Homework 2: Classification Metrics

Omar Pineda, Jeffrey Littlejohn, Sergio Ortega Cruz, Chester Poon, Simon Ustoyev

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

The dataset was downloaded and we load this data into a dataframe with the below code:

```
df <- read.csv('classification-output-data.csv')
```

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?

The below code creates our confusion matrix:

```
library(dplyr)
df1 <- df %>%
  select(scored.class,class)
cmatrix = table(df1)
cmatrix

##              class
## scored.class   0   1
##             0 119  30
##             1   5  27
```

The class columns are the actual/observed, while the scored.class rows are the predicted.

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.

The below code coverts our matrix into a dataframe:

```
cmatrix <- as.data.frame.matrix(cmatrix)
knitr::kable(cmatrix)
```

	0	1
0	119	30
1	5	27

And now we create our function that takes in the confusion matrix as a dataframe and returns the accuracy of the predictions:

```r
acc <- function(x) {
  (x[1,1] + x[2,2])/sum(x)
}
acc(cmatrix)

## [1] 0.8066298
```

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.

The function that returns a classification error rate is below:

```r
class_err <- function(x) {
  (x[1,2] + x[2,1])/sum(x)
}
class_err(cmatrix)

## [1] 0.1933702
```

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

When we add the accuracy with the error rate we get 1:

```r
class_err(cmatrix) + acc(cmatrix)

## [1] 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.

The function that returns the precision:

```r
prec <- function(x) {
  x[1,1]/(x[1,1] + x[1,2])
}
prec(cmatrix)

## [1] 0.7986577
```

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:

```r
sensitivity <- function(x) {
  x[1,1]/(x[1,1] + x[2,1])
}
sensitivity(cmatrix)

## [1] 0.9596774
```

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:

```r
specificity <- function(x) {
  x[2,2]/(x[2,2] + x[1,2])
}
specificity(cmatrix)

## [1] 0.4736842
```

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.

We can actually use some of our previously created functions to create this function.

F1-Score:

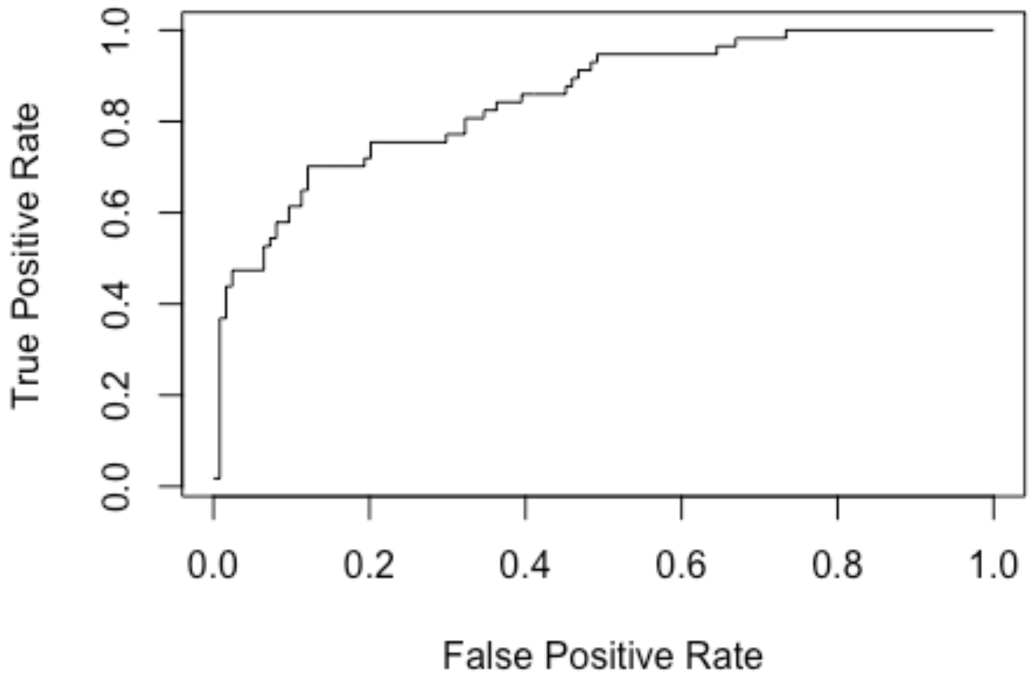
```r
f1 <- function(x) {
  (2*prec(x)*sensitivity(x))/(prec(x) + sensitivity(x))
}
f1(cmatrix)

## [1] 0.8717949
```

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 ⍰⍰ < ⍰.)

Because both precision and sensitivity will always be between 0 and 1, if you use the upper and lower bounds of precision and sensivity, you will always get a number between 0 and 1 from the formula for F1 score.

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(actual, scores){
  actual <- actual[order(scores, decreasing=TRUE)]
  plot(cumsum(!actual)/sum(!actual),
       cumsum(actual)/sum(actual),
       type='l',
       xlab = "False Positive Rate",
       ylab = "True Positive Rate")
  f <- approxfun(cumsum(!actual)/sum(!actual),
                 cumsum(actual)/sum(actual))
  C <- integrate(f,
                  min(cumsum(actual)/sum(actual)),
                  max(cumsum(actual)/sum(actual)))$value
  paste("AUC = ",C)
}

roc(df$class,df$scored.probability)
```



```
## [1] "AUC =  0.846480454012587"
```

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","Precision","Sensitivity","Specificity","F1-Score")

values <- c(acc(cmatrix),
            class_err(cmatrix),
            prec(cmatrix),
            sensitivity(cmatrix),
            specificity(cmatrix),
            f1(cmatrix))

metric_df <- data.frame(metric,values)
names(metric_df) <- c("Metric","Values")
```

```
knitr::kable(metric_df)
```

Metric	Values
Accuracy	0.8066298
Classification Error	0.1933702
Precision	0.7986577
Sensitivity	0.9596774
Specificity	0.4736842
F1-Score	0.8717949

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?

```
library(caret)
confusionMatrix(factor(df1$scored.class),factor(df1$class))

## 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
## 
```

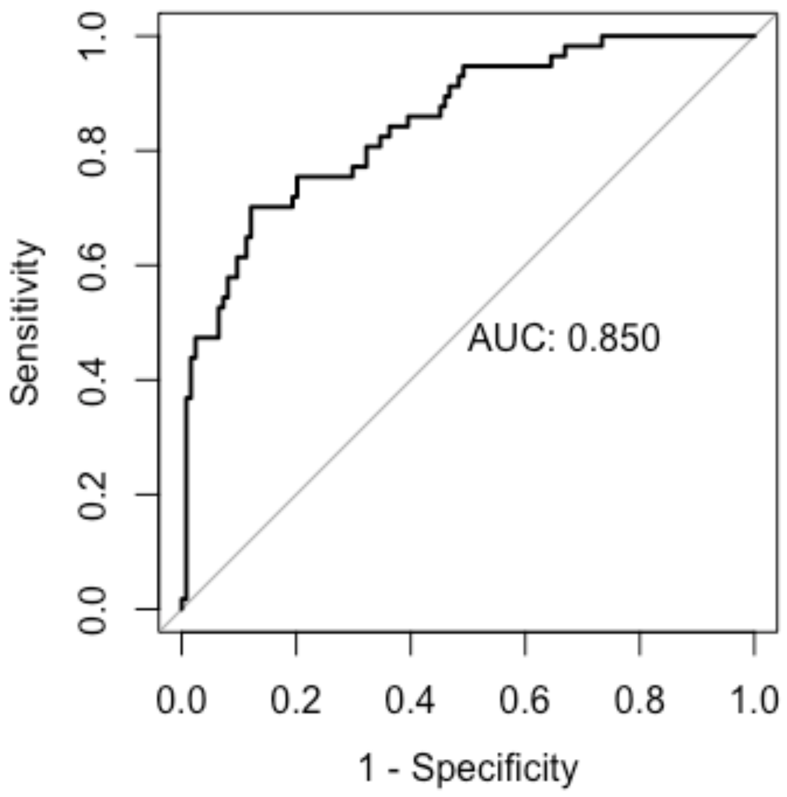
The values above from the `confusionMatrix` return identical values as the manually created functions from earlier.

The below two functions from the `caret` package returns the same sensitivity and specificity outputs as the `confusionMatrix` function above.

```
caret::sensitivity(factor(df1$scored.class),factor(df1$class))

## [1] 0.9596774

caret::specificity(factor(df1$scored.class),factor(df1$class))

## [1] 0.4736842
```

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?

```
library(pROC)
par(pty="s")
pROC::roc(df$class,df$scored.probability,
          plot=TRUE,
          legacy.axes=TRUE,
          print.auc=TRUE)
```



```
##
## Call:
## roc.default(response = df$class, predictor = df$scored.probability,
plot = TRUE, legacy.axes = TRUE, print.auc = TRUE)
```

```
##
## Data: df$scored.probability in 124 controls (df$class 0) < 57 cases
(df$class 1).
## Area under the curve: 0.8503
```

The ROC curves appear identical and the AUC values are approximately the same.