hw2-tora

Deepa, Tora

2022-09-26

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(tidyr);
library(dplyr);
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(kableExtra);
```

```
## Warning in !is.null(rmarkdown::metadata$output) && rmarkdown::metadata$output
## %in% : 'length(x) = 2 > 1' in coercion to 'logical(1)'

##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```
library(caret)
```

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

```
## Loading required package: lattice
```

```
library(ggplot2)
```

1. Loaded classification data set from GitHub.

```
data<- read.csv("https://raw.githubusercontent.com/deepasharma06/Data621-HW2/main/classification-output-
head(data) %>% kable() %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed"), full_w
```

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?

```
#row: predicted value; columns: actual value
conf_matrix = table(Prediction = data$scored.class, Actual = data$class)
conf_matrix %>% kable() %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed"), full_
```

	0	1
0	119	30
1	5	27

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 = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #Accurary = (TP + TN) / (TP + FP +TN +FN)
  return(round((TP+TN)/(TP + FP + TN + FN), 4))
}
```

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.

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

```
errorRate = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #Classification Error Rate = ( FP + FN )/(TP + FP +TN +FN)
  return(round((FP+FN)/(TP + FP + TN + FN), 4))
}
print(paste0("Error rate: ", errorRate(data, 'scored.class', 'class')))
```

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

```
#accuracy + error rate
print(paste0("Accuracy + Error rate = ", accuracy(data, 'scored.class', 'class'), " + ", errorRate(data
```

```
## [1] "Accuracy + Error rate = 0.8066 + 0.1934 = 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 = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #Precision = TP / (TP + FP)
```

```
  return(round((TP)/(TP + FP), 4))
}
print(paste0("Precision: ", precision(data, 'scored.class', 'class')))
```

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

6. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the sens of the predictions. Sensitivity is also known as recall.

Sensitivity = TP / (TP + FN)

```
sens = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #Sensitivity = TP / (TP + FN)
  return(round((TP)/(TP + FN), 4))
}
print(paste0("Sensitivity: ", sens(data, 'scored.class', 'class')))
```

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

7. Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the spec of the predictions.

specificity = TN / (TN+FP)

```
spec = function(data, predicted_col_name, actual_col_name) {

  conf = table(data[ , predicted_col_name], data[ , actual_col_name])
  TP = conf[2,2]
  TN = conf[1,1]
  FP = conf[2,1]
  FN = conf[1,2]

  #specificity= TN / (TN+FP)
  return(round((TN)/(TN + FP), 4))
}
print(paste0("specificity: ", spec(data, 'scored.class', 'class')))
```

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

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.

```
f1_score = function(data, predicted_col_name, actual_col_name) {
  p = precision(data, 'scored.class', 'class')
  s = sens(data, 'scored.class', 'class')
  return(2*p*s/(p+s))
}
print(paste0("F1 Score: ", f1_score(data, 'scored.class', 'class')))
```

```
## [1] "F1 Score: 0.606767453510436"
```

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. (Hint: If 0 < a < 1 and 0 < b < 1 then ab < a.)

The F1 score is equivalent to the following:

$$p = precision, s = Sensitivity$$

$$F1 = \frac{2 * p * s}{p + s}$$

Just by eye-balling this equation (and knowing that both precision and Sensitivity are between 0 and 1), if we assume mutual exclusivity of both metrics, F1 would simplify to the following:

$$p = 1, s = 1 F1 = \frac{2 * 1 * 1}{1 + 1} = 1$$

To figure out the minimum values, we can take the derivative of F1 and set it to zero. Since we have two metrics, we'll have to use partial derivatives:

$$dF1/dp = 2s^2/(p + s)^2$$

$$dF1/ds = 2p^2/(p + s)^2$$

If we set both values at 0, then precision and Sensitivity would be at 0 for the zeroes of the function. If we go back and plug those values in, F1 would be 0.

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.

```r
getROCcurve = function(col.true = "class", col.probability = "scored.probability", data) {
  vec.TPR = c()
  vec.TNR = c()

  for (i in seq(0, 1, 0.01)) {
    data = data %>% mutate(model.classification = ifelse(unlist(select(data, col.probability)) < i, 0, 1
    i.vec.TP = data %>% filter(model.classification == class & class == 1) %>% nrow()
    i.vec.FN = data %>% filter(model.classification != class & class == 1) %>% nrow()
    i.vec.TN = data %>% filter(model.classification == class & class == 0) %>% nrow()
    i.vec.FP = data %>% filter(model.classification != class & class == 0) %>% nrow()

    vec.TPR = c(vec.TPR, (i.vec.TP/(i.vec.TP + i.vec.FN)))
    vec.TNR = c(vec.TNR, (i.vec.TN/(i.vec.TN + i.vec.FP)))

  }
  df.ROC = data.frame(Threshold = seq(0, 1, 0.01), TNR = vec.TNR, TPR = vec.TPR)
  df.ROC = df.ROC %>% arrange(TNR, decreasing = T)
  plt.ROC = ggplot(aes(x = 1-(TNR), y = (TPR)), data = df.ROC) + geom_step() + labs(x = "1 - specificity
  df.AUC  = df.ROC %>% distinct(TPR, TNR)

  df.AUC <- df.AUC %>%  mutate(TNR_next = lead(TNR, n = 1L))
  df.AUC$width = df.AUC$TNR_next - df.AUC$TNR
  vec.AUC = sum(df.AUC$width * df.AUC$TPR, na.rm = T)
  return(list("AUC" = vec.AUC, "ROC" = plt.ROC))
}
```

11.

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

Accuracy

```r
print(paste0("Accuracy: ", accuracy(data, 'scored.class', 'class')))
```

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

Error rate

```r
print(paste0("Error rate: ", errorRate(data, 'scored.class', 'class')))
```

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

Precision

```
print(paste0("Precision: ", precision(data, 'scored.class', 'class')))
```

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

Sensitivity/Recall

```
print(paste0("sensitivity: ", sens(data, 'scored.class', 'class')))
```

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

specificity

```
print(paste0("specificity: ", spec(data, 'scored.class', 'class')))
```

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

F1 Score

```
print(paste0("F1 Score: ", f1_score(data, 'scored.class', 'class')))
```

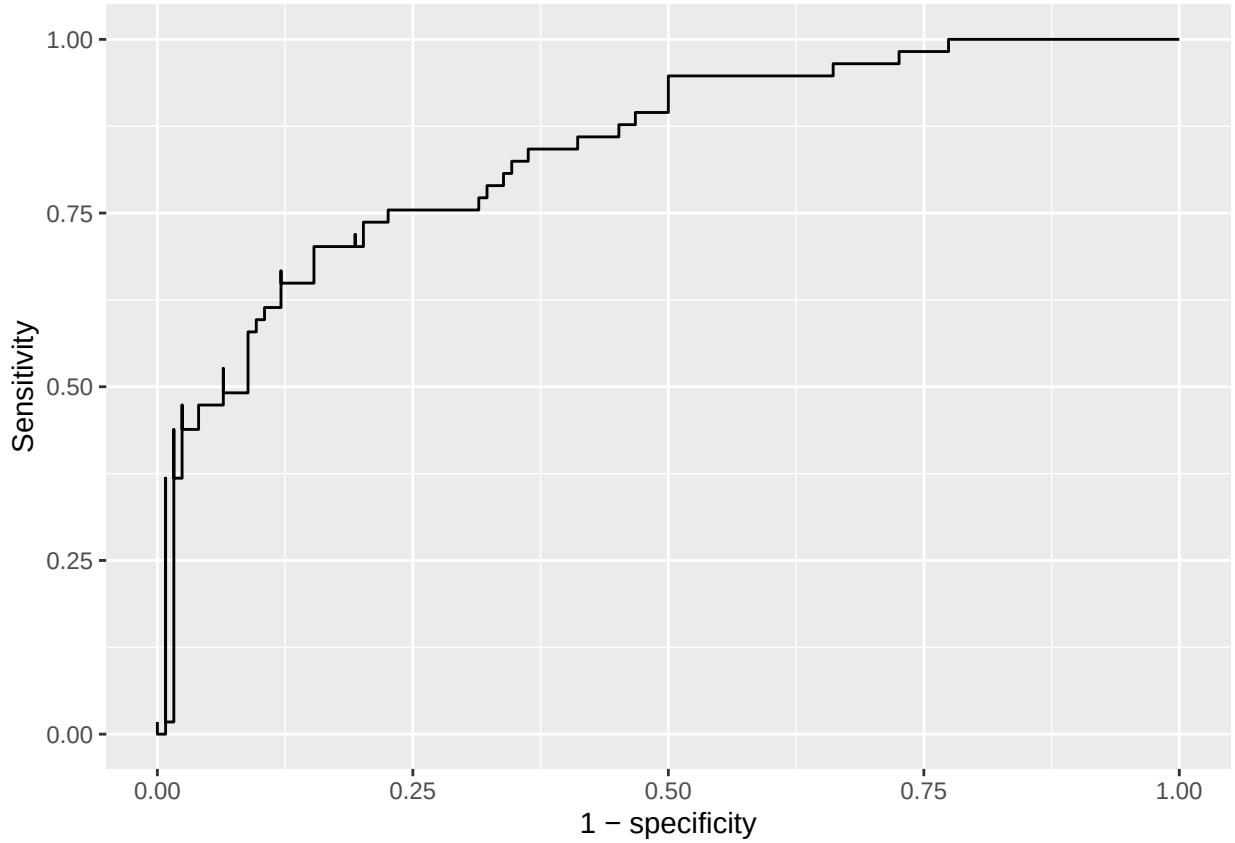
```
## [1] "F1 Score: 0.606767453510436"
```

ROC Curve

```
getROCcurve(data = data)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(col.probability)‘ instead of ‘col.probability‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.

## $AUC
## [1] 0.8539898
##
## $ROC
```



12. Investigate the carat 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?

all of the results that we calculated above for the confusion matrix, sensitivity, and specificity match the values givin by the matching functions as shown below.

```
scored<-as.factor(data$scored.class)
tclass<-as.factor(data$class)

confusionMatrix(scored, tclass)
```

```
## 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 : 0.9597
##             Specificity : 0.4737
##          Pos Pred Value : 0.7987
##          Neg Pred Value : 0.8438
##              Prevalence : 0.6851
##          Detection Rate : 0.6575
##    Detection Prevalence : 0.8232
##       Balanced Accuracy : 0.7167
##
##        'Positive' Class : 0
##
```

```
sensitivity(scored, tclass)
```

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

```
specificity(scored, tclass)
```

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

13.

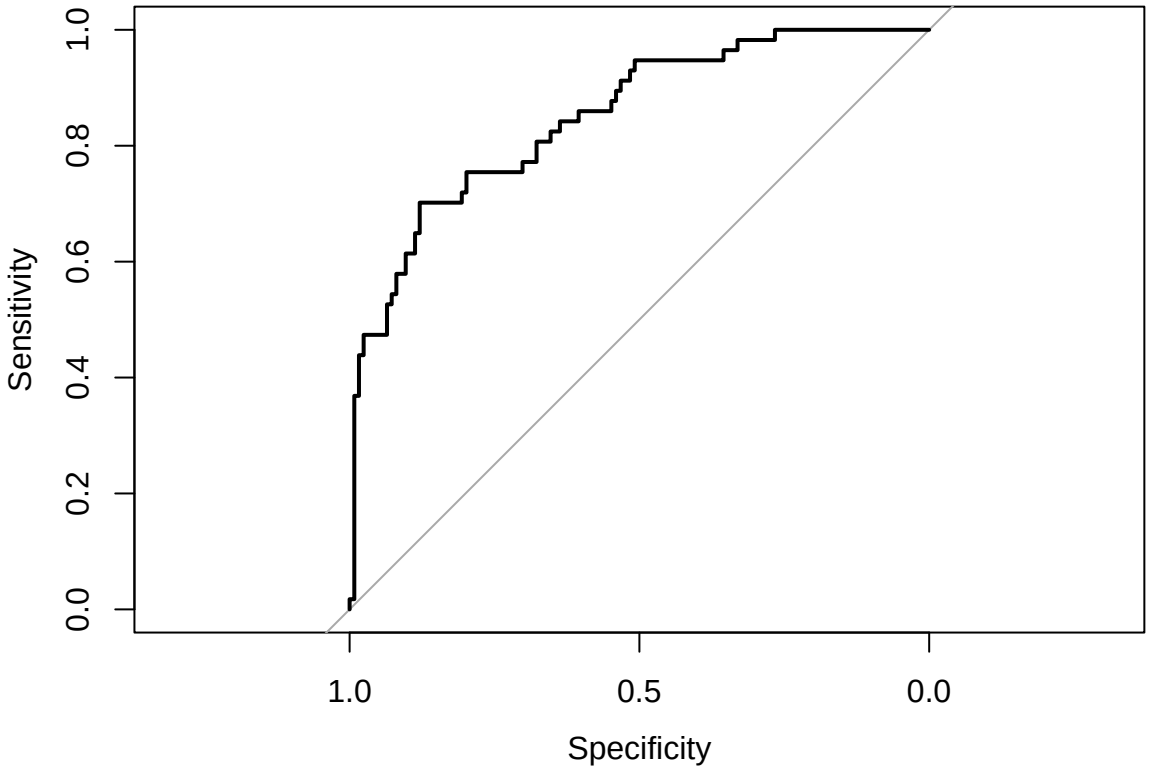
investigate the pROc package. use it to generate an ROC curve for the data set. How do the results compare with your own function.

altogether our two ROC curves are very close to one another. Both curves show a nonlinear relationship showing our model is significantly better then a randomized model aditionally both AUC measurements are close with our handmade function having an AUC of 0.854 and the pROC function having a AUC of 0.8503. Altogether both plots look very similar and reflect that the model is a fairly good predictor of the data.both plots look very similar.

```
roc_score=roc(data, class, scored.probability,plot=TRUE) #AUC score
```

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

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




```
#ggroc(roc_score)
auc(roc_score)
```

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