Data 621 Homework 2

Critical Thinking Group 3: Vyannna Hill, Jose Rodriguez, and Christian Uriostegui

2023-10-15

Introduction

For this analysis, the objective is create functions in R to carry out various classification metrics. We will also be investigating functions from packages in R that can produce the equivalent results. Lastly, we will be creating a graphical output that can also be used to evaluate the output of classification models, such as binary logistic regression.

Classification metrics and the ROC curve

Creating the confusion matrix

For the confusion matrix, the table is a 2x2 matrix. The table's rows correspond with the predicted value of observation and the columns are the actual value of the observation. Looking at the confusion matrix, it appears 119 observation are predicted correctly out of the 181 observations

```
#confusion matrix: rows are the predicted value and columns are the actual values
conf.matrix<-table(data$scored.class,data$class)
colnames(conf.matrix)<-c("Positive","Negative")
rownames(conf.matrix)<-c("Positive","Negative")
conf.matrix
```

```
##
##            Positive Negative
##   Positive      119       30
##   Negative        5       27
```

Accuracy of the confusion matrix

The accuracy of the prediction is given through the formula $Accruacy = \frac{TP+TN}{TP+FP+TN+FN}$.

In order to perform the formula, we will need the terms below from the confusion matrix. * Terms +TP=True Positive +TN=True Negative +FN=False Negative +FP=False Positive

Looking at our confusion matrix: True Positive is located at [1,1], True Negative is located at [2,2], False Negative located at [2,1], and False Positive located at [1,2].

When the confusion matrix enter the functions, its accuracy score was ~80%. The model predicted correctly on the observations 80% of probability time. There is not a background on the data set provided if 80% accuracy is appropriate probability.

```
accuracy<-function(a){
  #grabbing values from imported data set
  fp<-a[1,2]
  fn<-a[2,1]
  tp<-a[1,1]
  tn<-a[2,2]

  #Applying to the accuracy formula provided
  acc<-(tp+tn)/(tp+fp+tn+fn)
  sprintf("Accuarcy of the data set is %.03f",acc)
  return(acc)

}
#Realized question 11 wants this run all together
```

The classification error rate in the confusion matrix

The classification error is the rate of the model's Using the classification error formula below, let's calculate the error rate! $error = \frac{FP+FN}{TP+FP+TN+FN}$

The model saw a classification error of ~19%. Adding the accuracy score and the classification error rate together will sum to 1.

```
class.error<-function(a){
  #same terms uses above
  fp<-a[1,2]
  fn<-a[2,1]
  tp<-a[1,1]
  tn<-a[2,2]

  #Classification error formula provided in the PDF
  cerror<-(fp+fn)/(tp+fp+tn+fn)
  sprintf("The Model's classification error is %.03f",cerror)
  return(cerror)
}

#Verifying the accuracy and classification error sums to 1
#print(c.error+ac)
```

Precision in the confusion matrix

Precision is the rate of positive predictions where identified correctly. The formula below is used in this rate.

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

The model has a ~96% accuracy in correctly predicting positives.

```
precisn<-function(a){
    #only grabbing positives from confusion matrix
    fp<-a[1,2]
    tp<-a[1,1]

    #using the formula provided
```

```
    pre<-tp/(tp+fp)
    sprintf("The Model's precision is %.03f",pre)
    return(pre)
}
```

Sensitivity

Sensitivity is the ratio of number of actual positives correctly predicted to the total number of actual positives.

```
sensitivity<-function(a){
    #only grabbing positives from confusion matrix
    fn<-a[2,1]
    tp<-a[1,1]

    #using the formula provided
    sen<-tp/(tp+fn)
    sprintf("The Model's sensitivity is %.03f",sen)
    return(sen)
}
```

Specificity

Specificity is the ratio of number of actual True negatives to the total number of actual negatives.

```
specificity<-function(a){
    #only grabbing positives from confusion matrix
    fp<-a[1,2]
    tn<-a[2,2]

    #using the formula provided
    spe<-tn/(tn+fp)
    sprintf("The Model's specificity is %.03f",spe)
    return(spe)
}
```

F1 score

F1-Score is a measure of combining both precision and sensitivity. Its value ranges between 0 and 1, where the latter represents perfect precision and sensitivity in the model.

```
f1.score<-function(a){
    #generating precision and sensitivity
    pre <- precisn(a)
    sen <- sensitivity(a)

    #using the formula provided
    score<-(2*pre*sen)/(pre+sen)
    sprintf("The Model's f1 score is %.03f",score)
    return(score)
}
```

Bounds of the F1 score The F1-score is a harmonic mean. This means that it calculates the average of the ratios or rates by equalizing the weights. Because F1-Score is a ratio, it will always bound between 0 and 1. Below are some examples to demonstrate that is will always equal a value between 0 and 1.

1. When precision = sensitivity

```
precision_equal_sensitivity <- 0.5
f1_score_equal_precision_sensitivity <- 2 * (precision_equal_sensitivity * precision_equal_sensitivity)
sprintf("When precision = sensitivity, F1 Score = %.03f", f1_score_equal_precision_sensitivity)
```

```
## [1] "When precision = sensitivity, F1 Score = 0.500"
```

2. When precision = 0.1 to 1 and sensitivity = 1.0

```
precisions <- seq(0.1, 1.0, by = 0.1)
sensitivity_fixed <- 1.0  # Sensitivity is fixed at 1.0
f1_scores_varying_precision <- (2 * precisions * sensitivity_fixed) / (precisions + sensitivity_fixed)
cat("F1 Scores when precision varies with sensitivity fixed at 1.0:\n")
```

```
## F1 Scores when precision varies with sensitivity fixed at 1.0:
```

```
print(f1_scores_varying_precision)
```

```
##  [1] 0.1818182 0.3333333 0.4615385 0.5714286 0.6666667 0.7500000 0.8235294
##  [8] 0.8888889 0.9473684 1.0000000
```

3. When precision = 1.0 and sensitivity = 0.1 to 1

```
sensitivities <- seq(0.1, 1.0, by = 0.1)
precision_fixed <- 1.0  # Precision is fixed at 1.0
f1_scores_varying_sensitivity <- (2 * precision_fixed * sensitivities) / (precision_fixed + sensitiviti
cat("F1 Scores when sensitivity varies with precision fixed at 1.0:\n")
```

```
## F1 Scores when sensitivity varies with precision fixed at 1.0:
```

```
print(f1_scores_varying_sensitivity)
```

```
##  [1] 0.1818182 0.3333333 0.4615385 0.5714286 0.6666667 0.7500000 0.8235294
##  [8] 0.8888889 0.9473684 1.0000000
```

Plotting the ROC Curve

```
generateROC <- function(true_class, predicted_prob) {
  thresholds <- seq(0, 1, by = 0.01)
  sensitivity <- numeric(length(thresholds))
  specificity <- numeric(length(thresholds))
```

```
  for (i in 1:length(thresholds)) {
    threshold <- thresholds[i]
    predicted_class <- ifelse(predicted_prob >= threshold, 1, 0)

    # Calculate TP, TN, FP, and FN
    TP <- sum(predicted_class == 1 & true_class == 1)
    TN <- sum(predicted_class == 0 & true_class == 0)
    FP <- sum(predicted_class == 1 & true_class == 0)
    FN <- sum(predicted_class == 0 & true_class == 1)

    sensitivity[i] <- TP / (TP + FN)
    specificity[i] <- TN / (TN + FP)
  }

  # Create an ROC curve object (you can keep this part from your original code)
  roc_curve <- roc(true_class, predicted_prob)

  # Plot ROC Curve (you can keep this part from your original code)
  plot(roc_curve, main = "ROC Curve")

  # Calculate AUC (you can keep this part from your original code)
  auc_value <- auc(roc_curve)

  # Create a list containing the ROC curve plot and AUC value
  result_list <- list(ROC_Curve = roc_curve, AUC = auc_value)

  return(result_list)
}
```

Running all the created functions above

```
#Running all the functions above in one statement
accuracy(conf.matrix)
```

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

```
class.error(conf.matrix)
```

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

```
precisn(conf.matrix)
```

```
## [1] 0.7986577
```

```
sensitivity(conf.matrix)
```

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

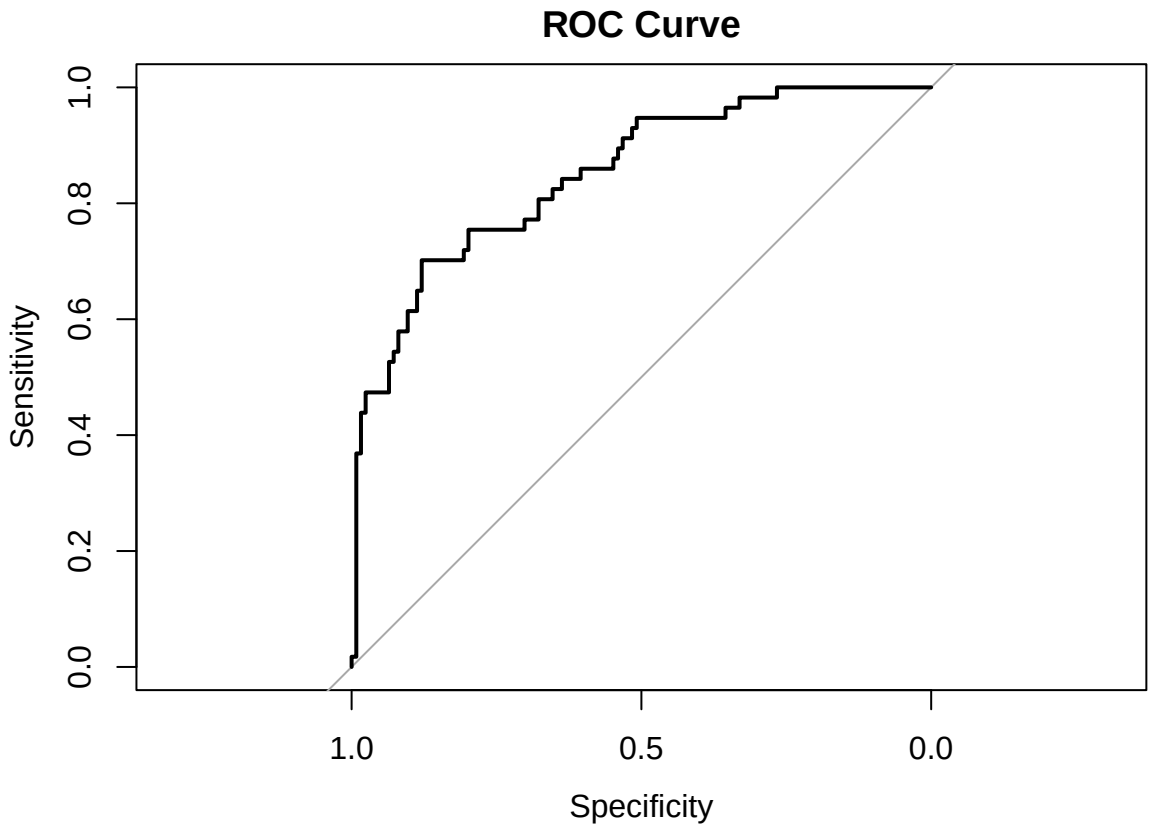
```r
specificity(conf.matrix)
```

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

```r
#plotting the ROC Curve
roc_result <- generateROC(data$class, data$scored.probability)
```

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

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




```r
# Access ROC curve and AUC values
roc_plot <- roc_result$ROC_Curve
auc_value <- roc_result$AUC
cat("AUC (Area Under the Curve):", auc_value, "\n")
```

```
## AUC (Area Under the Curve): 0.8503113
```

Investigating the caret package in comparsion to metrics above

Using the caret package below, we noticed specificity and sensitivity are closely matching to the values seen in the caret package. The confusion matrix is the same as the created matrix above as well.

```r
# convert columns to factors to run function
data$class <- as.factor(data$class)
data$scored.class <- as.factor(data$scored.class)

# Assuming your dataset is named 'data' with 'class' and 'scored.class' columns
conf_matrix_caret <- confusionMatrix(data$scored.class, data$class)

# Print the confusion matrix and other classification metrics
print(conf_matrix_caret)
```

```
## 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
##
```

Investigating the PROC package in comparsion of the ROC

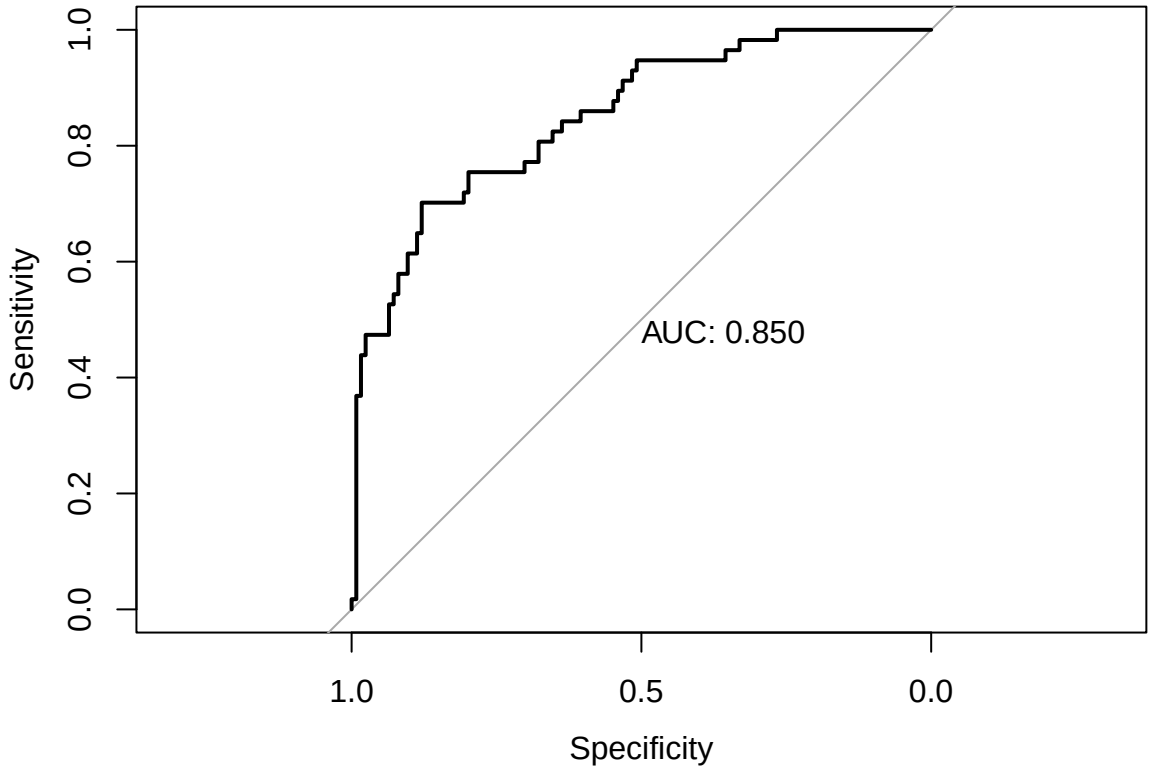
The created ROC function and pROC function for the ROC Curves look identicial with their AUC curves matches as well.

```r
roc_data <- roc(data$class, data$scored.probability)
```

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

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

```r
# Plot the ROC curve
plot(roc_data, print.auc = TRUE)
```




```
# Get the AUC
auc_value <- auc(roc_data)
sprintf("AUC: %.03f", auc_value)
```

```
## [1] "AUC: 0.850"
```