8717949

Solution 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?

We have already tested out our calculations with the caret package for each part. We will show it here. We find that the values we calculated using our R functions are exactly the same as those calculated by the caret package.

```
df2<- confusionMatrix(data = as.factor(data$scored.class),
reference = as.factor(data$class),
positive = '0')
df2
```

```
## 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 : 0.9597
##             Specificity : 0.4737
##          Pos Pred Value : 0.7987
##          Neg Pred Value : 0.8438
##              Prevalence : 0.6851
##          Detection Rate : 0.6575
##    Detection Prevalence : 0.8232
##       Balanced Accuracy : 0.7167
##
##        'Positive' Class : 0
##
```

Solution 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?

```
#Generate the function
rCurve <- roc(data$class, data$scored.probability, levels=c(1,0), direction=">")
```

Area under the curve

```
auc(rCurve)
```

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

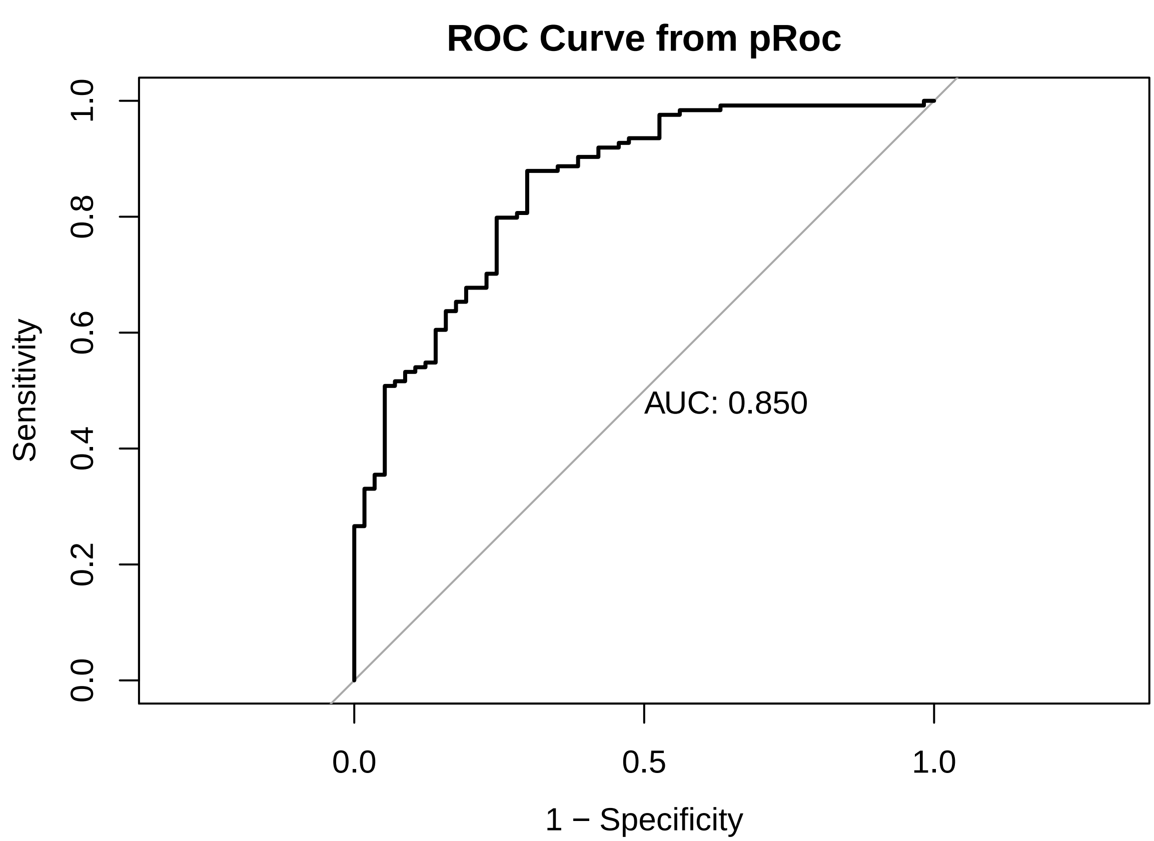
Confidence interval for the curve

```
ci(rCurve)
```

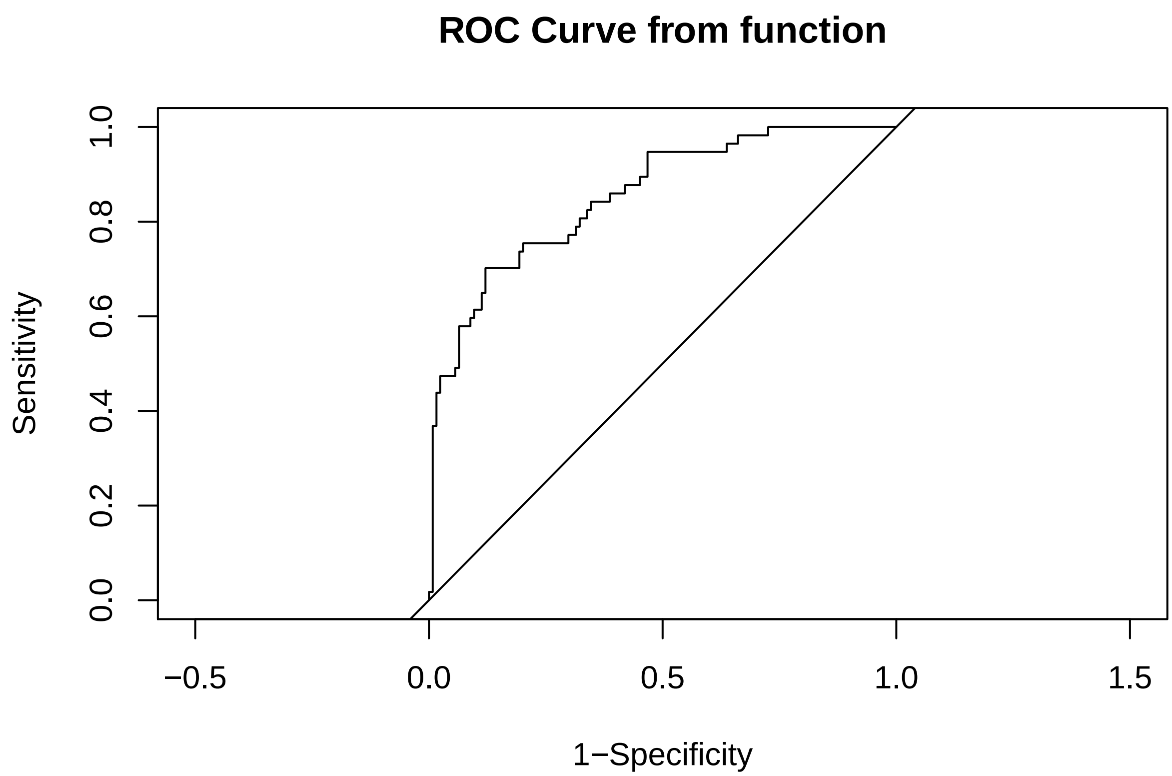
```
## 95% CI: 0.7905-0.9101 (DeLong)
```

Let us compare the ROC curve from the pRoc package to the one we generates in Solution 10. We see that graph looks the same, however we got Area under the curve of 0.8438 compared to 0.8503 from the pRoc package.

```
plot(rCurve, main="ROC Curve from pRoc", legacy.axes = TRUE, print.auc=TRUE)
```



```r
Roc_function(data)
```




```
## [1] "Area under the curve: 0.843803056027165"
```

References

1. https://www.kaggle.com/kumargh/pimaindiansdiabetescsv
2. https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative
3. https://en.wikipedia.org/wiki/Confusion_matrix
4. https://rdrr.io/cran/caret/man/sensitivity.html
5. https://stackoverflow.com/questions/41056896/proc-changing-scale-of-the-roc-chart

Github link for code

https://github.com/zahirf/Data621/blob/master/HW02/Data621-HW2.Rmd