Classification Model Metrics

Umer Farooq

2023-10-09

Introduction:

In this homework assignment, you will work through various classification metrics. You will be asked to create functions in R to carry out the various calculations. You will also investigate some functions in packages that will let you obtain the equivalent results. Finally, you will create graphical output that also can be used to evaluate the output of classification models, such as binary logistic regression.

Loading Data:

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

```
df <- read.csv(("https://raw.githubusercontent.com/Umerfarooq122/Data_sets/main/classification-output-da
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

Identifying the Columns:

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 :

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

	0	1
0	119	5
1	30	27

Accuracy:

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 = TP + TN / TP + FP + TN + FN
```

Answer :

let's create a function for the confusion matrix first:

```
confusion_matrix <- 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,])
             )
}

knitr::kable(confusion_matrix(df))
```

TP	TN	FP	FN
27	119	5	30

Now we can go ahead and create a function for accuracy and find out its value:

```
accuracy <- function(df){
  f <- confusion_matrix(df)
  (f$TP+f$TN)/(f$TP+f$TN+f$FP+f$FN)
}
```

```
accuracy(df)
```

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

Classification Error Rate:

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 = FP + FN / TP + FP + TN + FN
```

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

Answer :

Let's create the classification error rate function first:

```
classification_error_rate <- function(df){
  f <- confusion_matrix(df)
  (f$FP+f$FN)/(f$TP+f$TN+f$FP+f$FN)
}
```

Now let's find the answer of classification error rate:

```
classification_error_rate(df)
```

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

Now we can confirm the answer by adding accuracy and classification error rate and see if it comes up to be 1

```
verification <- accuracy(df)+classification_error_rate(df)
print(verification)
```

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

As we can see that the answer came out to be 1 which verifies our accuracy and error rate.

Precision:

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 = TP / TP + FP
```

Answer :

Creating function for precision:

```
precision <- function(df){
  f <- confusion_matrix(df)
  (f$TP)/(f$TP+f$FP)
}
```

Recalling the precision function to get value:

```
precision(df)
```

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

Sensitivity:

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 = TP / TP + FN
```

Answer :

Let's create the function for sensitivity:

```
sensitivity <- function(df){
  f <- confusion_matrix(df)
  (f$TP)/(f$TP+f$FN)
}
```

Recalling the function to get sensitivity:

```
sensitivity(df)
```

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

Specificity:

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 = TN / TN + FP

Answer :

Creating function for specificity:

```
specificity <- function(df){
  f <- confusion_matrix(df)
  (f$TN)/(f$TN+f$FP)
}
```

Recalling specificity function to get value:

```
specificity(df)
```

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

F1 Score:

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 = 2 X Precision X Sensitivity / Precision + Sensitivity

Answer :

Let's create the function first:

```
f1.score <- function(df){
  p <- precision(df)
  s <- sensitivity(df)
  (2*p*s)/(p+s)
}
```

Recalling the function to get F1 Score:

```
f1.score(df)
```

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

F1 Score Bounds:

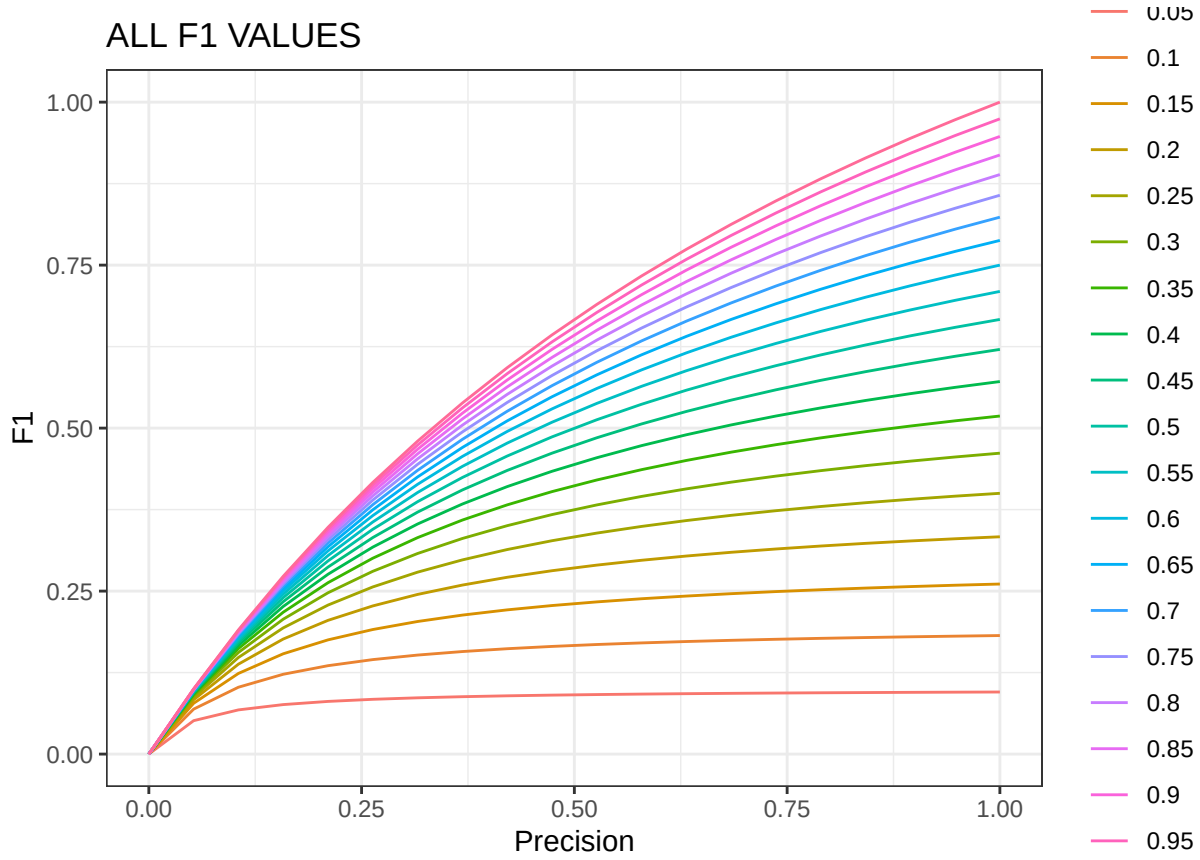
9. Before we move on, let's consider a question that was asked: What are the bounds on the F1 score? Show that the F1 score will always be between 0 and 1. (Hint: If 0 < < 1 and 0 < < 1 then < .)

Answer :

The F1 score is calculated from the precision and sensitivity scores which both have the bounds between 0 and 1, thus the highest score for F1 would be 1, from perfect precision and sensitivity, and the lowest would be 0, if either the precision or sensitivity value is 0.

We can also verify this graphically:

```
b = 20
p = seq(0,1, length.out = b)
df_n = NULL
for(i in 1:b) {
  s = i/b
  temp_df = data.frame(x = p, y = (2*p*s)/(p+s), col = rep(s:s,each = b))
  df_n = rbind(df_n, temp_df)
}
ggplot(df_n,aes(x=x,y=y,group=col,color=factor(col))) +
  geom_line() +
  ggtitle('ALL F1 VALUES') +
  theme(plot.title = element_text(hjust = 0.5)) +
  ylab('F1') +
  xlab('Precision') +
  ylim(0,1)+theme_bw()
```



or we can also:

```
set.seed(100)
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.
## 0.02261 0.21443 0.38498 0.43332 0.63433 0.94792
```

ROC Curve and AUC:

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.

Answer :

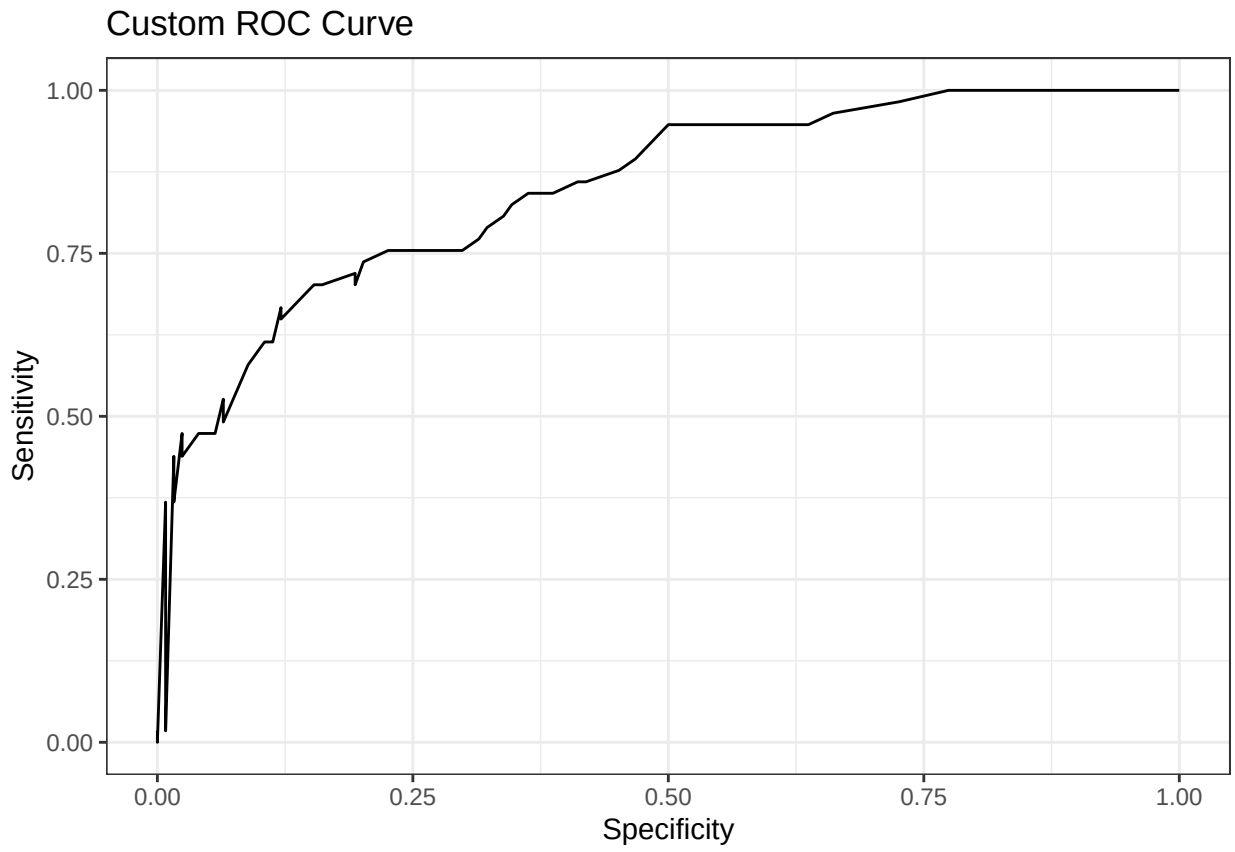
Let's create a function for ROC curve and AUC:

```
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')+theme_bw()
  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))
}
```

Let's recall the function to get the ROC curve and AUC

```
ROC(df)
```

```
## $Plot
```




```
##
## $AUC
## [1] 0.8489
```

Classification metrics:

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

Answer :

```
Name <- c('Accuracy','Classification Error Rate', 'Precision', 'Sensitivity','Specificity', 'F1 Score')
Value <- round(c(accuracy(df), classification_error_rate(df), precision(df), sensitivity(df), specificit
df_created <- as.data.frame(cbind(Name, Value))
kable(df_created)
```

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

Investigating `caret` Package:

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?

Answer :

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

```
## 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
##
```

As we can see our functions have returned the same values for accuracy, sensitivity, specificity and confusion matrix which cross verifies both our functions and `caret` package.

Investigating `pROC` package:

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?

Answer :

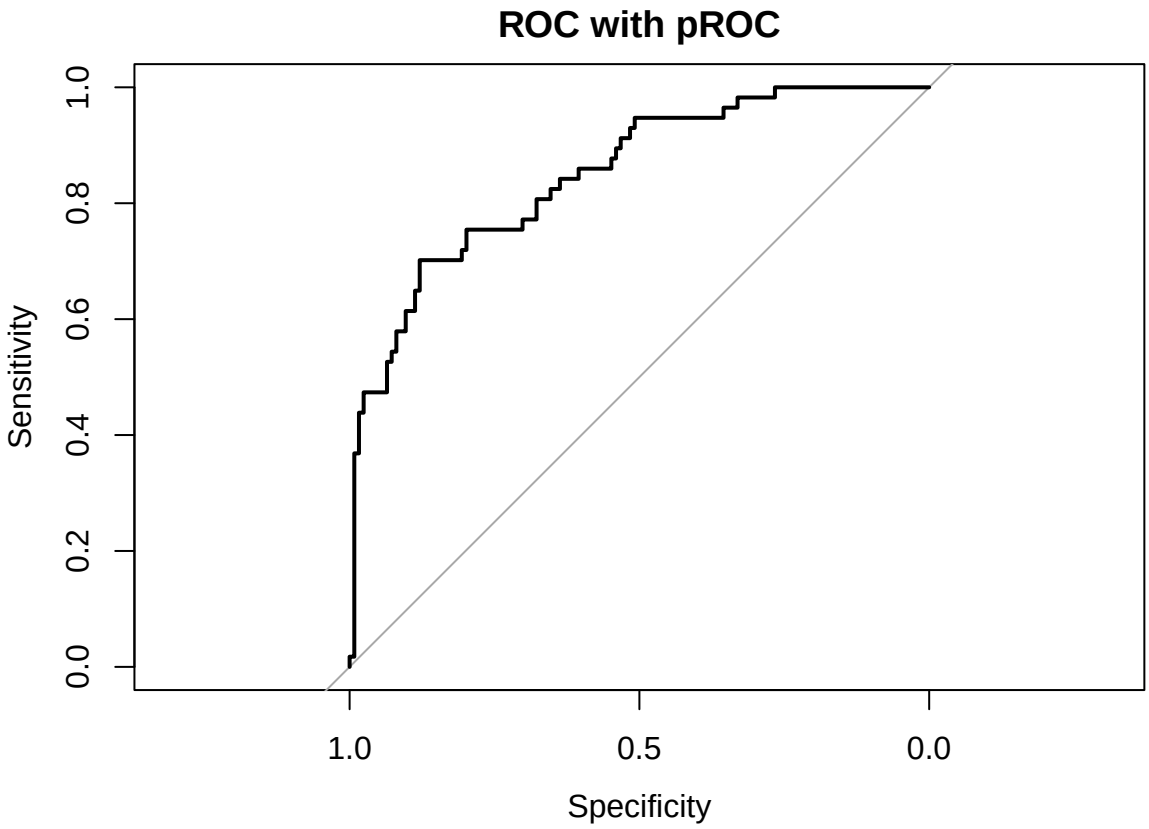
Let's create the ROC curve first:

```
Curve <- roc(df$class,df$scored.probability)
```

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

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

```r
plot(Curve, main = "ROC with pROC")
```



Similarly the AUC

```r
auc(Curve)
```

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

As we can see that the AUC using our created functions were around .84 and AUC using `pROC` package is around .85 which is really close to our value. Similarly, the ROC curve also looks similar.