Data621 - HW2

Group 4

Devin Teran, Atina Karim, Tom Hill, Amit Kapoor

3/21/2021

Contents

1. Download Dataset

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

```
dataset_df <- read.csv('https://raw.githubusercontent.com/hillt5/DATA_621/master/HW2/classification-outp
```

2. Confusion Matrix

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(dataset_df$class, dataset_df$scored.class)

dimnames(confusion_matrix) <- list(c('Not Diabetic', 'Diabetic'), c('Not Diabetic', 'Diabetic'))
names(dimnames(confusion_matrix) )<- c('Actual class', 'Predicted class')

confusion_matrix
```

```
##                Predicted class
## Actual class   Not Diabetic Diabetic
##   Not Diabetic          119        5
##   Diabetic               30       27
```

The rows represent the actual class, and columns the class predicted by the model. This dataset appears to be based on the Pima Indians dataset, which is being used to classify subjects as diabetic or not.

3. Accuracy

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

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

```
get_accuracy <- function(data_frame) {
TP <- sum((data_frame['class'] == 1)) - sum((data_frame['class'] == 1)[data_frame['scored.class'] == 0])
TN <- sum((data_frame['class'] == 0)) - sum((data_frame['class'] == 0)[data_frame['scored.class'] == 1])
denominator <- nrow(data_frame) #the denominator is the same as all observations
result <- (TP + TN)/denominator
return(100*round(result,3))
}
accuracy <- get_accuracy(dataset_df)
accuracy
```

```
## [1] 80.7
```

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

4. Classification Error Rate

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.

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

```
get_err_rate <- function(data_frame) {
FP <- sum((data_frame['class'] == 0)) - sum((data_frame['class'] == 0)[data_frame['scored.class'] == 0])
FN <- sum((data_frame['class'] == 1)) - sum((data_frame['class'] == 1)[data_frame['scored.class'] == 1])

denominator <- nrow(data_frame) #the denominator is the same as all observations

result <- (FP + FN)/denominator
```

```
return(100*round(result,3))
}
error_rate <- get_err_rate(dataset_df)
error_rate
```

```
## [1] 19.3
```

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

```
(accuracy + error_rate)/100
```

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

5. Precision

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

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

```
get_precision <- function(data_frame) {
TP <- sum((data_frame['class'] == 1)) - sum((data_frame['class'] == 1)[data_frame['scored.class'] == 0])
FP <- sum((data_frame['class'] == 0)) - sum((data_frame['class'] == 0)[data_frame['scored.class'] == 0])
result <- TP/(TP + FP)
return(100*round(result,3))

}

precision <- get_precision(dataset_df)

precision
```

```
## [1] 84.4
```

6. Sensitivity

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.

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

```
get_sensitivity <- function(data_frame) {
TP <- sum((data_frame['class'] == 1)) - sum((data_frame['class'] == 1)[data_frame['scored.class'] == 0])
FN <- sum((data_frame['class'] == 1)) - sum((data_frame['class'] == 1)[data_frame['scored.class'] == 1])
result <- TP/(TP + FN)
return(100*round(result,3))
}

sensitivity <- get_sensitivity(dataset_df)

sensitivity
```

```
## [1] 47.4
```

7. Specificity

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

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

```
get_specificity <- function(data_frame) {


TN <- sum((data_frame['class'] == 0)) - sum((data_frame['class'] == 0)[data_frame['scored.class'] == 1])
FP <- sum((data_frame['class'] == 0)) - sum((data_frame['class'] == 0)[data_frame['scored.class'] == 0])


result <- TN/(TN + FP)

return(100*round(result,3))

}

specificity <- get_specificity(dataset_df)

specificity
```

```
## [1] 96
```

8. F_1 Score

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.

$$\text{F}_1 \text{ Score} = \frac{2 \times \text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

```
get_f1 <- function(data_frame) {
TP <- sum((data_frame['class'] == 1)) - sum((data_frame['class'] == 1)[data_frame['scored.class'] == 0])
TN <- sum((data_frame['class'] == 0)) - sum((data_frame['class'] == 0)[data_frame['scored.class'] == 1])
FP <- sum((data_frame['class'] == 0)) - sum((data_frame['class'] == 0)[data_frame['scored.class'] == 0])
FN <- sum((data_frame['class'] == 1)) - sum((data_frame['class'] == 1)[data_frame['scored.class'] == 1])

sensitivity <- TP/(TP + FN)
precision <-  TP/(TP + FP)

result <- (2*precision*sensitivity)/(precision + sensitivity)

return(round(result,3))

}
```

```r
f1 <- get_f1(dataset_df)
f1
```

```
## [1] 0.607
```

9. Bounds on F_1 Score

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 ab<a)

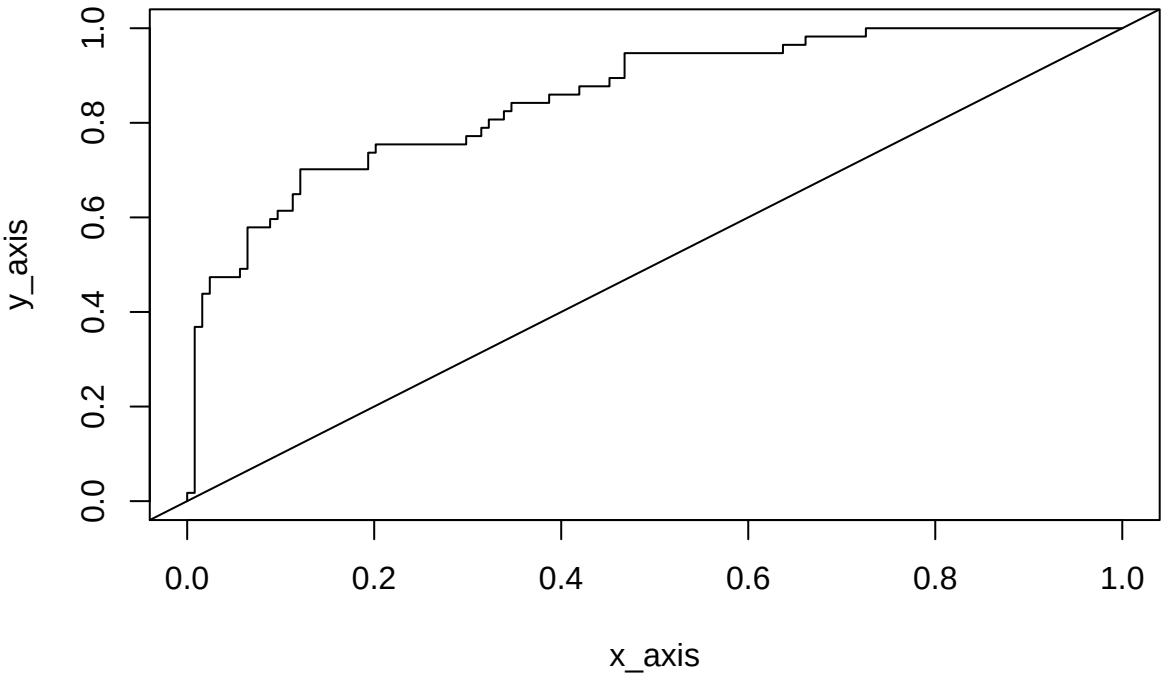
The minimum f1 score is zero, and the maximum is one. There is only one case in which f1 is equal to 1, which is when sensitivity and precision are both equal to 1. In this case, all true classes are detected. The f1 score is zero in cases when either precision or sensitivity are zero. A score of zero for either would indicate the classifier is missing all true classes, positive or negative. The f1 score is an average of precision and sensitivity, each having equal weights. Alternative weights could be considered when detection of true positives is more important, and vice versa.

10. ROC Curve

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
get_roc <- function(data_frame) {

  class <- data_frame['class']
  probability <- data_frame['scored.probability']

  threshold <- seq(0,1, 0.01)


  x_axis <- vector()
  y_axis <- vector()

  for (i in 1:length(threshold)) #iterate over classification thresholds

  {

    TP <- sum((probability >= threshold[i]) & (class == 1)) #threshold met and real classification = tr
    TN <- sum((probability < threshold[i]) & (class == 0)) #threshold not met and real classification =
    FP <- sum((probability >= threshold[i]) & (class == 0)) #threshold met and real classification = fa
    FN <- sum((probability < threshold[i]) & (class == 1)) #threshold not met but real classification

      FPR <- 1 - TN/(TN + FP) # 1- specificity
      TPR <- TP/(TP + FN)   #total positive rate


      x_axis[i] <- FPR
      y_axis[i] <- TPR

    }

  roc_graph <- plot(x_axis, y_axis,type = 's') + abline(0,1) #s-type graph is stair/step plot
```

```
  #Calculate AUC
  auc <- 0
  for (i in 1:length(x_axis)){
    if(i==length(x_axis)){
      auc <- auc
    }
    else if(y_axis[i] == 0){
      next
    }
    else{
      auc <- auc + ((x_axis[i]-x_axis[i+1])*y_axis[i])
    }
  }
  results <- list(c(auc),roc_graph)

  return(results)

}

get_roc(dataset_df)
```




```
## [[1]]
## [1] 0.8539898
##
## [[2]]
## integer(0)
```

11. Metrics

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

```
Metrics <- c('Accuracy','Classification Error Rate', 'Precision', 'Sensitivity','Specificity', 'F1')
Value<- round(c(get_accuracy(dataset_df), get_err_rate(dataset_df), get_precision(dataset_df), get_sens
dataset_df2 <- as.data.frame(cbind(Metrics, Value))
knitr::kable(dataset_df2)
```

Metrics	Value
Accuracy	80.7
Classification Error Rate	19.3
Precision	84.4
Sensitivity	47.4
Specificity	96
F1	0.607

12. Caret package

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.

The **confusionMatrix()** function puts the actual class in the columns and the predicted class in the rows, which is the opposite of the **table()** function. The results from the functions here match our results.

```
confusionMatrix(as.factor(dataset_df$scored.class),
                as.factor(dataset_df$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
##
```

```
# Precision
posPredValue(as.factor(dataset_df$scored.class),
             as.factor(dataset_df$class),
             positive = '1')
```

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

```
# Sensitivity
sensitivity(as.factor(dataset_df$scored.class),
             as.factor(dataset_df$class),
             positive = '1')
```

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

```
# specificity
specificity(as.factor(dataset_df$scored.class),
             as.factor(dataset_df$class),
             negative = '0')
```

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

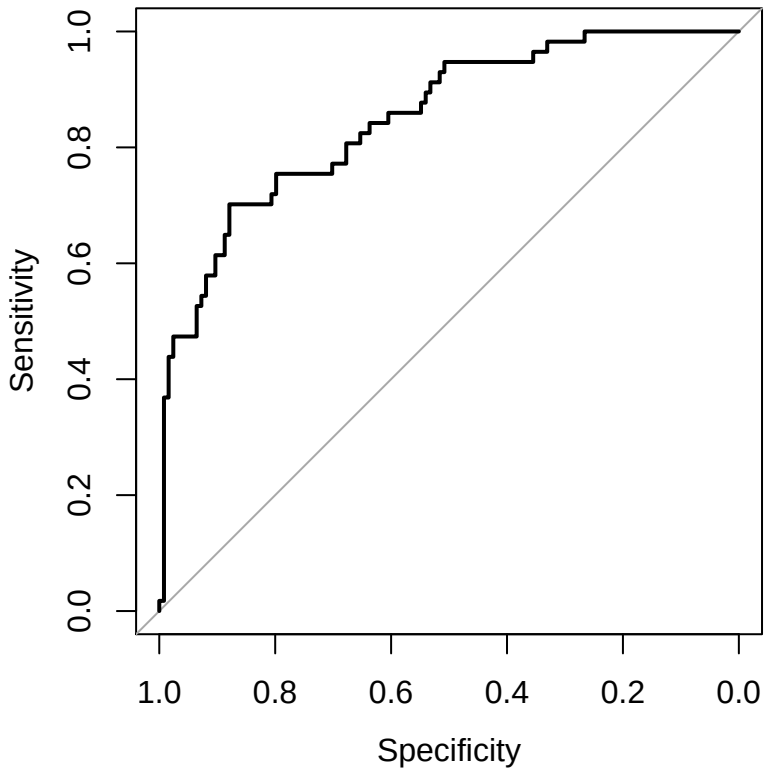
The results for the metrics are almost the same between the custom functions and the caret package.

13. pROC package

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

This function has the specificity plotted on the x-axis seen from 1.0 to 0.0. We plotted (1-specificity) on the x-axis. This doesn't change the plotted points, only the x-axis label. The AUC calculated here, 0.8503, which is very close to our value for AUC, 0.8539898.

```
roc <- roc(dataset_df, class, scored.probability)
par(pty="s")
plot(roc)
```




```
auc(roc)
```

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