Business Analytics and Data Mining - Homework 2

Abdellah AitElmouden | Gabriel Abreu | Jered Ataky | Patrick Maloney

3/16/2021

Objective

Classification is the process of predicting a categorical label of a data object based on its features and properties. In this assignment we created R functions to calculate several different classification metrics as R functions. We also verified the functions by checking R package implementations against our output. Lastly, we graphed the output of the classification model.

Dataset

The data set was provided to as csv file. On first examination of the data, 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:

Variables	Description
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	the actual class for the observation (1: positive for diabetes, 0 negative for diabetes)
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

The dataset has three key columns we will use: class, scored.class and scored.probability.

```
data_url <- "https://raw.githubusercontent.com/aaitelmouden/DATA621/master/Homework2/classification-outp
data <- read.csv(data_url)
```

Confusion Matrix

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?

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

data_table
```

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

In the output above, the rows represent the predicted values for the class of each observation (ie. the scored.class value), while the columns represent the actual values for class. Thus, the first quadrant is the true negatives, the second quadrant is the number of False Negatives, the third quadrant is the number of False Positives, while the fourth quadrant is the number of True Positives:

- 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

R Accuracy Function

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.

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

```
calc_accuracy <- function(df){
  TP <- sum(df$class == 1 & data$scored.class == 1)
  TN <- sum(df$class == 0 & data$scored.class == 0)
  (TP + TN)/nrow(df)
}

calc_accuracy(data)
```

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

R Classification Error Function

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.

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

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

```
calc_error_rate <- function(df){
   FP <- sum(df$class == 0 & data$scored.class == 1)
   FN <- sum(df$class == 1 & data$scored.class == 0)
   (FP + FN)/nrow(df)
}

calc_error_rate(data)
```

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

R Precision of the Predictions Function

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.

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

```
calc_precision <- function(df){
  FP <- sum(df$class == 0 & data$scored.class == 1)
  TP <- sum(df$class == 1 & data$scored.class == 1)
  TP/(TP +FP)
}

calc_precision(data)
```

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

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

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

R Sensitivity Function

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.

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

```
calc_sensitivity <- function(df){
  TP <- sum(df$class == 1 & data$scored.class == 1)
  FN <- sum(df$class == 1 & data$scored.class == 0)
  TP/(TP + FN)
}

calc_sensitivity(data)
```

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

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

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

R Specificity Function

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.

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

```
calc_specificity <- function(df){
  FP <- sum(data$class == 0 & data$scored.class == 1)
  TN <- sum(data$class == 0 & data$scored.class == 0)
  TN/(TN + FP)
}

calc_specificity(data)
```

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

R F1 Score Function

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.

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

```r
calc_f1 <- function(df){
  (2* calc_precision(df)*calc_sensitivity(df))/(calc_precision(df) + calc_sensitivity(df))
}

calc_f1(data)
```

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

F1 score Bounds

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
# To show this, we are going to use uniform distribution with function runif to generate
# random deviates Then run the calc_f1 function and show that min and max values of f1_score
# will always fall are within 0 and 1

# let pre be the precision and sens be the sensitivity (values in percent/between 0 and 1)

pre <- runif(100, min = 0, max = 1)
sens <- runif(100, min = 0, max = 1)

f1_score <- (2 * pre * sens) / (pre + sens)

summary(f1_score)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.001949 0.226569 0.418347 0.435022 0.649418 0.952422
```

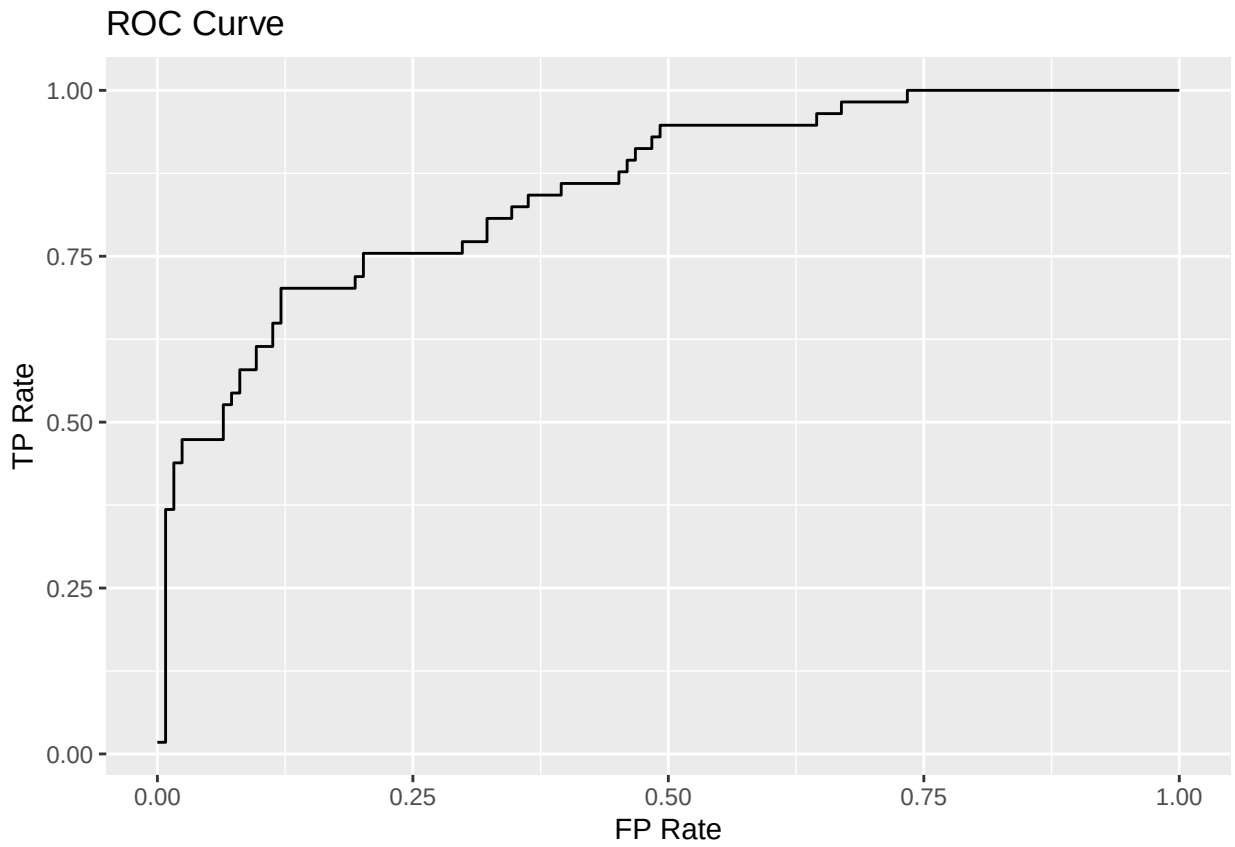
R ROC curve Function

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.

```r
ROC_function <- function(df){

  # Select the needed variables

  df <- select(df, class, scored.probability)

  x <- df$class
  y<- df$scored.probability

  # Calculate TPR and FPR

  x <- x[order(y, decreasing = TRUE)]
```

```
  TPR <- cumsum(x) / sum(x)
  FPR <- cumsum(!x) / sum(!x)

  # Calculate AUC

  # New df

  df <- data.frame(TPR, FPR, x)


  # Inputs already sorted, best scores first

  df_FPR <- c(diff(df$FPR), 0)
  df_TPR <- c(diff(df$TPR), 0)

  #  AUC = total area is the dot product of these vectors

  AUC <- round(sum(df$TPR * df_FPR) + sum(df_TPR * df_FPR)/2, 4)

  # Plot the curve

  g <- ggplot(df, aes(FPR, TPR)) + geom_line() + ggtitle("ROC Curve") +
    xlab("FP Rate") + ylab("TP Rate")


  # Return a list

  return(list(plot=g, AUC=AUC))

}

# Call the function with our data frame

ROC_function(data)
```

```
## $plot
```




```
##
## $AUC
## [1] 0.8503
```

R Precision of the Predictions Function

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

```
Metric <- c('Accuracy','Classification Error Rate', 'Precision', 'Sensitivity','Specificity', 'F1 Score
Value <- round(c(calc_accuracy(data), calc_error_rate(data), calc_precision(data), calc_sensitivity(data

# New variable that calculate the value en %
Percent_value <- Value * 100
df1 <- data_frame(Metric, Value, Percent_value)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

```
names(df1)[3] = "Value(%)"


kable(df1)
```

Metric	Value	Value(%)
Accuracy	0.8066	80.66
Classification Error Rate	0.1934	19.34
Precision	0.8438	84.38
Sensitivity	0.4737	47.37
Specificity	0.9597	95.97
F1 Score	0.6067	60.67

R 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?

```
# Using
# Confusion matrix, cm

cm <- confusionMatrix(data = as.factor(data$scored.class),
                      reference = as.factor(data$class),
                      positive = "1")

cm$table
```

```
##            Reference
## Prediction   0   1
##          0 119  30
##          1   5  27
```

```
# Get the metric from confusionMatrix features

Value <- round(c(cm$overall[1], 1 - cm$overall[1], cm$byClass[3],
            cm$byClass[1], cm$byClass[2], cm$byClass[7]), 4)


Percent_value <- Value * 100
df1 <- data_frame(Metric, Value, Percent_value)

names(df1)[3] = "Value(%)"


kable(df1)
```

Metric	Value	Value(%)
Accuracy	0.8066	80.66
Classification Error Rate	0.1934	19.34
Precision	0.8438	84.38
Sensitivity	0.4737	47.37
Specificity	0.9597	95.97
F1 Score	0.6067	60.67

The results are the same as our own functions

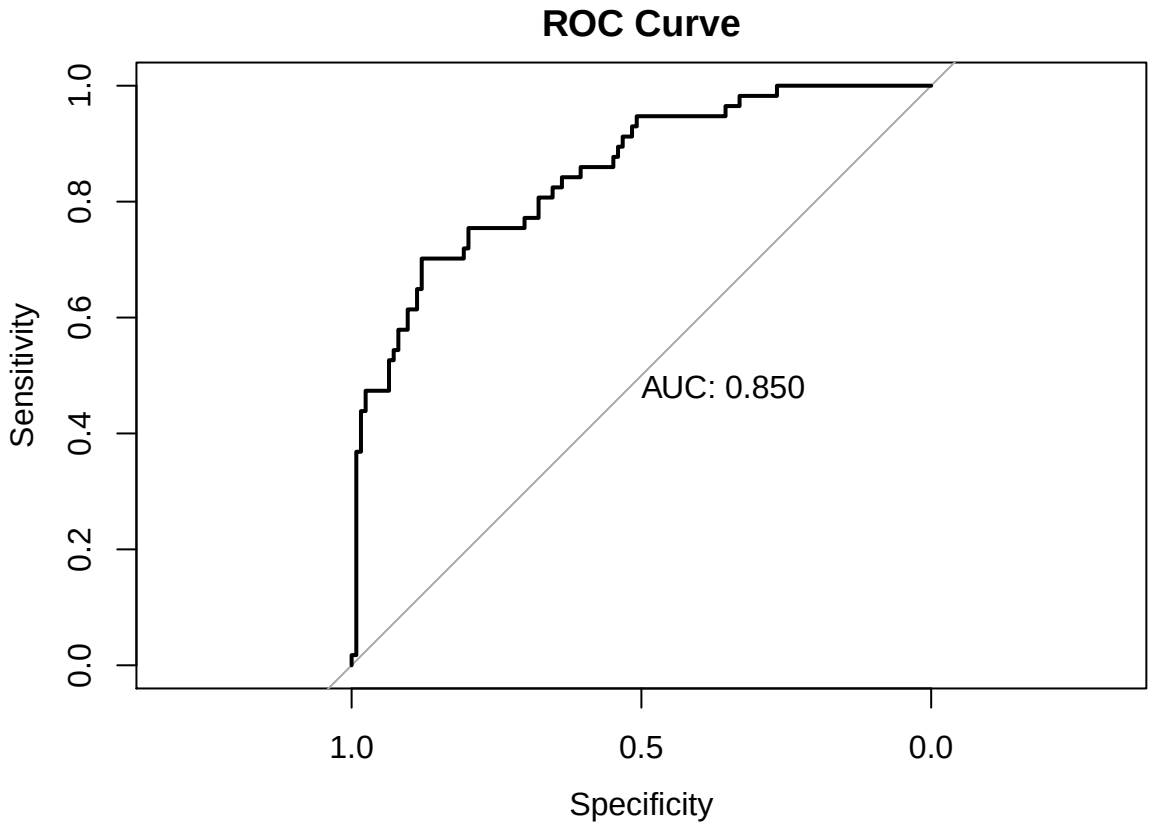
R pROC package

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?

```
# Using pRoc package

curve <- roc(response = data$class,
    predictor = data$scored.probability,
    plot = TRUE,
    print.auc = TRUE,
    main = "ROC Curve")
```

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

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



**The curve looks similar to the one with our functions. Also, the AUC is the same".

Github link for code

https://github.com/aaitelmouden/DATA621