Homework #2 Classification Metrics

Douglas Barley, Ethan Haley, Isabel Magnus, John Mazon, Vinayak Kamath

9/28/2021

DATA 621 – Business Analytics and Data Mining

Overview

In this homework assignment, we will work through various classification metrics. We will create functions in R to carry out the various calculations. We will also investigate some functions in packages that will let us obtain the equivalent results. Finally, we will create graphical output that also can be used to evaluate the output of classification models, such as binary logistic regression.

Step 1.

We start by downloading the classification output data set (attached in Blackboard to the assignment).

```
d.f = read.csv('https://raw.githubusercontent.com/ebhtra/msds-621/main/HW2/classification-output-data.c
head(d.f)
```

```
##   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
```

```
colnames(d.f)
```

```
##  [1] "pregnant"           "glucose"            "diastolic"
##  [4] "skinfold"           "insulin"            "bmi"
##  [7] "pedigree"           "age"                "class"
## [10] "scored.class"       "scored.probability"
```

Step 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

We'll use the **table()** function to get the raw confusion matrix for this scored dataset.

```
data1 <- d.f %>%
  select(class, scored.class, scored.probability)
head(data1)
```

```
##   class scored.class scored.probability
## 1     0            0         0.32845226
## 2     0            0         0.27319044
## 3     1            0         0.10966039
## 4     0            0         0.05599835
## 5     0            0         0.10049072
## 6     0            0         0.05515460
```

A confusion matrix shows the number of correct and incorrect predictions made by the classification model compared to the actual outcomes (target value) in the data. The matrix is NxN, where N is the number of target values (classes). Performance of such models is commonly evaluated using the data in the matrix. The following table displays a 2x2 confusion matrix for two classes (Positive and Negative).

```
cm_df <- dplyr::select(data1, scored.class, class)
table(cm_df)
```

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

Step 3.

Next we'll 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 = $\frac{TP+TN}{TP+FP+TN+FN}$

```
get_accuracy <- function(df, actual, predicted  ) {
  df_table <- table(select(df, actual, predicted))
  TP <- df_table[2,2]
  TN <- df_table[1,1]
  FN <- df_table[2,1]
  FP <- df_table[1,2]

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

Step 4.

Then we'll 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 $= \frac{FP+FN}{TP+FP+TN+FN}$

```r
get_err_rate <- function(df, actual, predicted  ) {
  df_table <- table(select(df, actual, predicted))
  TP <- df_table[2,2]
  TN <- df_table[1,1]
  FN <- df_table[2,1]
  FP <- df_table[1,2]

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

Verify that we get an accuracy and an error rate that sums to one:

```r
get_accuracy(d.f, "class", "scored.class") + get_err_rate(d.f, "class", "scored.class")
```

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

Step 5.

Next we 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 $= \frac{TP}{TP+FP}$

```r
get_precision <- function(df, actual, predicted  ) {
  df_table <- table(select(df, actual, predicted))
  TP <- df_table[2,2]
  FP <- df_table[1,2]

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

Step 6.

Here we 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 $= \frac{TP}{TP+FN}$

```r
get_sensitivity <- function(df, actual, predicted  ) {
  df_table <- table(select(df, actual, predicted))
  TP <- df_table[2,2]
  FN <- df_table[2,1]

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

Step 7.

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

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

```r
get_specificity <- function(df, actual, predicted  ) {
  df_table <- table(select(df, actual, predicted))
  TN <- df_table[1,1]
  FP <- df_table[1,2]

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

Step 8.

Now that we've built the components for it, we can write a function that returns the F1 score of the predictions.

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

```r
get_F1_score <- function(df, actual, predicted  ) {
  precision <- get_precision(df, actual, predicted)
  sensitivity <- get_sensitivity(df, actual, predicted)

  return ((2*precision*sensitivity) / (precision+sensitivity))
}

#testing the function
get_F1_score(d.f, "class", "scored.class")
```

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

Step 9.

Before we move on, let's consider a question that was asked: What are the bounds on the F1 score? We'll show that the F1 score will always be between 0 and 1.

$F1\ Score = \frac{2 \times Precision \times Sensitivity}{Precision + Sensitivity}$, where Precision $= \frac{TP}{TP+FP}$ and Sensitivity $= \frac{TP}{TP+FN}$

All of TP, FP, and FN are non-negative integers by definition, so we don't have to consider negative values here.

If TP equals 0, then Precision and Sensitivity are either 0 or undefined, and in either case, the F1 Score is undefined.

All remaining possibilities involve TP being a positive integer.

The $F1\ Score = \frac{2 \times \frac{TP}{TP+FP} \times \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$

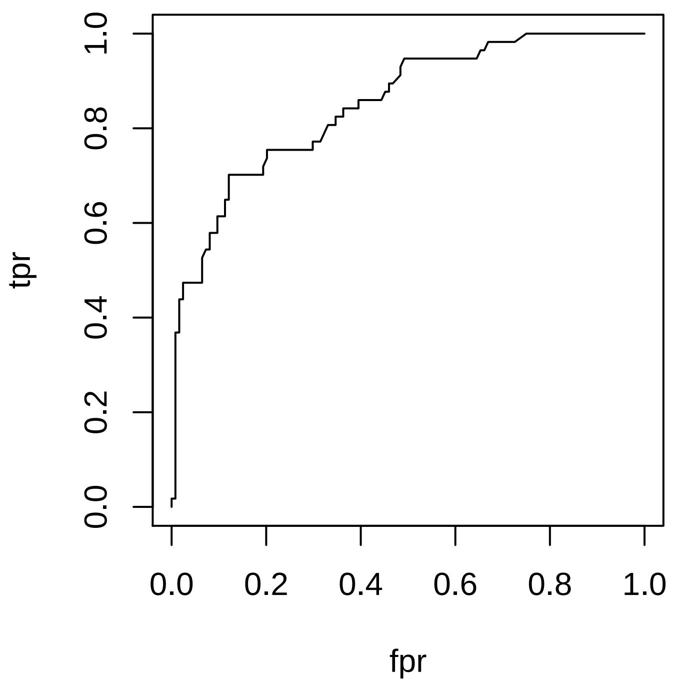
If both FN and FP are equal to 0, then we get the upper bound on F1, which is 1.
The lower bound is if TP is equal to 1, in which case we get $\frac{2}{2+FN+FP}$, which approaches 0 as FN and/or FP approach infinity. Thus, $0 <$ F1 ≤ 1 if F1 is defined. Note that a tighter lower bound is provided by the number of predictions, n: $\frac{2}{1+n}$, since $FN + FP \leq n - 1$.

Step 10.

Now we'll move on to writing 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). Our function returns a list that includes the plot of the ROC curve and the calculated area under the curve (AUC). We used a sequence of thresholds ranging from 0 to 1 at 0.001 intervals, for our AUC calculation.

```r
get_roc_curve <- function(df, actual, predicted_probability ) {

  tpr = c()  # collect true positive rates at each threshold
  fpr = c()  # and false positive rates

  for (threshold in seq(1, 0, -.001) ) {
    predictions = 1 * (df[predicted_probability] > threshold)

    tpr <- c(tpr, sum(predictions * df[actual] == 1) / sum(df[actual] == 1) )
    fpr <- c(fpr, sum(predictions - df[actual] == 1) / sum(df[actual] == 0) )

  }

  par(pty='s')  # square plot is best
  plot_roc <- plot(fpr, tpr, type='l', asp=1)

  auc = 0
  for (i in 2:length(fpr)){
    if (fpr[i] > fpr[i-1]){  # every time fpr changes, calculate new area using trapezoid area
      auc = auc + (fpr[i] - fpr[i-1]) * (tpr[i] + tpr[i-1]) / 2
    }
  }
  return(c(plot_roc, paste("AUC using trapezoid approximation:", round(auc, 4))))
}
```

```r
#test our function
get_roc_curve(data1 , "class", "scored.probability")
```



```
## [1] "AUC using trapezoid approximation: 0.8504"
```

Step 11.

Now we'll use our created R functions and the provided classification output data set to produce all of the classification metrics discussed above.

```r
paste("Accurancy : " , round( get_accuracy(data1, "class", "scored.class") ,4) )
```

```
## [1] "Accurancy :  0.8066"
```

```r
paste("Error Rate: " , round( get_err_rate(data1,   "class", "scored.class") ,4) )
```

```
## [1] "Error Rate:  0.1934"
```

```r
paste("Precision : " , round( get_precision(d.f,   "class", "scored.class") ,4) )
```

```
## [1] "Precision :  0.8438"
```

```r
paste("Sensitivity : " , round( get_sensitivity(data1,   "class", "scored.class") ,4) )
```

```
## [1] "Sensitivity :  0.4737"
```

```r
paste("Specificity : " , round( get_specificity(data1,   "class", "scored.class") ,4) )
```

```
## [1] "Specificity :  0.9597"
```

```r
paste("F1 Score : " , round( get_F1_score(data1,   "class", "scored.class") ,4) )
```

```
## [1] "F1 Score :  0.6067"
```

Step 12.

To check the results from above, we'll use the caret package and compare its results to ours.

```r
confusionMatrix(table(cm_df), positive = '1')
```

```
## Confusion Matrix and Statistics
##
##              class
## scored.class   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
##
```

We end up with the same values for our metrics.

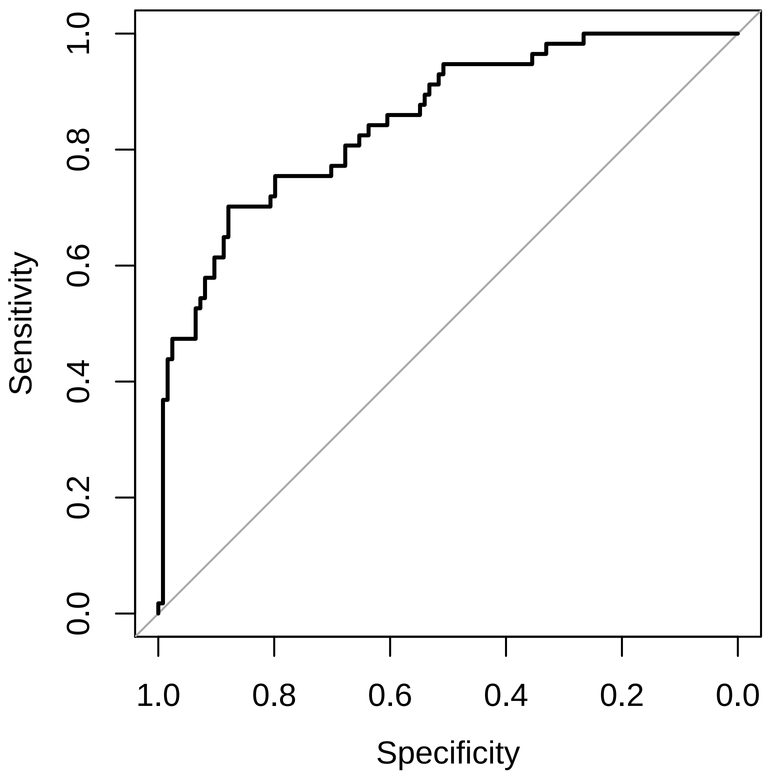
Step 13.

And finally, for verifying the correctness of our ROC-AUC function, we'll use the pROC package.

```
 #calculate AUC and plot ROC
par(pty='s')  # square plot is best
plot(roc(d.f$class, d.f$scored.probability))
```

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

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



```
auc(d.f$class, d.f$scored.probability)
```

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

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

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

Our AUC value of .8504 is almost identical to the pROC value of .8503.