Assignment 2 - Data 621

Coco Donovan

2024-03-01

Loading Packages:

```
library(readr)
library(knitr)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.1
```

```
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.3.1

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

##
## Attaching package: 'pROC'

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

```
library(caret)
```

```
## Loading required package: lattice

## Warning: package 'lattice' was built under R version 4.3.1
```

```
library(glue)
```

```
## Warning: package 'glue' was built under R version 4.3.1
```

1.

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

```
df <- read_csv('classification-output-data.csv')
```

```
## Rows: 181 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (11): pregnant, glucose, diastolic, skinfold, insulin, bmi, pedigree, ag...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

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?

Answer:

Confusion Matrices generally follow this format:

```
# Create the confusion matrix data
confusion_matrix <- matrix(c("TN", "FP", "FN", "TP"), nrow = 2, byrow = TRUE)
rownames(confusion_matrix) <- c("Predicted Negative", "Predicted Positive")
colnames(confusion_matrix) <- c("Actual Negative", "Actual Positive")

# Print the confusion matrix using kable
kable(confusion_matrix, format = "markdown")
```

	Actual Negative	Actual Positive
Predicted Negative	TN	FP
Predicted Positive	FN	TP

This would be our raw confusion matrix:

```
confusion_matrix <- table(df$scored.class, df$class)
confusion_matrix_with_totals <- addmargins(confusion_matrix)
print(kable(confusion_matrix_with_totals, format = "markdown"))
```

```
##
##
## |    |   0|  1| Sum|
## |:---|---:|--:|---:|
## |0   | 119| 30| 149|
## |1   |   5| 27|  32|
## |Sum | 124| 57| 181|
```

Note: Here 0 is just the binary form of "Negative" and 1 is just the binary form "Positive."

Based on the general format and my raw confusion matrix, I can say that 119 cases that were actually negative were predicted as negative, whereas 5 of those cases that were actually negative were predicted as positive. For the cases that were actually positive, 30 were incorrectly predicted as negative, whereas the other 27 cases associated with an actual positive were correctly predicted as positive.

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

```
Accuracy = function(df) {

  TP = sum(df$class == 1 & df$scored.class == 1)
  TN = sum(df$class == 0 & df$scored.class == 0)
  FP = sum(df$class == 0 & df$scored.class == 1)
  FN = sum(df$class == 1 & df$scored.class == 0)

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

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.

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

```
Classification_Error_Rate = function(df) {

  TP = sum(df$class == 1 & df$scored.class == 1)
  TN = sum(df$class == 0 & df$scored.class == 0)
  FP = sum(df$class == 0 & df$scored.class == 1)
  FN = sum(df$class == 1 & df$scored.class == 0)

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

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

Answer:

Classification Error Rate is represented by the expression $\frac{FP+FN}{TP+FP+TN+FN}$ and accuracy is represented by the expression $\frac{TP+TN}{TP+FP+TN+FN}$. To check if the two sum to 1, I can just add the expressions, like so:

$(\frac{FP+FN}{TP+FP+TN+FN}) + (\frac{TP+TN}{TP+FP+TN+FN})$

This gives me the new fraction:

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

This fraction has the same numerator and denominator, meaning that the sum of Accuracy and Classification Error Rate should sum to 1, and if that is not the case, then something is wrong with the derived calculations.

Later in this analysis, I will check to see if my calculated Classification Error Rate and Accuracy sum to 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) {

  TP = sum(df$class == 1 & df$scored.class == 1)
  FP = sum(df$class == 0 & df$scored.class == 1)

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

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.

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

```
Sensitivity = function(df) {

  FN <- sum(df$class == 1 & df$scored.class == 0)
  TP <- sum(df$class == 1 & df$scored.class ==1)

  return((TP)/(TP+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) {

  TP = sum(df$class == 1 & df$scored.class == 1)
  TN = sum(df$class == 0 & df$scored.class == 0)
  FP = sum(df$class == 0 & df$scored.class == 1)
  FN = sum(df$class == 1 & df$scored.class == 0)

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

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

```
F1 = function (df) {
  f1 = (2 * Precision(df) * Sensitivity(df)) / (Precision(df) + Sensitivity(df))
  return(f1)
}
```

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 $ab < a$.)

Answer

Sensitivity = $\frac{TP}{TP+FN}$ and Precision = $\frac{TP}{TP+FP}$, where $TP, FPandFN$ are non-negative values. From that fact that all the values in these formulas are non-negative we can tell that:

$0 \leq Sensitivity \leq 1$

and

$0 \leq Precision \leq 1$.

It is true that if you multiply a number, a, by another number, b that is in between 0 and 1, the product of the two numbers will be at most a or b (depending on which number is greater). To show that this is true, I'll show what happens if b is 0 (the minimum value it can take on) and if b is 1 (the maximum value it can take on).

When $b = 0$:

$a \times b = a \times 0 = 0$

When $b = 1$:

$a \times b = a \times 1 =$ a

So, we can say that $ab \leq a$ and using this logic (given that a operates under the same between 0 and 1 condition) we can also say that $ab \leq b$. Further, I can then say that $2 \times ab \leq a + b$ (coming as a direct result of combining both of these inequalities).

Now, Precision and Sensitivity operate under the same between 0 and 1 condition as both a and b, so I can then say that $Precision \times Sensitivity \leq Sensitivity$ and that $Precision \times Sensitivity \leq Precision$.

Using the combination of inequalities I employed above, I can say that:

$2 \times Precision \times Sensitivity \leq Precision + Sensitivity$

The Left side of this inequality is the numerator of the F1 Score function and the right side of the inequality is the denominator of the F1 Score function. What I have shown is the the denominator will always be as large or larger than the numerator (meaning the maximum value it can take on is 1).

Therefore, because the F1 Score is positive and the maximum value it can take on is 1, the F1 Score will always be 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.

11.

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

```
df_accuracy = Accuracy(df)

print(df_accuracy)
```

Accuracy:

```
## [1] 0.8066298
```

It seems that this data represents a classification with about 80.6% accuracy.

```
df_cle = Classification_Error_Rate(df)

print(df_cle)
```

Classification Error Rate:

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

It seesm that this data represents a classifcation with a Classification Error Rate of about 19.3%.

```
print((df_cle + df_accuracy) == 1)
```

Verifying that Accuracy + Classification Error Rate = 1:

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

The accuracy and error rate sum to 1 in this example using the functions I created.

```
df_precision = Precision(df)
print(df_precision)
```

Precision:

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

Based on this calculation, this data represents a classification method with about 84.4% precision.

```
df_sensitivity = Sensitivity(df)
print(df_sensitivity)
```

Sensitivity:

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

Based on this calculation, this data represents a classification method with about 47.4% sensitivity.

```
df_specificity = Specificity(df)
print(df_specificity)
```

Specificity:

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

Based on this calculation, this data represents a classification method with about 96% specificity.

```
F1(df)
```

F1 Score:

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

It seems that this data represents a classification with an F1 Score of 0.6067416.

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?

```
caret = confusionMatrix(as.factor(df$scored.class), as.factor(df$class), positive = "1")

print(glue("caret: {caret$overall[['Accuracy']]}"))
```

```
## caret: 0.806629834254144
```

```
print(glue("My calculation: {Accuracy(df)}"))
```

```
## My calculation: 0.806629834254144
```

The value I calculated for Accuracy is the same as the value produced using the confusionMatrix package.

```
print('Sensitivity')
```

```
## [1] "Sensitivity"
```

```
print(glue("confusionMatrix: {caret$byClass[['Sensitivity']]}"))
```

```
## confusionMatrix: 0.473684210526316
```

```
print(glue("My calculation: {Sensitivity(df)}"))
```

```
## My calculation: 0.473684210526316
```

```
print(glue('sensitivity function: {sensitivity(as.factor(df$scored.class), as.factor(df$class), positive
```

```
## sensitivity function: 0.473684210526316
```

The value I calculated for Sensitivity is the same as the value produced using the confusionMatrix and sensitivity functions.

```
print('Specificity')
```

```
## [1] "Specificity"
```

```
print(glue("caret: {caret$byClass[['Specificity']]}"))
```

```
## caret: 0.959677419354839
```

```
print(glue("My calculation: {Specificity(df)}"))
```

```
## My calculation: 0.959677419354839
```

```
print(glue('specificity function: {specificity(as.factor(df$scored.class), as.factor(df$class), negative
```

```
## specificity function: 0.959677419354839
```

The value I calculated for Specificity is the same as the value produced using the confusionMatrix and specificity functions.

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?

```
plot(roc(df$class, df$scored.probability), main="ROC Curve from pROC Package")
```

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

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