data621hw2

jim lung

03-10-2018

```
knitr::opts_chunk$set(echo = TRUE)
```

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

```
require("plyr")
```

```
## Loading required package: plyr
```

```
require("knitr")
```

```
## Loading required package: knitr
```

```
require("psych")
```

```
## Loading required package: psych
```

```
require("knitr")
require("ggplot2")
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following objects are masked from 'package:psych':
##
##     %+%, alpha
```

```
require("pracma")
```

```
## Loading required package: pracma
```

```
## Warning in library(package, lib.loc = lib.loc, character.only = TRU
E,
## logical.return = TRUE, : there is no package called 'pracma'
```

```
df <- read.csv(url('https://raw.githubusercontent.com/fung1091/data621/
master/HW2/classification-output-data.csv'))
kable(head(df))
```

pregnant	glucose	diastolic	skinfold	insulin	bmi	pedigree	age	class	scored.class	scored.probability
7	124	70	33	215	25.5	0.161	37	0	0	0.3284523
2	122	76	27	200	35.9	0.483	26	0	0	0.2731904
3	107	62	13	48	22.9	0.678	23	1	0	0.1096604
1	91	64	24	0	29.2	0.192	21	0	0	0.0559984
4	83	86	19	0	29.3	0.317	34	0	0	0.1004907
1	100	74	12	46	19.5	0.149	28	0	0	0.0551546

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

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?

```
kable(table(df$class,df$scored.class))
```

	0	1
0	119	5
1	30	27

To assume 0 is negative, and 1 is positive, Scored.class is the predicted class, and is represented horizontally. Class is the actual class, and is represented vertically.

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.

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

First, let's define a helper function to calculate the confusion matrix values:

```
confusion_mx <- function(df){
  data.frame(tp=nrow(df[df$class==1 & df$scored.class==1,]),
             tn=nrow(df[df$class==0 & df$scored.class==0,]),
             fp=nrow(df[df$class==0 & df$scored.class==1,]),
             fn=nrow(df[df$class==1 & df$scored.class==0,])
  )
}
kable(confusion_mx(df))
```

tp	tn	fp	fn
27	119	5	30

And now accuracy:

```
accuracy<-function(df){
  f <- confusion_mx(df)
  (f$tp+f$tn)/(f$tp+f$fp+f$tn+f$fn)
}
accuracy(df)
```

```
## [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.

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

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

```
classification_error<-function(df){
  f <- confusion_mx(df)
  (f$fp+f$fn)/(f$tp+f$fp+f$tn+f$fn)
}
classification_error(df)
```

```
## [1] 0.1933702
```

verifing the sum equal to one:

```
sum <- classification_error(df) + accuracy(df)
sum
```

```
## [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.

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

```
precision<-function(df){
  f <- confusion_mx(df)
  (f$tp)/(f$tp+f$fp)
}
precision(df)
```

```
## [1] 0.84375
```

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.

$$Sensitivit \quad = \frac{TP}{TP + FN}$$

```
sensitivity<-function(df){
  f <- confusion_mx(df)
  (f$tp)/(f$tp+f$fn)
}
sensitivity(df)
```

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

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.

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

```
specificity<-function(df){
  f <- confusion_mx(df)
  (f$tn)/(f$tn+f$fp)
}
specificity(df)
```

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

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.

$$F1Score = \frac{2 \times Precision \times Sensitivity}{Precision + Sensitivity}$$

```
f1_score<-function(df){
  p<- precision(df)
  s<- sensitivity(df)
  2*p*s/(p+s)
}
f1_score(df)

## [1] 0.6067416
```

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.

F1 score are bound by the prescision and sensitivity both between 0 and 1:

```
# assume p is prescision and s is sensitivity.
p <- runif(100, min = 0, max = 1)
s <- runif(100, min = 0, max = 1)
f <- (2*p*s)/(p+s)
summary(f)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0104  0.2196  0.3905  0.4206  0.6154  0.9179
```

F1 score are proved between 0 and 1.

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.

```
ROC <- function(df)
{
  data1 = df
  thresholds <- seq(0,1,0.01)
  Y <- c()
  X <- c()
  for (threshod in thresholds) {
    data1$scored.class <- ifelse(data1$scored.probability > threshod,1,
0)
    X <- append(X,1-specificity(data1))
    Y <- append(Y,sensitivity(data1))
    }
  df1 <- data.frame(X=X,Y=Y)
  df1 <- na.omit(df1)
  g <- ggplot(df1,aes(X,Y)) + geom_line() + ggtitle('Custom ROC Curve')
 +
    xlab('Specificity') + ylab('Sensitivity')
  height = (df1$Y[-1]+df1$Y[-length(df1$Y)])/2
```

```
  width = -diff(df1$X)
  area = round(sum(height*width),4)
  return(list(Plot =g,AUC = area))
}
```

11. Use your created R functions and the provided classification output data set to produce all of the classification metrics discussed above.

```
Name <- c('Accuracy','Classification Error Rate', 'Precision', 'Sensiti
vity','Specificity', 'F1 Score')
Value <- round(c(accuracy(df), classification_error(df), precision(df),
 sensitivity(df), specificity(df), f1_score(df)),4)
df1 <- as.data.frame(cbind(Name, Value))
kable(df1)
```

Name	Value
Accuracy	0.8066
Classification Error Rate	0.1934
Precision	0.8438
Sensitivity	0.4737
Specificity	0.9597
F1 Score	0.6067

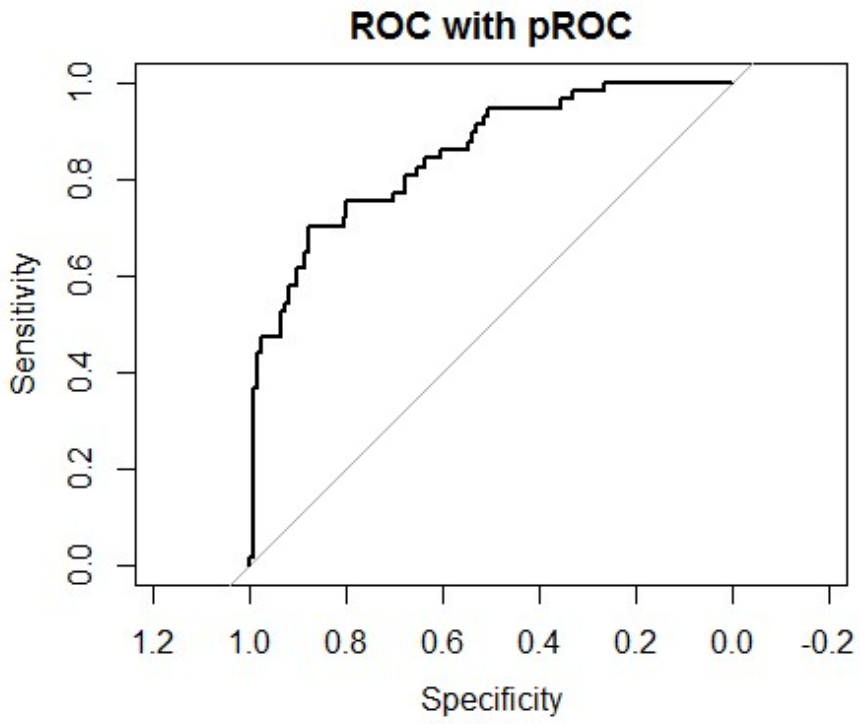
12. Investigate the caret package. In particular, consider the functions confusionMatrix, sensitivity, and specificity. Apply the functions to the data set. How do the results compare with your own functions?

```
require("caret")

## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following objects are masked _by_ '.GlobalEnv':
##
##     precision, sensitivity, specificity

d_tab <- table(df$class,df$scored.class)
confusionMatrix(d_tab, reference = df$class)

## Confusion Matrix and Statistics
##
##
##       0   1
##   0 119   5
##   1  30  27
##
```

```
##                Accuracy : 0.8066
##                  95% CI : (0.7415, 0.8615)
##     No Information Rate : 0.8232
##     P-Value [Acc > NIR] : 0.7559
##
##                   Kappa : 0.4916
##  Mcnemar's Test P-Value : 4.976e-05
##
##             Sensitivity : 0.7987
##             Specificity : 0.8438
##          Pos Pred Value : 0.9597
##          Neg Pred Value : 0.4737
##              Prevalence : 0.8232
##          Detection Rate : 0.6575
##    Detection Prevalence : 0.6851
##       Balanced Accuracy : 0.8212
##
##        'Positive' Class : 0
##
```

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?

```
require("pROC")
d_roc <- roc(df$class,df$scored.probability)
plot(d_roc, main = "ROC with pROC")
```



```r
ci(d_roc)
```

```
## 95% CI: 0.7905-0.9101 (DeLong)
```