Data621 - Assignment 2

John DeBlase, Sekhar Mekala, Sonya Hong

October 5, 2016

In this homework assignment, we were to work through various classification metrics and create functions in R to carry out the calculations. The output of these functions were then compared to functions in the caret and pROC packages.

Building the Confusion Matrix

Steps 1 and 2

For this assignment, the variables we will be working with are class, scored class, and scored probabilty. The class is the actual binary classification of the observation. The scored class is the prediction based on a threshold of 0.5. The scored probability is the predicted probability of a successful observation. The head() printout below shows the three variables of interest. Each row represents the class, scored class and scored probability for an observation in the dataset.

```
df = read.csv("classification-output-data.csv")
head(df[,9:11], 10)
```

```
##    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
## 7      0            0         0.10711542
## 8      0            0         0.45994744
## 9      0            0         0.11702368
## 10     0            0         0.31536320
```

The first step in calculating the classification metrics is to produce a raw confusion matrix from the class and scored class variables.

This is done in base R using the table() function and the process is encapsulated in a function for convenience in future calculations.

```
make_conf_matrix = function(df){

    # classes as factors
    actual = as.factor(df$class)
    predicted = as.factor(df$scored.class)

    # build confusion matrix
    conf_matrix = table(predicted, actual)
    conf_matrix
}

make_conf_matrix(df)
```

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

This matrix contains true positives in the lower right corner (1,1) and true negatives in the upper left corner (0,0). The lower left corner indicates the number of false postives and the top right corner the number of false negatives.

The rows in the confusion matrix represent the predicted class, and columns represent the actual class (the reference class). We will use the below format of the confusion matrix in this document.

	Actual Negative	Actual Positive
Predicted Neg	TN	FN
Predicted Pos.	FP	TP

Calculating Metrics

Using the confusion matrix we can calculate the metrics for accuracy, classification error rate, precision, sensitivity, specificity, and the model's F1 score. Each function takes the dataframe with actual and predicted classifications identified and creates a confusion matrix using the above utility method.

Each cell in the table is assigned to its particular variable, TP for true positive, TN for true negative, FP for false positive and FN for false negative. The appropriate metric is calculated based on the formula given in the assignment handout.

Step 3

The following formula will be used to calculate the *Accuracy*

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

The R code to calculate the *Accuracy* is given below:

```r
accuracy = function (df){

    # build confusion matrix
    conf_matrix = make_conf_matrix(df)

    # assign formula values
    TP = conf_matrix[4]
    TN = conf_matrix[1]
    FP = conf_matrix[2]
    FN = conf_matrix[3]

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

Step 4

The following formula will be used to calculate the *Classification Error Rate*

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

The R code to calculate the *Classification Error Rate* is given below:

```
class_err_rate = function(df){

    # build confusion matrix
    conf_matrix = make_conf_matrix(df)

    # assign formula values
    TP = conf_matrix[4]
    TN = conf_matrix[1]
    FP = conf_matrix[2]
    FN = conf_matrix[3]

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

Before proceeding further we need to check that the error rate and accuracy sum to one.

```
accuracy(df) + class_err_rate(df) == 1
```

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

Step 5

The *Precision* metric is calculated using the following formula.

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

The R code to calculate the *Precision* is give below:

```
precision = function(df){

   # build confusion matrix
    conf_matrix = make_conf_matrix(df)

    # assign formula values
    TP = conf_matrix[4]
    TN = conf_matrix[1]
    FP = conf_matrix[2]
    FN = conf_matrix[3]

    TP/(TP + FP)

}
```

Step 6

The following formula will be used to calculate the *Sensitivity*

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

The R code to calculate the *Sensitivity* is given below:

```r
sensitivity = function(df){

     # build confusion matrix
    conf_matrix = make_conf_matrix(df)

    # assign formula values
    TP = conf_matrix[4]
    TN = conf_matrix[1]
    FP = conf_matrix[2]
    FN = conf_matrix[3]

    TP/(TP + FN)
}
```

Step 7

The following formula will be used to calculate the *Specificity*

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

The R code to calculate the *Specificity* is given below:

```r
specificity = function(df){

    # build confusion matrix
    conf_matrix = make_conf_matrix(df)

    # assign formula values
    TP = conf_matrix[4]
    TN = conf_matrix[1]
    FP = conf_matrix[2]
    FN = conf_matrix[3]

    TN/(TN + FP)
}
```

Step 8

The F1 score is an overall measure of the model's accuracy based on sensitivity and precision. This function uses the above methods to calculate the score.

$$F1 = \frac{2 * Precision * Sensitivity}{Precision + Sensitivity}$$

```r
F1 = function(df){
    # use above functions to return F1 score
    (2 * precision(df) * sensitivity(df))/(precision(df) + sensitivity(df))
}
```

Step 9

What are the bounds on the F1 score? Show that the F1 score will always be between 0 and 1.

The formula given for F1 is:

$$F_1 = \frac{2 * Precision * Sensitivity}{Precision + Sensitivity}$$

The formula for precision is given as:

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

The formula for sensitivity is given as:

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

Let us find the maximum and minimum values of sensitivity and precision to determine the bounds of F_1: The TP, FP, and FN always assume integer values which are always ≥ 0. The precision and sensitivity will equal zero when TP=0 and both FP $\geq$ 1 and FN $\geq$ 1. Hence the minimum value of precision and sensitivity is 0. Precision will equal 1, when TP $>$ 0 and FP = 0. The Sensitivity will equal 1 when TP $>$ 0 and FN =0. Hence, both precision and sensitivity can have a maximum value of 1 (since TP, FP, and FN are always $\geq$ 0).

But when both TP and FN are 0, then sensitivity will lead to undefined value. Similarly, the precision will be undefined when TP=0 and FP=0. Let us find how sensitivity function behaves in the viscinity of TP=FN=0.

Mathematically, if we substitute TP=FN=0 in the $Sensitivity = \frac{TP}{TP+FN}$, we will get an undefined form 0/0. Let us apply the following limit, and find if the limit exists when $TP \to FN \to 0$.

$$\lim_{(TP,FN)\to 0} \frac{TP}{TP + FN}$$

We evaluate the above limit in 2 different cases:

Case-1

We will evaluate the limit when $FN \to 0$ and then applying $TP \to 0$.

$$\lim_{(TP,FN)\to 0} \frac{TP}{TP + FN} = \lim_{TP\to 0} \lim_{FN\to 0} \frac{TP}{TP + FN} = \lim_{TP\to 0} \frac{TP}{TP + 0}$$
$$\lim_{TP\to 0} 1 = 1$$

Case-2

We will evaluate the limit when $TP \to 0$ and then applying $FN \to 0$.

$$\lim_{(TP,FN)\to 0} \frac{TP}{TP + FN} = \lim_{FN\to 0} \lim_{TP\to 0} \frac{TP}{TP + FN} = \lim_{FN\to 0} \frac{0}{FN + 0}$$
$$\lim_{FN\to 0} 0 = 0$$

The limit has resulted in different values in *case-1* and *case-2*. So the limit is assuming different values depending on the order of applying $TP \to 0$ and $FN \to 0$. Hence we can conclude that sensitivity does not exist when both TP and FN are 0.

Similar reasoning can be applied for precision, and we can conclude that the precision is undefined when both TP and FP are 0.

Hence Sensitivity will always have a value in the interval [0,1], and is undefined when TP=FN=0. Similarly precision will always have a value in the interval [0,1], and is undefined when TP=FP=0.

Finding the maximum and minimum value of F1:

The formula given for F1 is:

$$F_1 = \frac{2 * Precision * Sensitivity}{Precision + Sensitivity}$$

For maximum at precision and sensitvity = 1, the F1 score will equal 1 based on the above formula:

$$F_1 = \frac{2 * Precision * Sensitivity}{Precision + Sensitivity}$$

$$F_1 = \frac{2 * 1 * 1}{1 + 1} = 1$$

When precision=sensitivity=0, the F_1 will become undefined (0/0 form). We can check if the value of F_1 is having a definite value (when precision=sensitivity=0) by applying the following limit:

$$\lim_{s,p \to 0} = \frac{2 * p * s}{p + s}$$

where s = sensitivity and p = precision.

We will evaluate the above limit in 3 different cases.

Case-1

Let us evaluate the limit, by applying $p \to 0$, followed by applying $s \to 0$.

$$\lim_{s,p \to 0} \frac{2 * p * s}{p + s} = \lim_{s \to 0} \lim_{p \to 0} \frac{2 * p * s}{p + s} = \lim_{s \to 0} 0 = 0$$

Case-2

Let us evaluate the limit, by applying $s \to 0$, followed by applying $p \to 0$.

$$\lim_{s,p \to 0} \frac{2 * p * s}{p + s} = \lim_{p \to 0} \lim_{s \to 0} \frac{2 * p * s}{p + s} = \lim_{p \to 0} 0 = 0$$

Case-3

Let us evaluate the limit, by applying $s \to 0$ and $p \to 0$ simultaneously. As $s \to 0$ and $p \to 0$ simultaneously, we can assume that $s \to p \to 0$. So, let us substitute $s = p$ in the limit and evaluate the limit at $p \to 0$.

$$\lim_{s,p \to 0} \frac{2 * p * s}{p + s} = \lim_{p \to 0} \frac{2 * p * p}{p + p} = \lim_{p \to 0} p = 0$$

In all the three cases the limit evaluates to 0. Hence we can conclude that the F_1 function exists when both sensitivity and precision assume 0, and the F_1 value will be 0, in such scenario.

Therefore the range of F_1 is always between [0,1].

Step 10

Computing an ROC curve

The myROC function takes the initial dataframe and returns a list containing the ROC plot and AUC computed using the trapezoidal method. The sensitivity and specificity are computed at threshold levels which are in 0.01 increments from 0 to 1.

Special utility functions designed to handle edge cases are used to calculate both the sensitivity and the specificity.

```
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
##
## The following object is masked _by_ '.GlobalEnv':
##
##     msleep
```

```
# specialized sensitivity and specificity functions
# catch non-2x2 table and return 0
sensitivity_roc = function(df) {

    t = table(df[,"scored.class"],df[,"class"])
    tryCatch(t["1","1"]/sum(t[,"1"]),error = function(e) 0)
}


specificity_roc = function(df){

    t = table(df[,"scored.class"],df[,"class"])
    tryCatch(t["0","0"]/sum(t[,"0"]),error = function(e) 0)
}

myROC = function(df){

    class = as.vector(df[,"class"])
    scored.probability = as.vector(df[,"scored.probability"])

    temp = data.frame(class, scored.probability)
    sen = vector(length=100)
    spe = vector(length=100)

    #Create a threshold vector, with 0.01 interval size
    threshold = seq(from=0,to=1,by=0.01)

    #Calculate the sensitivity and specificity at various
    #threshold values
    for(i in 1:100) {
        temp$scored.class = 0  # start with zero
        # if thresh is greater change to one
        temp$scored.class[temp$scored.probability >= threshold[i]] = 1
```

```r
        sen[i] = sensitivity_roc(temp)
        spe[i] = specificity_roc(temp)
    }

    #Prepare the plot variable using ggplot()
    plot = ggplot(data.frame(False_Positives = 1-spe, Sensitivity=sen),
                  aes(x=False_Positives,y=Sensitivity)) +
        geom_point() +
        geom_path() +
        geom_abline(slope=1,intercept=0,linetype = 2, color='red')+
        labs(title="ROC", x="False Positive Rate")

    #Using trapezoidal rule to find the area under the ROC
    height = (sen[-1]+sen[-length(sen)])/2
    width = -diff(1-spe)
    AUC = sum(width*height)

    return(list(plot=plot,AUC=AUC))

}
```

Step 11

Calculate all metrics, ROC curve and AUC for the dataset. The initial metrics are wrapped in a function that returns a labelled list of the measurements.

```r
classification_model_eval = function(df){

    return(list(
        accuracy = accuracy(df),
        class_err_rate = class_err_rate(df),
        precision = precision(df),
        sensitivity = sensitivity(df),
        specificity = specificity(df),
        F1_score = F1(df)
    ))

}

classification_model_eval(df)
```

```
## $accuracy
## [1] 0.8066298
##
## $class_err_rate
## [1] 0.1933702
##
## $precision
## [1] 0.84375
##
## $sensitivity
## [1] 0.4736842
```

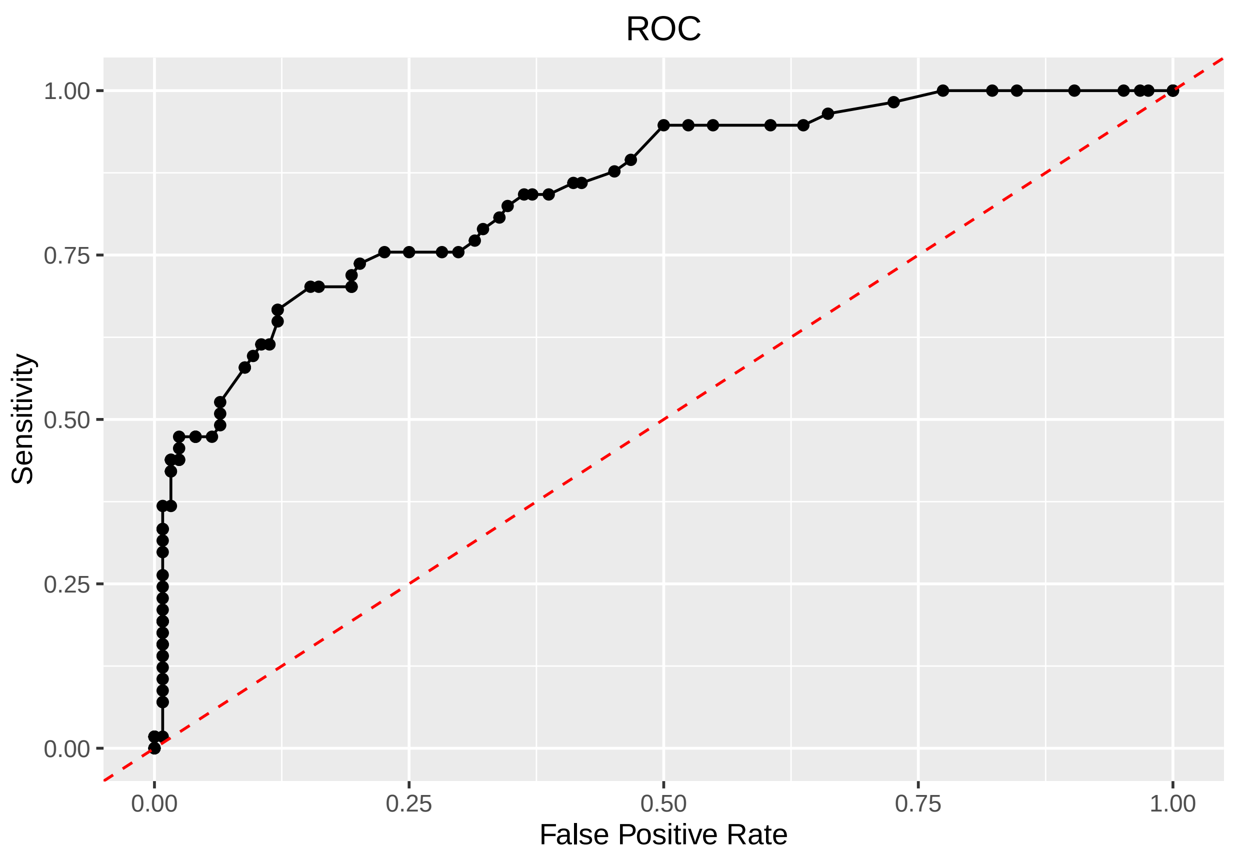
```
##
## $specificity
## [1] 0.9596774
##
## $F1_score
## [1] 0.6067416
```

According to the handmade functions:

- Accuracy = 0.8066298
- Classification Error Rate = 0.1933702
- Precision = 0.84375
- Sensitivity = 0.4736842
- Specificity = 0.9596774
- F1 score = 0.6067416

```r
myROC(df)
```

```
## $plot
```



```
##
## $AUC
## [1] 0.8488964
```

The ROC plot shows the typical logarithmic curve and an AUC value of 0.8488964.

Diagnostics with *caret* and *pROC*

Step 11

First we will check the results of our confusion matrix with results from equivalent functions int the **caret** package.

```
library(caret)

conf_matrix = confusionMatrix(data=df$scored.class,reference=df$class,positive="1")

conf_matrix
```

```
## 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
##
```

```
caret::specificity(conf_matrix$table, positive="1",negative="0")
```

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

```
caret::sensitivity(conf_matrix$table,positive="1",negative="0")
```

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

The results of confusionMatrix(), sensitivity() and specificity() from the caret package are equivalent to the results our hand-made calculations.

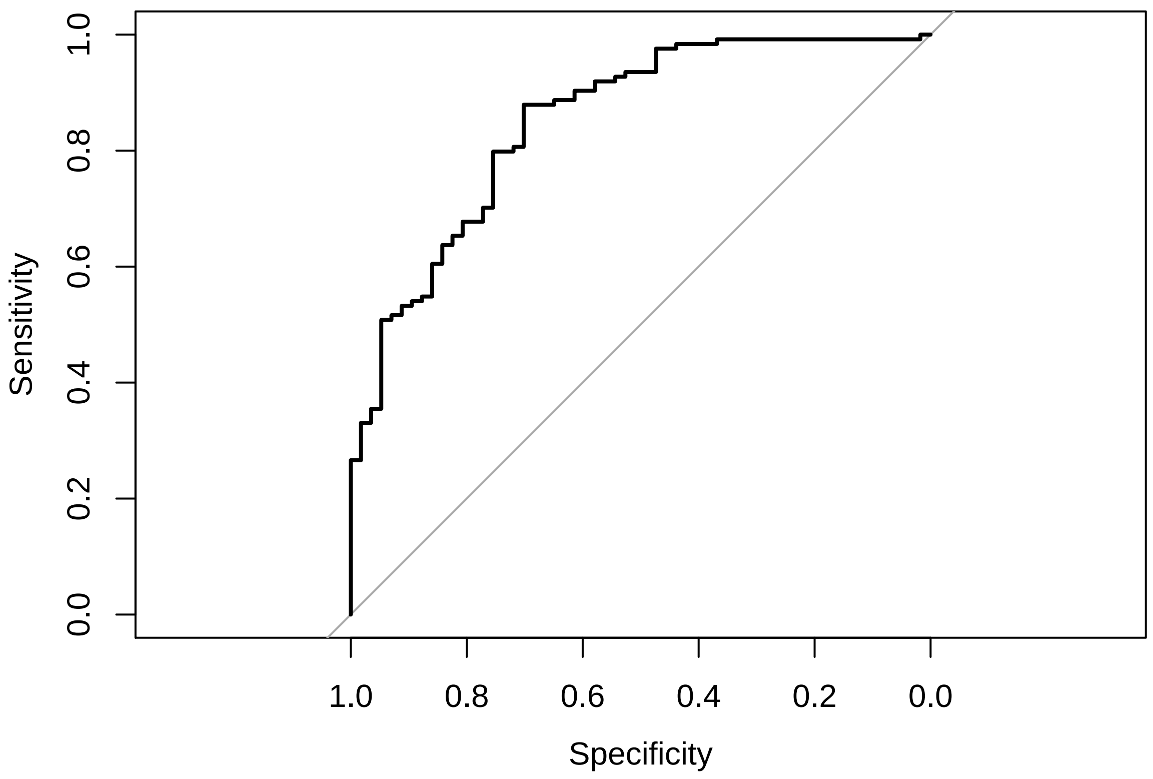
Step 12

Next we will compare our handmade plot and AUC calculation with those from the pROC package.

```r
library(pROC)

roc_obj = roc(response=df$class,predictor=df$scored.probability,
              levels=rev(levels(as.factor(df$class))))

plot.roc(roc_obj)
```



```
##
## Call:
## roc.default(response = df$class, predictor = df$scored.probability,     levels = rev(levels(as.facto
##
## Data: df$scored.probability in 57 controls (df$class 1) > 124 cases (df$class 0).
## Area under the curve: 0.8503
```

We can see that the AUC of 0.8503 is very close to our area calculation of 0.8488964. The curve generated by plot.roc() is also nearly identical to our ggplot.