Andrea Bruckner
Predict 422, Sec 56

Individual Project #2

Introduction

In this report I am presenting several functions I created in R to carry out the calculations of various classification metrics. I am also exploring some existing functions in a couple packages that produce the same classification metrics. I am using a data set containing 181 observations and 11 variables, but I am focusing on just three variables for this report: class, scored.class, and scored.probability.

Confusion Matrix

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

In the confusion matrix above, the rows represent the actual class (variable “class”), while the columns represent the predicted class (variable “scored.class”). This confusion matrix serves at the basis for the functions I created in R.

Functions

Accuracy

I created the following function to calculate accuracy. As evident in the code, the confusion matrix above is an essential part of this function. When I ran the function, I input the entire data set (“data.all”) as df, “class” as actual, and “scored.class” as predicted. I used the same inputs for each function that required df, actual, and predicted. I have placed my R code in Appendix A for reference. This function resulted in the value 0.8066298.

```
accuracy <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP

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

Classification Error Rate

I created the following function to calculate the classification error rate. It resulted in 0.1933702. By adding the values from the accuracy and class.error.rate functions, I verified they sum to 1.

```
class.error.rate <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
```

```
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP

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

Precision

I created the following function to calculate the precision. It resulted in 0.84375.

```
precision <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP

  TP/(TP + FP)
}
```

Sensitivity

I created the following function to calculate the sensitivity. It resulted in 0.4736842.

```
sensitivity <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP

  TP/(TP + FN)
}
```

Specificity

I created the following function to calculate the specificity. It resulted in 0.9596774.

```
specificity <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP

  TN/(TN + FP)
}
```

Andrea Bruckner
Predict 422, Sec 56

F1 Score

I created the following function to calculate the F1 score. It resulted in 0.6067416.

```
f1.score <- function(df, actual, predicted){
  (2*precision(df, actual, predicted)*sensitivity(df, actual, predicted))/(precision(df, actual,
predicted) + sensitivity(df, actual, predicted))
}
```

The bounds of the F1 score are 0 and 1. As shown in the function above, the calculation for the F1 Score requires the precision and sensitivity metrics. Both precision and sensitivity will always be between the values 0 and 1 because they represent a portion of data and therefore cannot be negative values nor exceed 1. For example, precision and sensitivity equal 0 when the True Positive (TP) value is 0 since TP is in the numerator. To recall, Precision = TP/(TP + FP) and Sensitivity = TP/(TP + FN). When TP is 1 and FP is 0, precision equals 1. When TP is 1 and FN is 0, sensitivity equals 1. If TP = 2 and FP = 2, precision equals 0.50. If TP = 4 and FN = 6, sensitivity equals 0.40. Because TP is in both the numerator and denominator for the equations for precision and sensitivity, these metrics will always be between 0 and 1. Similarly, because the F1 Score calculation has precision and sensitivity in both the numerator and denominator, the numerator and denominator are proportional to each other. Even if both precision and sensitivity are maximized to 1, the F1 score will only be 1 (i.e., (2*1*1/(1+1) = 2/2 = 1).

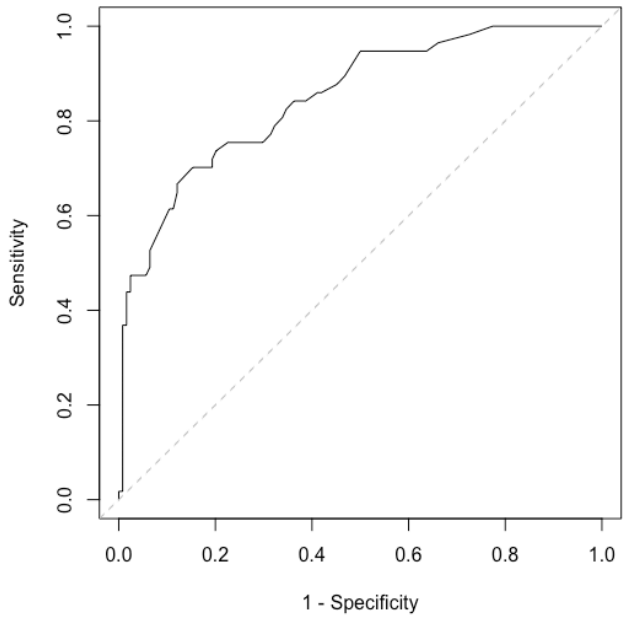
ROC Curve and AUC

I created the following function to produce a ROC curve and calculate the AUC.

```
my.roc <- function(df, actual, probability){
  roc.curve=function(t,print=FALSE){
    p=probability
    a=actual
    Pt=(p>t)*1
    FP=sum((Pt==1)*(a==0))/sum(a==0) # false positive
    TP=sum((Pt==1)*(a==1))/sum(a==1) # true positive
    if(print==TRUE){
      print(table(Observed=a, Predicted=Pt))
    }
    vect=c(FP,TP)
    names(vect)=c("FPR","TPR")
    return(vect)
  }
  threshold = 0.5
  roc.curve(threshold, print=TRUE)

  ROC.curve=Vectorize(roc.curve)

  M.ROC=ROC.curve(seq(0,1,by=.01))
  plot(M.ROC[1,],M.ROC[2,],type="l",xlab = "1 - Specificity", ylab = "Sensitivity")
  abline(0,1, col="gray", lty=2)

my.auc<-function(df, actual, probability){
```

```
    N=length(probability)
    N_pos=sum(actual)
    df=data.frame(out=actual, prob=probability)
    df=df[order(-df$prob),]
    df$above=(1:N)-cumsum(df$out)
    return(1-sum(df$above*df$out)/(N_pos*(N-N_pos)))
  }
  my.auc(df, actual, probability)
}
```

The my.roc function above produces the following plot and calculates an AUC of 0.8503113.



The high AUC value indicates that the classifier does a pretty good job separating the two classes, 0 and 1. The gray dashed line represents a random classifier with an AUC value of 0.50. While the my.roc function shows the classifier does a decent job predicting the class, a higher AUC value closer to 1 would indicate a better classifier.

Summary of Classification Metrics

Classification Metric	Value
Accuracy	0.8066298
Classification Error Rate	0.1933702
Precision	0.8437500
Sensitivity	0.4736842
Specificity	0.9596774
F1 Score	0.6067416

From the table above, we can see that the classes were identified correctly approximately 80.7 percent of the time (as indicated by Accuracy) and misidentified approximately 19.3 percent of the time (as indicated by Classification Error Rate). The Precision value indicates that when the classifier predicted a class of 1, it was correct approximately 84.4 percent of the time. The Sensitivity value indicates that the classifier predicted a class of 1 approximately 47.4 percent of the time the class equaled 1, while the Specificity value indicates that the classifier predicted a class of 0 approximately 96.0 percent of the time the class equaled 0. The F1 Score is approximately 60.7 percent, which indicates that the classifier does a mediocre job at balancing Precision and Sensitivity. A higher F1 Score would indicate that the classifier does a good job at maximizing both Precision and Sensitivity.

Comparison of Functions

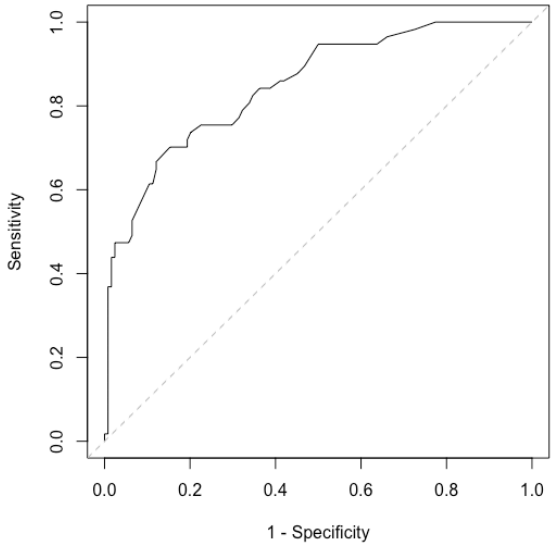
In addition to creating my own functions, I explored existing functions for these classification metrics in the caret and pROC packages.

Functions from caret Package

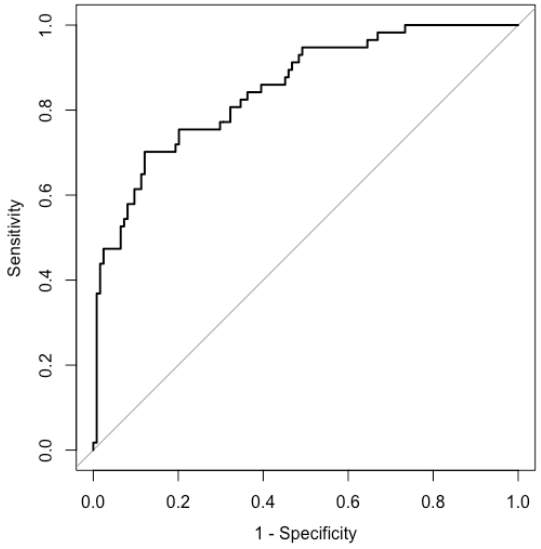
When I used the confusionMatrix, sensitivity, and specificity functions, I discovered that they resulted in the same classification metric values *only if* I specified that class 1 was "positive" while class 0 was "negative." Per the R documentation, "If there are only two factor levels, the first level will be used as the 'positive' result." Using the function's defaults results in different values for Precision, Sensitivity, Specificity, and the F1 Score since the function is treating 0s as "positive" and 1s as "negative."

Functions from pROC Package

When I used the plot.roc function, it produced a ROC curve nearly identical the plot my function created. As evident in the plots below, my function resulted in a plot with a smoother line while the plot.roc function has a more jagged line. The plot.roc function also calculated an AUC value of 0.8503, which is equivalent to the AUC value my function calculated (0.8503113) if we round to the same decimal place.



My ROC Curve Function



The ROC Curve Function from pROC

Conclusion

The functions created and explored in this report demonstrate that there are numerous ways to calculate classification metrics. While the results revealed that the classifier has a pretty good accuracy of 80.7 percent, the other metrics indicated that the classifier can definitely improve. Creating my own functions allowed me to “get under the hood” of the classification metrics, but I much prefer to use the existing metrics if possible. I am curious about the functions in the two packages I explored and will definitely experiment with them more later.

Andrea Bruckner
Predict 422, Sec 56

Appendix A: R Code

```
# Individual Project 2
# Andrea Bruckner
# Predict 422, Sec 56

# 1
# Load the csv
data.all <-
read.csv(file.path("/Users/annie/Desktop/Northwestern/PREDICT_422/Individual_Project_02","cl
assification-output-data.csv"),sep=",",header=TRUE)
attach(data.all)

# Get overview of data
summary(data.all) # no missing data
str(data.all) # all int or numeric
head(data.all)
class(data.all) # data.frame
names(data.all)
nrow(data.all) # 181 rows
ncol(data.all) # 11 variables

# 2
table(class,scored.class)

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

# rows = actual class (class)
# columns = predicted class (scored.class)

# left column = NEGATIVES
# right column = POSITIVES

# upper left = True Negative  | Upper right = False Positive
# lower left = False Negative | lower right = True Positive

# The diagonal elements of the confusion matrix indicate correct predictions,
# while the off-diagonals represent incorrect predictions. (pg 172/440)

# Set up for functions -- If I include this code within each function, the function would be more
self-contained.
c.matrix <- data.frame(table(class,scored.class))
c.matrix
str(c.matrix) # column 3 = int
TN <- c.matrix[1,3] # 119, TN
FN <- c.matrix[2,3] # 30, FN
FP <- c.matrix[3,3] # 5, FP
```

```
TP <- c.matrix[4,3] # 27, TP

# 3 -- Accuracy = (TP + TN)/(TP + FP + TN + FN)
accuracy <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP

  (TP + TN)/(TP + FP + TN + FN)
}
accuracy(data.all,class,scored.class) # 0.8066298

# 4 -- Classification Error Rate = (FP + FN)/(TP + FP + TN + FN)
class.error.rate <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP

  (FP + FN)/(TP + FP + TN + FN)
}
class.error.rate(data.all,class,scored.class) # 0.1933702

# Verify that you get an accuracy and an error rate that sums to one.
accuracy.out <- accuracy(data.all,class,scored.class)
class.error.rate.out <- class.error.rate(data.all,class,scored.class)
accuracy.out + class.error.rate.out # 1

# 5 -- Precision = TP/(TP + FP)
precision <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP

  TP/(TP + FP)
}
precision(data.all,class,scored.class) # 0.84375

# 6 -- Sensitivity = TP/(TP + FN)       ## aka True Positive Rate (TPR)
sensitivity <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP
```

```
  TP/(TP + FN)
}
sensitivity(data.all, class,scored.class) # 0.4736842

# 7 -- Specificity = TN/(TN + FP)       ## aka True Negative Rate (TNR)
specificity <- function(df, actual, predicted){
  c.matrix <- data.frame(table(actual, predicted))
  TN <- c.matrix[1,3] # 119, TN
  FN <- c.matrix[2,3] # 30, FN
  FP <- c.matrix[3,3] # 5, FP
  TP <- c.matrix[4,3] # 27, TP

  TN/(TN + FP)
}
specificity(data.all, class,scored.class) # 0.9596774

# 8 -- F1 Score = (2*precision*sensitivity)/(precision + sensitivity)
f1.score <- function(df, actual, predicted){
  (2*precision(df, actual, predicted)*sensitivity(df, actual, predicted))/(precision(df, actual,
predicted) + sensitivity(df, actual, predicted))
}
f1.score(data.all, class, scored.class) # 0.6067416

# 9. Let's consider the following question: 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.)

# 10
my.roc <- function(df,actual,probability){

  # http://freakonometrics.hypotheses.org/9066
  roc.curve=function(t,print=FALSE){
    p=probability
    a=actual
    Pt=(p>t)*1
    FP=sum((Pt==1)*(a==0))/sum(a==0) # false positive
    TP=sum((Pt==1)*(a==1))/sum(a==1) # true positive
    if(print==TRUE){
      print(table(Observed=a,Predicted=Pt))
    }
    vect=c(FP,TP)
    names(vect)=c("FPR","TPR")
    return(vect)
  }
  threshold = 0.5
  roc.curve(threshold,print=TRUE)

  ROC.curve=Vectorize(roc.curve)

  M.ROC=ROC.curve(seq(0,1,by=.01))
  plot(M.ROC[1,],M.ROC[2,],type="l",xlab = "1 - Specificity", ylab = "Sensitivity")
```

```
  abline(0,1, col="gray", lty=2)

  # https://www.kaggle.com/c/GiveMeSomeCredit/forums/t/942/r-script-for-auc-calculation
  my.auc<-function(df,actual,probability){
    N=length(probability)
    N_pos=sum(actual)
    df=data.frame(out=actual,prob=probability)
    df=df[order(-df$prob),]
    df$above=(1:N)-cumsum(df$out)
    return(1-sum(df$above*df$out)/(N_pos*(N-N_pos)))
  }
  my.auc(df,actual,probability)
}

my.roc(data.all, class, scored.probability) # 0.8503113

# 11
accuracy(data.all,class,scored.class) # 0.8066298
class.error.rate(data.all,class,scored.class) # 0.1933702
precision(data.all,class,scored.class) # 0.84375
sensitivity(data.all,class,scored.class) # 0.4736842
specificity(data.all,class,scored.class) # 0.9596774
f1.score(data.all,class,scored.class) # 0.6067416

my.roc(data.all,class,scored.probability) # 0.8503113
# AUC = # 0.8503113

# 12
# use caret
# install.packages("caret")
library(caret)
ls("package:caret") # lists objects in package
lsf.str("package:caret") # lists functions in package

confusionMatrix(data.all$scored.class,data.all$class)
# This results in sensitivity and specificity values that are flipped from the values I got using my
functions.

confusionMatrix(data.all$scored.class,data.all$class, positive="1")
# This results in the same sensitivity and specificity values I got using my functions.

# Because I named my sensitivity and specificity functions the same as the functions in the
caret package, I have to specify the package.
caret::sensitivity(as.factor(data.all$scored.class),as.factor(data.all$class)) # 0.9596774
caret::sensitivity(as.factor(data.all$scored.class),as.factor(data.all$class), positive = "1") #
0.4736842

caret::specificity(as.factor(data.all$scored.class),as.factor(data.all$class)) # 0.4736842
caret::specificity(as.factor(data.all$scored.class),as.factor(data.all$class), negative = "0") #
0.9596774
```

Andrea Bruckner
Predict 422, Sec 56

```
# 13
# code from ch. 11 page 23/27
# install.packages("pROC")
library(pROC)
ls("package:pROC") # lists objects in package
lsf.str("package:pROC") # lists functions in package
rocCurve <- function(actual, probability){
  class <- as.factor(class)
  roc.plot <- roc(response = class,
                  predictor = scored.probability,
                  ## This function assumes that the second
                  ## class is the event of interest, so we
                  ## reverse the labels.
                  levels = rev(levels(class)),
                  auc = TRUE)
  plot(roc.plot, legacy.axes = TRUE)
}

rocCurve(class,scored.probability) # 0.8503
# AUC = 0.8503


# A shorter way to do this?
plot.roc(class, scored.probability, legacy.axes = TRUE) # 0.8503
```