Data 621 - Homework 2

Group 1

October, 2019

Table of Contents

Overview

In this homework assignment, we are to work through various classification metrics.

Objectives

- To create functions in R to carry out the various calculations
- To investigate some functions in packages that will obtain the equivalent results
- To create graphical output that can be used to evaluate the output of classification models, such as binary logistic regression.

Import data

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

Using the read.csv function we attach the imported data, **class_data**, making its variables available to be referenced by name. We import the data and output the first 10 rows with the head function, using kable to format the output for readability.

Classification Output Dataset

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
9	89	62	0	0	22.5	0.142	33	0	0	0.1071154
8	120	78	0	0	25.0	0.409	64	0	0	0.4599474
1	79	60	42	48	43.5	0.678	23	0	0	0.1170237
2	123	48	32	165	42.1	0.520	26	0	0	0.3153632

We check for any missing data and verify that the dataset is complete.

```
sum(is.na(class_data))
```

```
## [1] 0
```

Write R Functions

Raw Confusion matrix

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?

a. We wrap the predictions and actual results in a table which returns a raw confusion matrix and print out the table.

b. We observe that each column, (actual) class, categorizes the result by a false score (0) and a true score (1). Each each row (predicted) scored.class categorizes the result by a negative score (0) and a positive score (1).

c. Therefore when the predicted class is negative (0) and the actual class is false (0), the result is a true negative. Likewise when the actual and predicted class are both positive and true (1), the result is a true positive. A false positive result (a Type I error) refers to when the predicted result was positive however the actual class was negative. A false netgative result (a Type II error) refers to when the predicted result was negative however the actual class was positive.

d. The table index positions 1 and 2 are accessed with brackets and returned as a variable representing true-positives, true-negatives, false-positives, and false-negatives as printed below.

```
(tbl <- table(class_data$scored.class, class_data$class))

##
##       0   1
##   0 119  30
##   1   5  27

tn <- tbl[1,1]
tp <- tbl[2,2]
fp <- tbl[2,1]
fn <- tbl[1,2]
```

Confusion matrix parameters

The confusion matrix is as follows, were again columns are actual class, and rows are predicted class (scored.class):

$\begin{array}{|c|c|} \hline & 0 & 1 \\ \hline 0 & TN & FN \\ 1 & FP & TP \\ \hline \end{array} = \begin{array}{|c|c|} \hline & 0 & 1 \\ \hline 0 & 119 & 30 \\ 1 & 5 & 27 \\ \hline \end{array}$

True positive (TP) is: 27
True negative (TN) is: 119
False positive (FP) is: 5
False negative (FN) is: 30

Accuracy of the Predictions

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 : the proportion of the total number of predictions that were correct.

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

```
prd_accuracy <- function () {
    accuracy = round((tp + tn) / (tp + fp + tn + fn),10)
    return (accuracy)
}
```

The prediction accuracy is: **0.8066298**

Classification Error Rate of the Predictions

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: the proportion of total prediction that were incorrect.

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

```
prd_class_error <- function () {
    class_err_rate = round((fp + fn) / (tp + fp + tn + fn),4)
    return (class_err_rate)
}
```

The classification error rate of the prediction is: **0.1934**

Precision of the Predictions

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: proportion of positive predictions that were correct.

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

```
prd_precision <- function () {
    precision = round(tp / (tp + fp),4)
    return (precision)
}
```

The prediction precision is: **0.8438**

Sensitivity of the Predictions

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 or Recall : the proportion of actual positive cases which are correctly identified.

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

```
prd_recall <- function () {
    recall = round(tp / (tp + fn),4)
    return (recall)
}
```

The prediction sensitivity is: **0.4737**

Specificity of the Predictions

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: proportion of actual negatives that were correctly identified.

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

```
prd_specificity <- function () {
    specificity = round(tn / (tn + fp),4)
    return (specificity)
}
```

The prediction specificity is: **0.9597**

F1 Score of the Predictions

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: it is a balanced measure of a classifier's which uses precision and sensitivity (recall). It is defined as the weighted harmonic mean of the test's precision and sensitivity(recall).

$$F_1\ Score = \frac{2 * Precision * Sensitivity}{Precision + Sensitivity}$$

```
prd_f1_score <- function () {
    f1_score = round((2*prd_precision()*prd_recall()) / (prd_precision()+prd_
recall()),4)
    return (f1_score)
}
```

The F1 score of the prediction is: **0.6068**

F1 Score range

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.

The F score reaches the best value, meaning perfect precision and recall, at a value of 1. The worst F score, which means lowest precision and lowest recall, would be a value of 0. We can see this by observing that precision and sensitivity are also bound between 0 and 1.

For the max value we know that the numerator of $F_1 Score$ has a maximum of 2, that is Precision*Sensitivity=1 then multiplied by 2. The denominators maximum value is also 2 as max Precision + max Sensitivity is 1+1. The the maximum value of $F_1 Score$ is 1

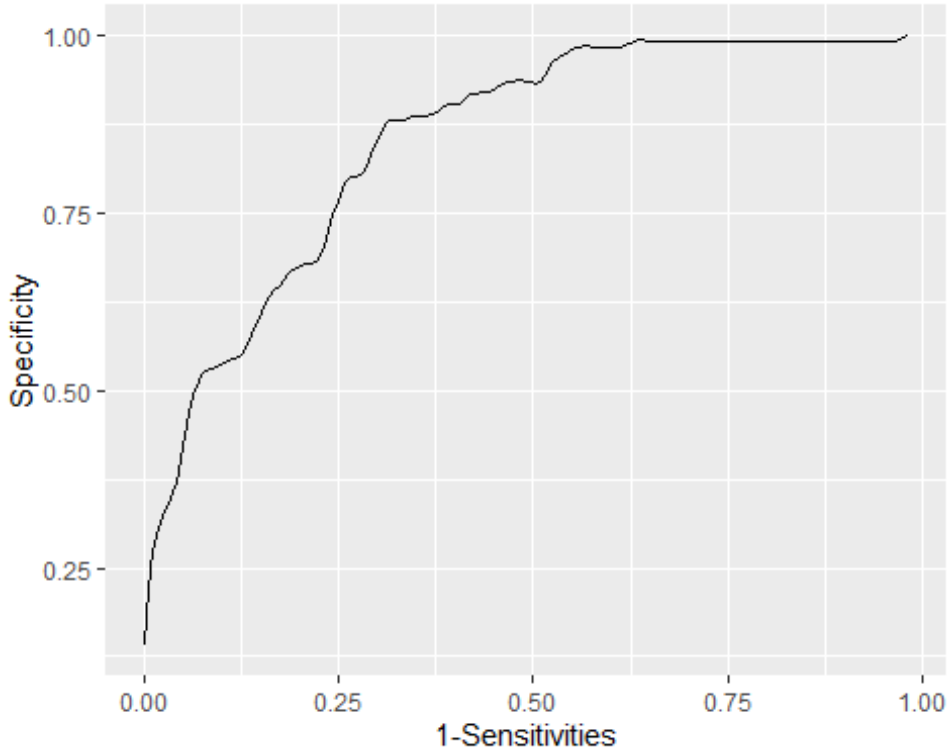
$$F_1\ Score = \frac{2 * 1 * 1}{1 + 1} = \frac{2}{2} = 1$$

For the minimum we simply observe that a Precision or Sensitivity of zero will result in a $F_1 Score$ of zero.

ROC Function

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 generates the ROC by calculating true positives (TP) and false negatives (FN) between 0 and 1 with increments of 1 divided by the size of the dataset (181 in this case). The area under the curve under the curve AUC is calculated by multiplying each increment in FP by the max TP for that increment, which gives us the area under the ROC. Note the plot is our roc function given spline interpolation during plotting.



The calculated AUC is **0.84**

All Classification Metrics

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

```
Name <- c('Accuracy','Classification Error Rate', 'Precision', 'Sensitivity', 'Specificity', 'F1 Score')
Value <- round(c(prd_accuracy(), prd_class_error(), prd_precision(), prd_recall(), prd_specificity(), prd_f1_score()),4)
df1 <- as.data.frame(cbind(Name, Value))
kable(df1)
```

Name	Value
Accuracy	0.8066
Classification Error Rate	0.1934
Precision	0.8438
Sensitivity	0.4737
Specificity	0.9597
F1 Score	0.6068

Caret Package

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 first run the confusionMatrix function from the caret package

```
library(caret)

class_data$class <- as.factor(class_data$class)
class_data$scored.class <- as.factor(class_data$scored.class)

(cm<-confusionMatrix(class_data$scored.class, class_data$class, positive = "1
"))
```

```
## 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
##
```

Now we can compare these results with the one computed with our functions and confirm we obtain the same result:

Accuracy

```
round(cm$overall[1],4)==round(prd_accuracy(),4)
```

```
## Accuracy
##     TRUE
```

Sensitivity

```
round(cm$byClass[1],4)==round(prd_recall(),4)

## Sensitivity
##        TRUE
```

Specificity

```
round(cm$byClass[2],4)==round(prd_specificity(),4)

## Specificity
##        TRUE
```

Precision

```
round(cm$byClass[5],4)==round(prd_precision(),4)

## Precision
##      TRUE
```

F1

```
round(cm$byClass[7],3)==round(prd_f1_score(),3)

##   F1
## TRUE
```

We can also explore the two caret function for Sensitivity and Specificity.

Sensitivity.

```
(c_sen<-sensitivity(class_data$scored.class, class_data$class, positive = "1"
))

## [1] 0.4736842

round(c_sen,4)==prd_recall()

## [1] TRUE
```

Specificity

```
(c_spe<-specificity(class_data$scored.class, class_data$class, negative = "0"
))

## [1] 0.9596774

round(c_spe,4)==prd_specificity()

## [1] TRUE
```

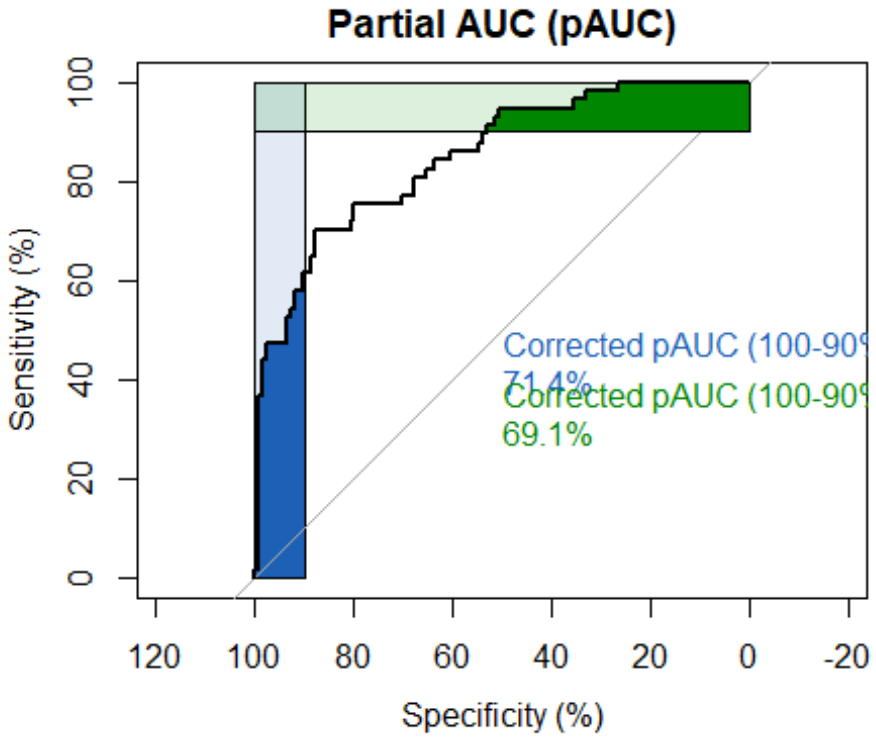
pROC Package

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?

The ROC curve can be plotted using Specificity and Sensitivity as the axis. This is equivalent to the FP and TP axis plot we performed previously and these two metrics are calculated using FP and TP. With this in mind we find both the ROC plotted with our previous function and that plotted with the pROC ROC function are similar graphs.

Defining the partcial AUC (pAUC) and also showing th pAUC as a polygon

```
if(!require(pROC)) install.packages("pROC")
plot.roc(class_data$class,class_data$scored.probability,
         percent = TRUE,                    # show all values in percent
         partial.auc=c(100, 90),
         partial.auc.correct=TRUE,
         print.auc=TRUE,
         #display pAUC value on the plot with following options:
         print.auc.pattern = "Corrected pAUC (100-90%% SP):\n%.1f%%",
         print.auc.col = "#1c61b6",
         auc.polygon = TRUE,
         auc.polygon.col = "#1c61b6",
         max.auc.polygon = TRUE,
         max.auc.polygon.col = "#1c61b622",
         main = "Partial AUC (pAUC)")
plot.roc(class_data$class,class_data$scored.probability,
         percent = TRUE,
         add = TRUE,
         type = "n",
         partial.auc = c(100, 90),
         partial.auc.correct = TRUE,
         partial.auc.focus = "se",          # focus pAUC on the sensitivity
         print.auc = TRUE,
         print.auc.pattern = "Corrected pAUC (100-90%% SE):\n%.1f%%",
         print.auc.col = "#008600",
         print.auc.y = 40,
         auc.polygon = TRUE,
         auc.polygon.col = "#008600",
         max.auc.polygon = TRUE,
         max.auc.polygon.col = "#00860022")
```

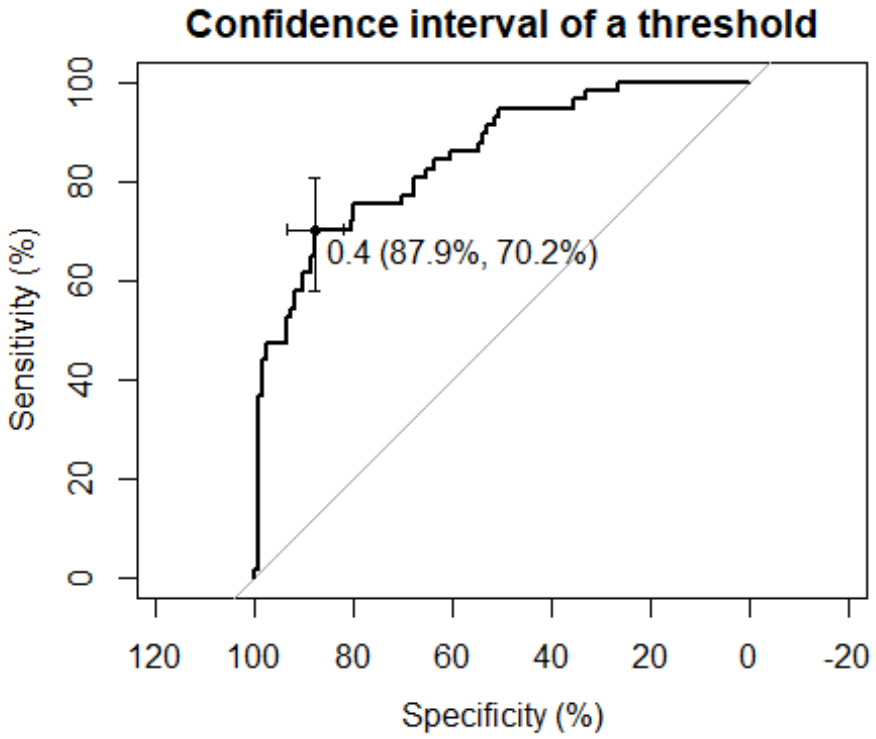



```
plot.roc(class_data$class,class_data$scored.probability,
main="Confidence interval of a threshold", percent=TRUE,
ci=TRUE, of="thresholds", # compute AUC (of threshold)
thresholds="best", # select the (best) threshold
print.thres="best") # also highlight this threshold on the plot

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

## Setting direction: controls < cases

## Warning in coords.roc(roc, x = thresholds, input = "threshold", ret
## = "threshold", : An upcoming version of pROC will set the 'transpose'
## argument to FALSE by default. Set transpose = TRUE explicitly to keep the
## current behavior, or transpose = FALSE to adopt the new one and silence
## this warning. Type help(coords_transpose) for additional information.
```



Here we plot our ROC against the graph generated from the base pROC graph.

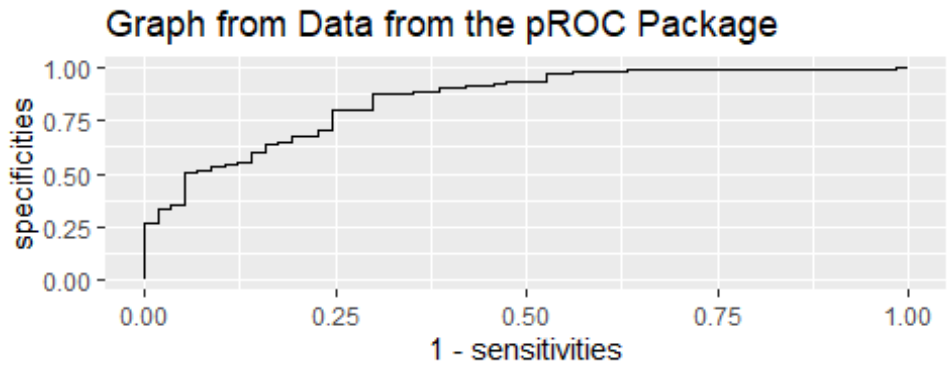
Note in this case our plotted graph is not done with spline interpolation, so as to more closely match the output from the pROC graph.

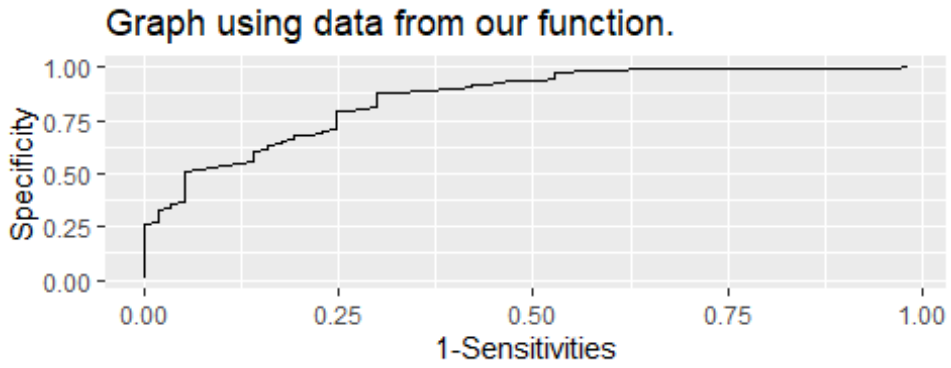
```
library(pROC)
rocCurve <- roc(response = class_data$class,
                predictor = class_data$scored.probability)
g1 <- ggplot(data.frame(sensitivities = rocCurve$sensitivities, specificities
= rocCurve$specificities), aes(1-sensitivities, specificities))  +
    geom_line() +
    ggtitle("Graph from Data from the pROC Package")

require(gridExtra)

g2 <- rocResults[[1]] + # graph from above
    ggtitle("Graph using data from our function.")

gridExtra::grid.arrange(g1,g2)
```





We can also confirm the AUC calculated previously is almost identical to that calculated by the AUC pROC function.

```
aucPROC <- round(auc(rocCurve),2)
aucOurs <- round(sum(unlist(rocResults[3])),2)
```

The calculated AUC from the pRoc package is **0.85** The calculated AUC from our function is **0.84**