Data 621 HW2

Team 2

03/13/2019

Contents

Overview

Task: Upon following the instructions below, use your created R functions and the other packages to generate the classification metrics for the provided data set. A write-up of your solutions submitted in PDF format.

Instructions

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.

Dependencies

This assignment requires the following dependencies to complete Steps 12 and 13:

```
require(caret)
require(pROC)
```

Steps

Step 1

Step 1 requires us to download the classification output dataset file. We accomplished this by reading the data.

```
data<-read.csv('data.csv')
```

Step 2

Step 2 requires us to use the `table()` function to get the raw confusion matrix for the scored dataset. The provided 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

The confusion matrix puts the measured value on x-axis and the actual value on the y-axis. The rows in the table below represent the predictions from the `scored.class` column, whereas the columns depict the actual values from the `class` variable.

In this matrix, 0 represents measured or expected falsehood where as 1 represents an expected/measured truth. We can see from the results that the major diagonal contains most of our data. That is, the model generally classifies things correctly. There are, however, 35 classifieds data points.

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

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

Step 3

Step 3 requires us to 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 function we created calculates accuracy from the confusion matrix for the `class` and `scored.class` variables. It measures the number of correct classifications and divides that sum by the total number of classifications.

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

```
accuracy <- function(df) {
mtx = table(df$class,df$scored.class)
head(mtx)
TN=mtx[1,1]
FP=mtx[1,2]
FN=mtx[2,1]
TP=mtx[2,2]

return((TP+TN)/(TP+FP+TN+FN))
}
```

Step 4

Step 4 requires us to 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. We expect both our functions for accuracy and an error rate to return a sum of one.

Below is the error rate function, which measures the incorrect classifications divided by the total.

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

```
error <- function(df) {
mtx = table(df$class,df$scored.class)
TN=mtx[1,1]
FP=mtx[1,2]
FN=mtx[2,1]
TP=mtx[2,2]

return((FP+FN)/(TP+FP+TN+FN))

}
```

As expected, the accuracy and error rates add up to 1.

```
accuracy(data) + error(data)
```

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

Step 5

Step 5 requires us to create a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the precision of the predictions.

The precisions function below measures how many detected positives were correctly classified.

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

```
precision <- function(df) {
mtx = table(df$class,df$scored.class)
FP=mtx[1,2]
TP=mtx[2,2]


return((TP)/(TP+FP))

}
```

Step 6

Step 6 requires us to create 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 measures the number of true positive classifications divided by the number of positives in the population.

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

```
sensitivity <- function(df) {
mtx = table(df$class,df$scored.class)
TP=mtx[2,2]
FN=mtx[2,1]

return((TP)/(TP+FN))

}
```

Step 7

Step 7 requires us to create a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the specificity of the predictions.

Specificity is the negative equivalent of the sensitivity. This value measures number of correctly classified negatives in relation to the total number of negatives in the classified set.

```
specificity <- function(df) {
mtx = table(df$class,df$scored.class)
TN=mtx[1,1]
FP=mtx[1,2]
FN=mtx[2,1]
TP=mtx[2,2]

return((TN)/(TN+FP))

}
```

Step 8

Step 8 requires us to create a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the F_1 score of the predictions. We know that the F_1 score is calculated as follows:

$$F_1 = \frac{2 * \text{precision} * \text{sensitivity}}{\text{precision} + \text{sensitivity}}$$

F_1 score is a hybrid score that attempts to weigh the recall(1/specificity) and precision in one metric using the harmonic average. Simply put the F1 score is:

$$F_1 = \frac{(\text{recall}^{-1} + \text{precision}^{-1})^{-1}}{2}$$

```
f1 <- function(df) {
mtx = table(df$class,df$scored.class)
TN=mtx[1,1]
FP=mtx[1,2]
FN=mtx[2,1]
TP=mtx[2,2]

return (2*TP/(2*TP+FN+FP))

}
```

Step 9

Step 9 asks us to evaluate the bounds on the F1 score and show that the F1 score will always be between 0 and 1.

Since precision and sensitivity can only be on the interval $[0, 1]$ the denominator will always be at least as large as the numerator. That is to say that

$$a + b \geq a * b$$

when a, b are between $[0, 1]$.

That means our fraction will be at most 1. However, we must also consider the 2. For that we look at the harmonic form referenced in Step 8 .

Here it is easy to see that even if we have a large value for both precision and recall (1), the numerator can be at most 2, giving us a maximum score of 1. We can verify that with a simulation of 10000 pairs in R.

```
a <- runif(10000,0,1)
b <- runif(10000,0,1)
f1.sim <- (2*a*b/(a+b))
max(f1.sim)
```

```
## [1] 0.9944263
```

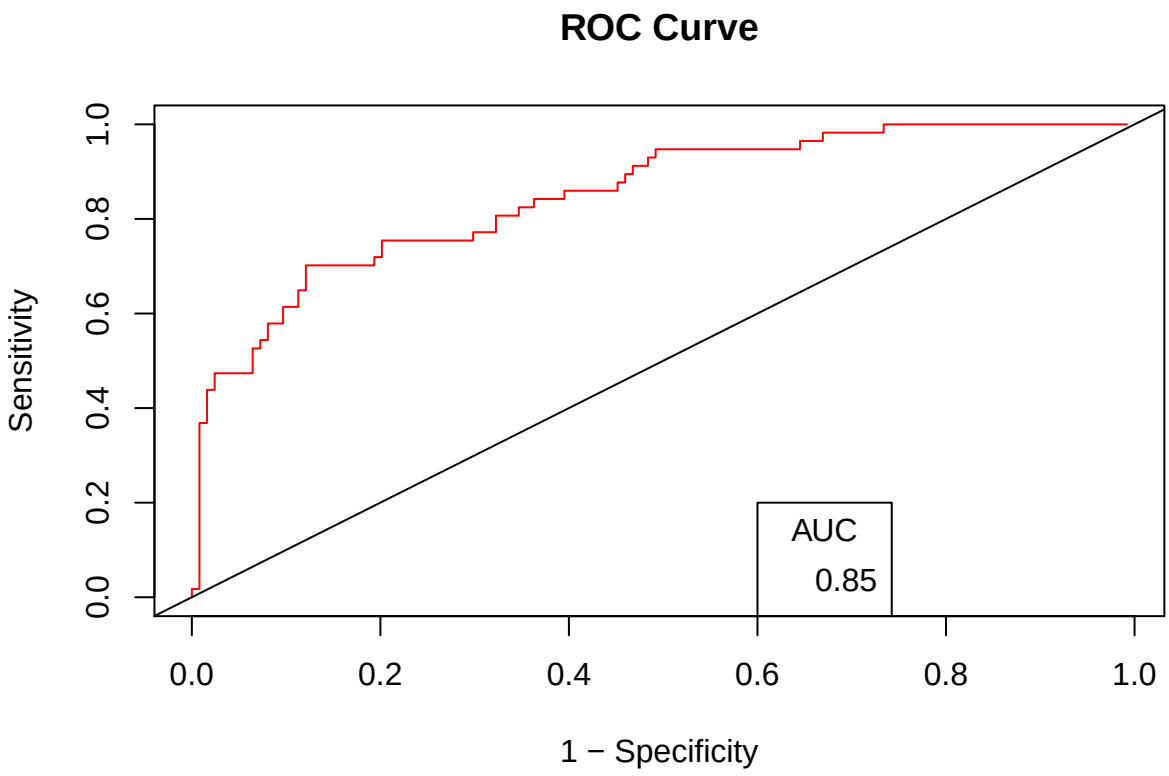
Step 10

Step 10 requires us to create 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).

The function below returns the following requirements: a list that includes the plot of the ROC curve and a vector that contains the calculated area under the curve (AUC).

Per instructions, we used a sequence of thresholds ranging from 0 to 1 at 0.01 intervals to complete this step.

```
data$class<- as.integer(data$class)

roc_fun<-function(class, probability){

  actuals <- class[order(probability)]

  sens <- (sum(actuals) - cumsum(actuals))/sum(actuals)
  spec <- cumsum(!actuals)/sum(!actuals)
  (auc <- round(sum(spec*diff(c(0, 1 - sens))),3))

  plot(1 - spec, sens, type = "l", col = "red", main ="ROC Curve", ylab = "Sensitivity", xlab = "1 - Sp
  abline(c(0,0),c(1,1))
  legend(.6,.2,auc,title = "AUC")

}

roc_fun(data$class, data$scored.probability)
```



Step 11

In Step 11, we apply our custom functions to the provided classification output dataset to produce all of the classification metrics discussed above.

```
accuracy(data)
sensitivity(data)
specificity(data)
error(data)
precision(data)
f1(data)
```

The following contains the output from these functions:

accuracy	sensitivity	specificity	error	precision	f1
0.8066	0.4737	0.9597	0.1934	0.8438	0.6067

Step 12

In Step 12, we explored the `caret` package to verify our calculations and convert the `scored.class` and `class` vectors to factors.

The output from this package matches the results achieved in our functions above.

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

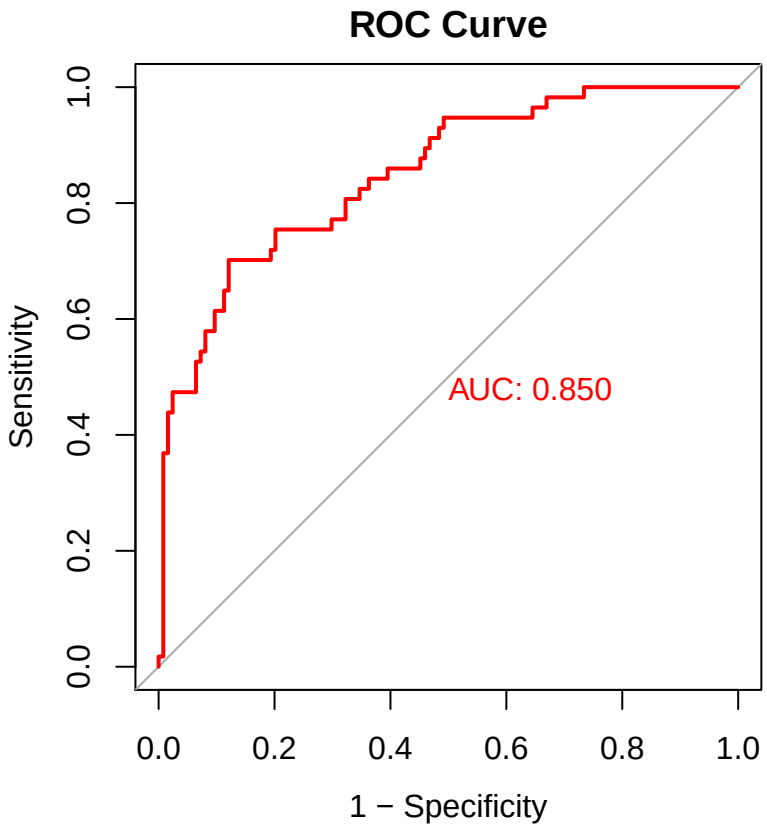
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 119  30
##          1   5  27
##
##                Accuracy : 0.8066
##                  95% CI : (0.7415, 0.8615)
##     No Information Rate : 0.6851
##     P-Value [Acc > NIR] : 0.0001712
##
##                   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
##
```

Step 13

Similarly, we also investigated the `pROC` package and used its functions to verify our results from the ROC curve.

As you can see below, the `roc` function and our custom `roc_function` from step 10 both achieve the same result.

```
par(pty = "s")
roc(class ~ scored.probability, data, smooth=FALSE, plot = TRUE, print.auc=TRUE,legacy.axes =TRUE,  col=
```




```
##
## Call:
## roc.formula(formula = class ~ scored.probability, data = data,     smooth = FALSE, plot = TRUE, print
##
## Data: scored.probability in 124 controls (class 0) < 57 cases (class 1).
## Area under the curve: 0.8503
```