DATA 621 HW 2

Tilon Bobb, Jian Quan Chen, Shamecca Marshall

2024-03-04

Overview

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.

```
library(tidyverse)
library(caret)
library(pROC)
library(ROCR)
```

1.

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

```
df <- read.csv("https://raw.githubusercontent.com/LeJQC/DATA-621-Group-2/main/HW2/classification-output
head(df)
```

```
##   pregnant glucose diastolic skinfold insulin  bmi pedigree age class
## 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
##   scored.class scored.probability
## 1            0         0.32845226
## 2            0         0.27319044
## 3            0         0.10966039
## 4            0         0.05599835
## 5            0         0.10049072
## 6            0         0.05515460
```

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?

```
confusion_matrix <- table(df$class, df$scored.class)

print(confusion_matrix)
```

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

The table() function here is used to count how many times each combination of actual and predicted classes occur. In this confusion matrix, the rows represent the actual `class` and the columns represent the predicted `class` for the observation. Looking at the first row, there were 119 instances of True Negatives and 5 instances of False Positives. In the second row, there are 30 instances of False Negatives and 27 instances of True Positives.

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.

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

```
calculate_accuracy <- function(df) {
  confusion_matrix <- table(df$class, df$scored.class)

  TP <- confusion_matrix[2, 2]
  TN <- confusion_matrix[1, 1]
  FP <- confusion_matrix[1, 2]
  FN <- confusion_matrix[2, 1]

  accuracy <- (TP + TN) / (TP + FP + TN + FN)

  return(accuracy)
}

accuracy <- calculate_accuracy(df)
print(accuracy)
```

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

The accuracy of the predictions is 0.8066.

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.

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

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

```
calculate_error_rate <- function(df) {
  confusion_matrix <- table(df$class, df$scored.class)

  TP <- confusion_matrix[2, 2]
  TN <- confusion_matrix[1, 1]
  FP <- confusion_matrix[1, 2]
  FN <- confusion_matrix[2, 1]

  error_rate <- (FP + FN) / (TP + FP + TN + FN)

  return(error_rate)
}

error_rate <- calculate_error_rate(df)
print(paste("Error Rate: ", error_rate))
```

```
## [1] "Error Rate:  0.193370165745856"
```

```
accuracy <- calculate_accuracy(df)
print(paste("Accuracy: ", accuracy))
```

```
## [1] "Accuracy:  0.806629834254144"
```

```
print(paste("Sum: ", accuracy + error_rate))
```

```
## [1] "Sum:  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.

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

```
calculate_precision <- function(df) {
  confusion_matrix <- table(df$class, df$scored.class)

  TP <- confusion_matrix[2, 2]
  FP <- confusion_matrix[1, 2]
```

```
  precision <- TP / (TP + FP)

  return(precision)
}

precision <- calculate_precision(df)
print(precision)
```

```
## [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.

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

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

sensitivity <- func_sensitivity(df)
print(sensitivity)
```

```
## [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.

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

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

specificity <- func_specificity(df)
print(specificity)
```

```
## [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

$$F1Score = \frac{2 * Precision * Sensitivity}{Precision + Sensitivity}$$

```
f1 <- function(df){
  ## Call the previously defined functions
  precision <- calculate_precision(df)
  sensitivity <- func_sensitivity(df)

  ## Calculate F1 score
  f1_score <- (2 * precision * sensitivity)/(precision + sensitivity)
  return(f1_score)
}
(f1_score <-  f1(df))
```

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

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 $0 < a < 1$ and $0 < b < 1$ then a b $< a$.)

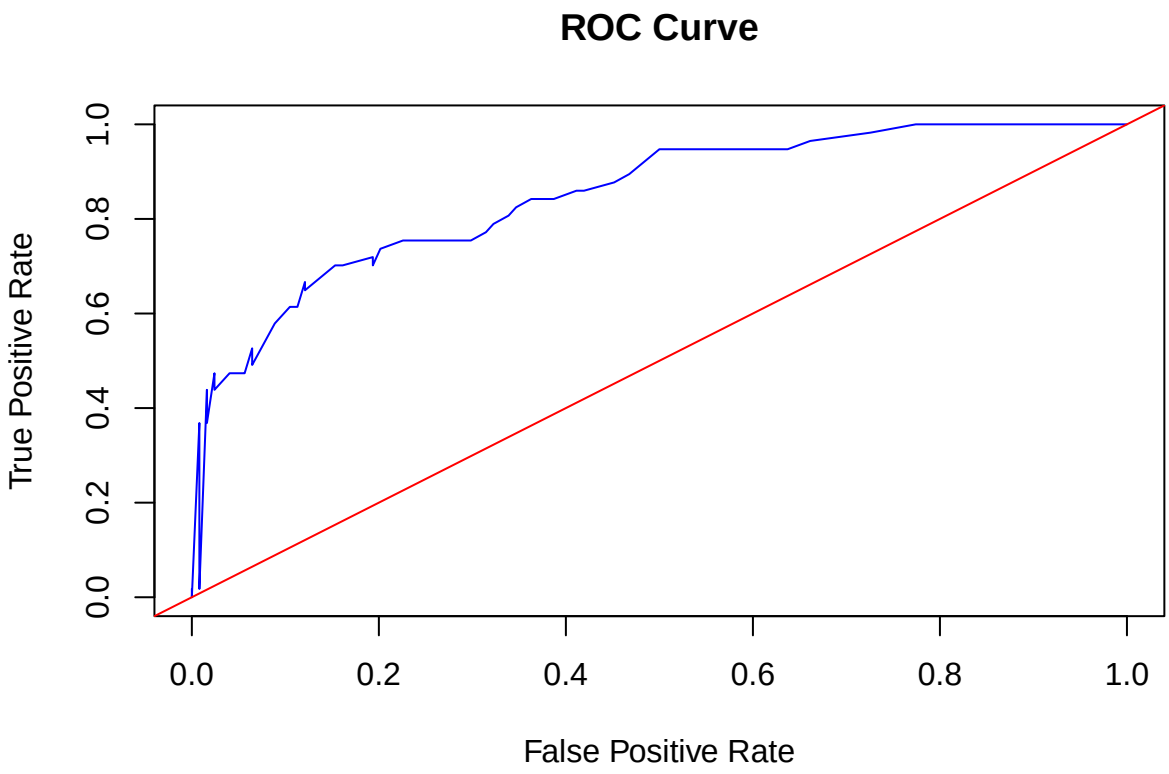
We determine the F1 score by considering both precision and sensitivity. As precision and sensitivity values range from 0 to 1, any computation involving values within this range will yield results within the same boundaries. Consequently, the F1 score will always fall between 0 and 1.

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.

```
generate_ROC_curve <- function(data) {
  # Sort the data
  data <- data[order(data$scored.probability, decreasing = TRUE), ]


  true_positive_rate <- numeric()
  false_positive_rate <- numeric()
  auc <- 0

  total_positive <- sum(data$class == 1)
  total_negative <- sum(data$class == 0)

  # Initialize variables for ROC curve points
```

```r
  roc_points <- data.frame(TPR = numeric(), FPR = numeric())

  for (threshold in seq(0, 1, by = 0.01)) {
    predicted_positive <- ifelse(data$scored.probability >= threshold, 1, 0)
    true_positive <- sum(predicted_positive == 1 & data$class == 1)
    false_positive <- sum(predicted_positive == 1 & data$class == 0)
    tpr <- true_positive / total_positive
    fpr <- false_positive / total_negative

    roc_points <- rbind(roc_points, cbind(TPR = tpr, FPR = fpr))
  }

  # Sort ROC curve points by ascending FPR
  roc_points <- roc_points[order(roc_points$FPR), ]

  # Calculate AUC using trapezoidal rule
  for (i in 1:(nrow(roc_points) - 1)) {
    auc <- auc + (roc_points$TPR[i] + roc_points$TPR[i + 1]) *
      (roc_points$FPR[i + 1] - roc_points$FPR[i]) / 2
  }

  # Plot ROC curve
  plot(roc_points$FPR, roc_points$TPR, type = "l", col = "blue",
       xlab = "False Positive Rate", ylab = "True Positive Rate", main = "ROC Curve")

  # Add diagonal line
  abline(0, 1, col = "red")

  # Return list with AUC value
  return(list(auc = auc))
}

set.seed(123)

result <- generate_ROC_curve(df)
```




```
#AUC value
result$auc
```

```
## [1] 0.8484012
```

11.

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

```
created_functions <- c(calculate_accuracy(df),
                       calculate_error_rate(df),
                       calculate_precision(df),
                       func_sensitivity(df),
                       func_specificity(df),
                       f1(df))

names(created_functions) <- c("Accuracy",
                              "Classification Error Rate",
                              "Precision",
                              "Sensitivity",
                              "Specificity",
                              "F1 Score")
```

```
knitr::kable(created_functions, col.names = "Classification Metrics")
```

	Classification Metrics
Accuracy	0.8066298
Classification Error Rate	0.1933702
Precision	0.8437500
Sensitivity	0.4736842
Specificity	0.9596774
F1 Score	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?

```
df$class <- factor(df$class, levels = c(0, 1))
predicted_class <- factor(ifelse(df$scored.probability >= 0.5, 1, 0), levels = c(0, 1))
cm <- confusionMatrix(df$class, predicted_class)
print(cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 119   5
##          1  30  27
##
##                Accuracy : 0.8066
##                  95% CI : (0.7415, 0.8615)
##     No Information Rate : 0.8232
##     P-Value [Acc > NIR] : 0.7559
##
##                   Kappa : 0.4916
##
##  Mcnemar's Test P-Value : 4.976e-05
##
##             Sensitivity : 0.7987
##             Specificity : 0.8438
##          Pos Pred Value : 0.9597
##          Neg Pred Value : 0.4737
##              Prevalence : 0.8232
##          Detection Rate : 0.6575
##    Detection Prevalence : 0.6851
##       Balanced Accuracy : 0.8212
##
##        'Positive' Class : 0
##
```

The calculated Accuracy using the confusionMatrix function is 0.81 while the calculated accuracy using the function made in question 3 is 0.81 which shows that both functions work the same in that aspect.

The confusion matrix function also had the value of 0.7987 for sensitivity while the custom function had the value of 0.4736842 in question 6 this is a big difference in that the built in function is saying that 79.87% of the actual positive cases were correctly identified by the model while the custom function is saying that 47.37% of the actual positive cases were correctly identified by the model.

The built in function had a sensitivity of 0.9596774 while the confusionMatrix function had a value of 0.8438 meaning that approximately 95.97% of the actual positive cases were correctly identified by the built-in function while 84.38% of the actual positive cases were correctly identified by the confusionMatrix function. And depending on what someone values, they would prefer the custom models results since it has a high sensitivity

```
specificity <- specificity(df$class, predicted_class)
print(specificity)
```

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

Again, we see that the custom function had a value of 47.37% while the specificity function had a value of 84.48%, meaning that the specificity function accurately identifies 84.48% of the positive cases while the custom function created in question 7 accurately identifies 47.37%, making the sensitivity function more favorable.

```
sensitivity <- sensitivity(df$class, predicted_class)
print(sensitivity)
```

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

In this comparison, we observed that the custom sensitivity function yielded a value of 0.8438 (84.38%), while the built-in function, done in question 6 returned a slightly lower value of 0.7986577. On the other hand, when we looked at specificity, the built-in function outperformed the custom one, achieving a higher score of 0.8448 (84.48%) compared to the custom function's 0.4737 (47.37%). This indicates that the built-in sensitivity function is more favorable as it accurately identifies a higher percentage of positive cases, contrasting with the custom function's weaker performance in this regard.

13.

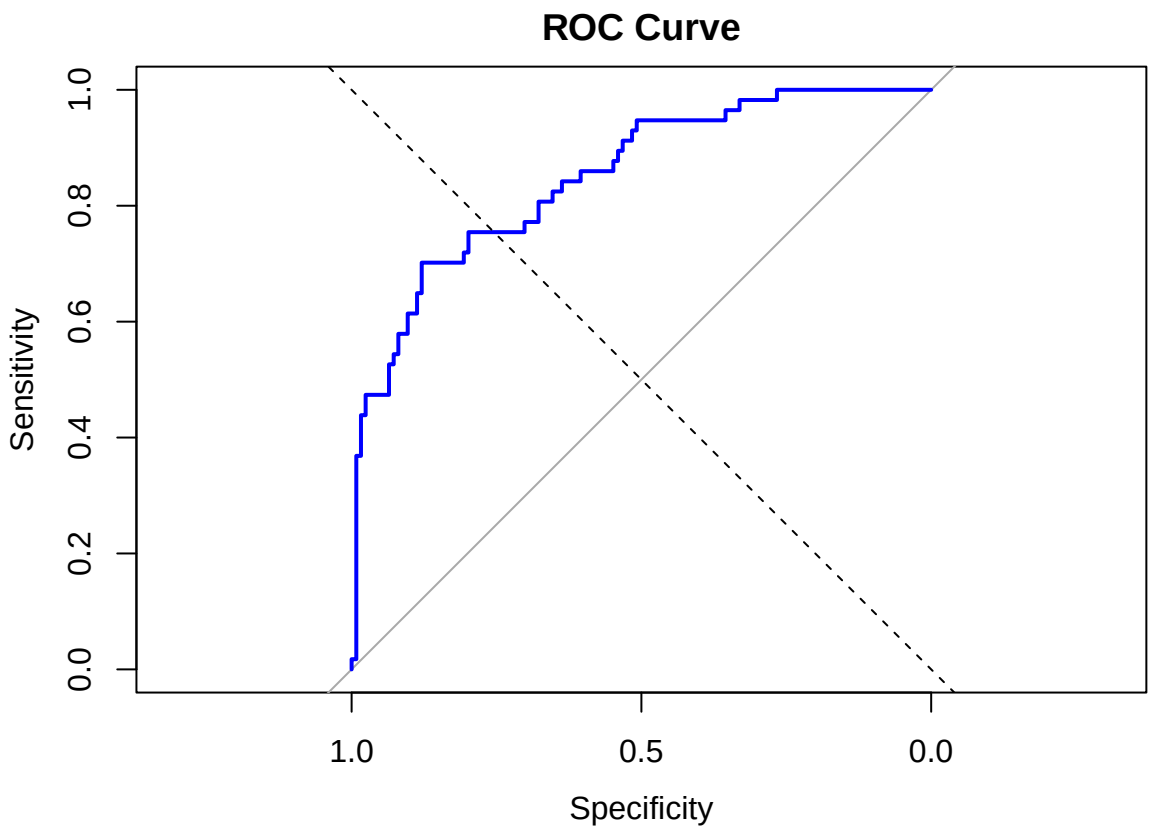
Investigate the pROC package. Use it to generate an ROC curve for the data set.How do the results compare with your own functions?

```
roc_curve <- roc(df$class, df$scored.probability)
```

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

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

```
# Plot ROC curve
plot(roc_curve, main = "ROC Curve", col = "blue", lwd = 2)

# Add diagonal reference line (random classifier)
abline(a = 0, b = 1, lty = 2)
```




```
# Calculate AUC
auc_score <- auc(roc_curve)
print(paste("AUC:", auc_score))
```

```
## [1] "AUC: 0.850311262026033"
```

Using the pROC package compared to writing my own function is different in that the pROC package made it very easy to implement the ROC curve on the data set. Also note that the AUC from the pROC package is .8503 compared to the functions .8484