DATA 621 Business Analytics and Data Mining

Group 2 - Gabriel Campos, Melissa Bowman, Alexander Khaykin, & Jennifer Abinette

Last edited October 17, 2023

Homework #2 Assignment Requirements

Overview

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.

Supplemental Material

- Applied Predictive Modeling, Ch. 11 (provided as a PDF file).
- Web tutorials: http://www.saedsayad.com/model_evaluation_c.htm

Deliverables (100 Points)

- Upon following the instructions below, use your created R functions and the other packages to generate the classification metrics for the provided data set. A write-up of your solutions submitted in PDF format

Instructions

Complete each of the following steps as instructed:

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

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.

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

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.

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

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

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.

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

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.

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

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.

$$Specifity{=}\frac{TN}{TN+FN}$$

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.

$$F1\ Score{=}\frac{2\times Precision\times Sensitivity}{Presicion+Sensitivity}$$

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 $0 < \quad < 1$ and $0 < \quad < 1$ then $\quad < \quad$.)
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.
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?
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?

Data Exploration

1) Load the data

```r
git_url<-
  "https://raw.githubusercontent.com/GitableGabe/Data621_Data/main/"

df_classif <-
  read.csv(paste0(git_url,"classification-output-data.csv"))
head(df_classif,n=10)
```

```
##    pregnant glucose diastolic skinfold insulin  bmi pedigree age class
## 1         7     124        70       33     215 25.5    0.161  37     0
## 2         2     122        76       27     200 35.9    0.483  26     0
## 3         3     107        62       13      48 22.9    0.678  23     1
## 4         1      91        64       24       0 29.2    0.192  21     0
## 5         4      83        86       19       0 29.3    0.317  34     0
## 6         1     100        74       12      46 19.5    0.149  28     0
## 7         9      89        62        0       0 22.5    0.142  33     0
## 8         8     120        78        0       0 25.0    0.409  64     0
## 9         1      79        60       42      48 43.5    0.678  23     0
## 10        2     123        48       32     165 42.1    0.520  26     0
##    scored.class scored.probability
## 1             0         0.32845226
## 2             0         0.27319044
## 3             0         0.10966039
## 4             0         0.05599835
## 5             0         0.10049072
## 6             0         0.05515460
## 7             0         0.10711542
## 8             0         0.45994744
## 9             0         0.11702368
## 10            0         0.31536320
```

2) Confusion Matrix

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?

```r
table(df_classif$scored.class,df_classif$class)
```

```
##
##       0   1
##   0 119  30
##   1   5  27
```

Confusion matrices are often displayed in the ABCD format - Actual (Reference) as the columns with Predicted as Rows, and always displaying the outcome of interest (here "1") as the first column. **See Table 11.1 on page 254 of your Applied Predictive Modeling Chapter**. Thus, if you set up your table backwards as shown above when using the table function (Event = 1, but it was putting Nonevent = 0 first), then you've flipped the TP, TN, FN, and FP. *If you do not re-level the classification variables here, then you end up with a matrix that is inverted and thus most metrics are incorrect.* You even have to do this for confusionMatrix in **caret** to work correctly; it asks you to set the reference and predictions, but it will assume that the **lowest value** (so, 0) is the outcome of interest, which is not what we want here. We want to set **1 to be the outcome of interest**.

```
ls_class<-factor(df_classif$class)
ls_scr_class<-factor(df_classif$scored.class)
ls_sr_prb<-df_classif$scored.probability
```

```
# let's set the positive outcome to "1" with relevel
ls_class <- relevel(ls_class, ref = "1") ## changes it from the default ref of 0
ls_scr_class <- relevel(ls_scr_class, ref = "1")
```

Define a function to return a confusion matrix using **table()**. **Remember that table() requires we list the data in (rows,columns).**

```
conf_mat <- function(actual, predicted){
  ## Have to relevel again within the function
  actual <- relevel(ls_class, ref = "1") ## changes it from the default ref of 0
  predicted <- relevel(ls_scr_class, ref = "1")
  confusion_matrix <- table(predicted, actual)
  return(confusion_matrix)
}
conf_mat(actual, predicted)
```

```
##          actual
## predicted   1   0
##         1  27   5
##         0  30 119
```

As stated above, you have to **relevel()** to get the correct orientation of Event and Nonevent. We also want Actual values to be in the columns and Predicted values to be in the rows. We can see that, after releveling, our table is now in the correct orientation provided we give the data in (rows, columns) [**previously, it was given in (columns, rows) so the diagonal was inverted, further messing up metrics**].

A function to calculate the TP (True Positive):

```
tp_calc <- function(actual, predicted){
  tp <- conf_mat(actual, predicted)[1, 1]
  return(tp)
}
tp_calc(actual, predicted)
```

```
## [1] 27
```

A function to calculate the TN (True Negative):

```
tn_calc <- function(actual, predicted){
  tn <- conf_mat(actual, predicted)[2, 2]
  return(tn)
}
tn_calc(actual, predicted)
```

```
## [1] 119
```

A function to calculate the FP (False Positive):

```
fp_calc <- function(actual, predicted){
  fp <- conf_mat(actual, predicted)[1, 2]
  return(fp)
}
fp_calc(actual, predicted)
```

```
## [1] 5
```

A function to calculate the FN (False Negative):

```
fn_calc <- function(actual, predicted){
  fn <- conf_mat(actual, predicted)[2, 1]
  return(fn)
}
fn_calc(actual, predicted)
```

```
## [1] 30
```

3) Accuracy Function

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_calc <- function(df, col1, col2){
  actual <- df[,col1]
  predicted <- df[,col2]
  ## Call the previously defined functions
  tp  <- tp_calc(actual, predicted)
  tn  <- tn_calc(actual, predicted)
  fp  <- fp_calc(actual, predicted)
  fn  <- fn_calc(actual, predicted)
  ## Calculate accuracy
  accuracy <- (tp + tn)/(tp + fp + tn + fn)
  return(accuracy)
}

(accuracy <- accuracy_calc(df_classif, "class", "scored.class"))
```

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

4) Classification Error Rate Function

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.

```
class_error_rate <- function(df, col1, col2){
  actual <- df[,col1]
  predicted <- df[,col2]
  ## Call the previously defined functions
  tp  <- tp_calc(actual, predicted)
  tn  <- tn_calc(actual, predicted)
  fp  <- fp_calc(actual, predicted)
  fn  <- fn_calc(actual, predicted)
  ## Calculate classification error rate
  classification_error_rate <- (fp + fn)/(tp + fp + tn + fn)
  return(classification_error_rate)
}
(classification_error_rate <- class_error_rate(df_classif, "class", "scored.class"))
```

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

```
(accuracy + classification_error_rate)
```

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

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

5) Precision Function

$Precision = \frac{TP}{TP+FP}$ 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_calc <- function(df, col1, col2){
  actual <- df[,col1]
  predicted <- df[,col2]
  ## Call the previously defined functions
  tp <- tp_calc(actual, predicted)
  fp  <- fp_calc(actual, predicted)
  ## Calculate classification error rate
  precision <- tp/(tp + fp)
  return(precision)
}
(precision <- precision_calc(df_classif, "class", "scored.class"))
```

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

6) Sensitivity Function

$Sensitivity + \frac{TP}{TP+FN}$ 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_calc <- function(df, col1, col2){
  actual <- df[,col1]
  predicted <- df[,col2]
  ## Call the previously defined functions
  tp  <- tp_calc(actual, predicted)
  fn  <- fn_calc(actual, predicted)
  ## Calculate classification error rate
  sensitivity <- tp/(tp + fn)
  return(sensitivity)
}
(sensitivity <-  sensitivity_calc(df_classif, "class", "scored.class"))
```

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

7) Specificity Function

$Specifity = \frac{TN}{TN+FN}$ 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_calc <- function(df, col1, col2){
  actual <- df[,col1]
  predicted <- df[,col2]
  ## Call the previously defined functions
  tn  <- tn_calc(actual, predicted)
  fp  <- fp_calc(actual, predicted)
  ## Calculate classification error rate
  specificity <- tn/(tn + fp)
  return(specificity)
}
(specificity <-  specificity_calc(df_classif, "class", "scored.class"))
```

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

8) F1 score Function

$F1\ Score = \frac{2 \times Precision \times Sensitivity}{Presicion + Sensitivity}$ 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 <- function(df){
  ## Call the previously defined functions
  precision  <- precision_calc(df_classif, "class", "scored.class")
  sensitivity  <- sensitivity_calc(df_classif, "class", "scored.class")
  ## Calculate F1 score
  f1_score <- (2 * precision * sensitivity)/(precision + sensitivity)
  return(f1_score)
}
(f1_score <-  f1(df_classif))
```

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

9) F1 bounds

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 < .)

Step 1. Create a sequence of precision values and calculate f1 when sensitivity equals 50%

```
precision_seq <- seq(0, 1, length.out = 25)
f1_df <- data.frame(precision_seq)
# to calculate f1 using varying  precision and sensitivity = 50%
f1_df <- f1_df %>%
  mutate(f1_50 = (2 * precision_seq * 0.50)/(precision_seq + 0.50))
```

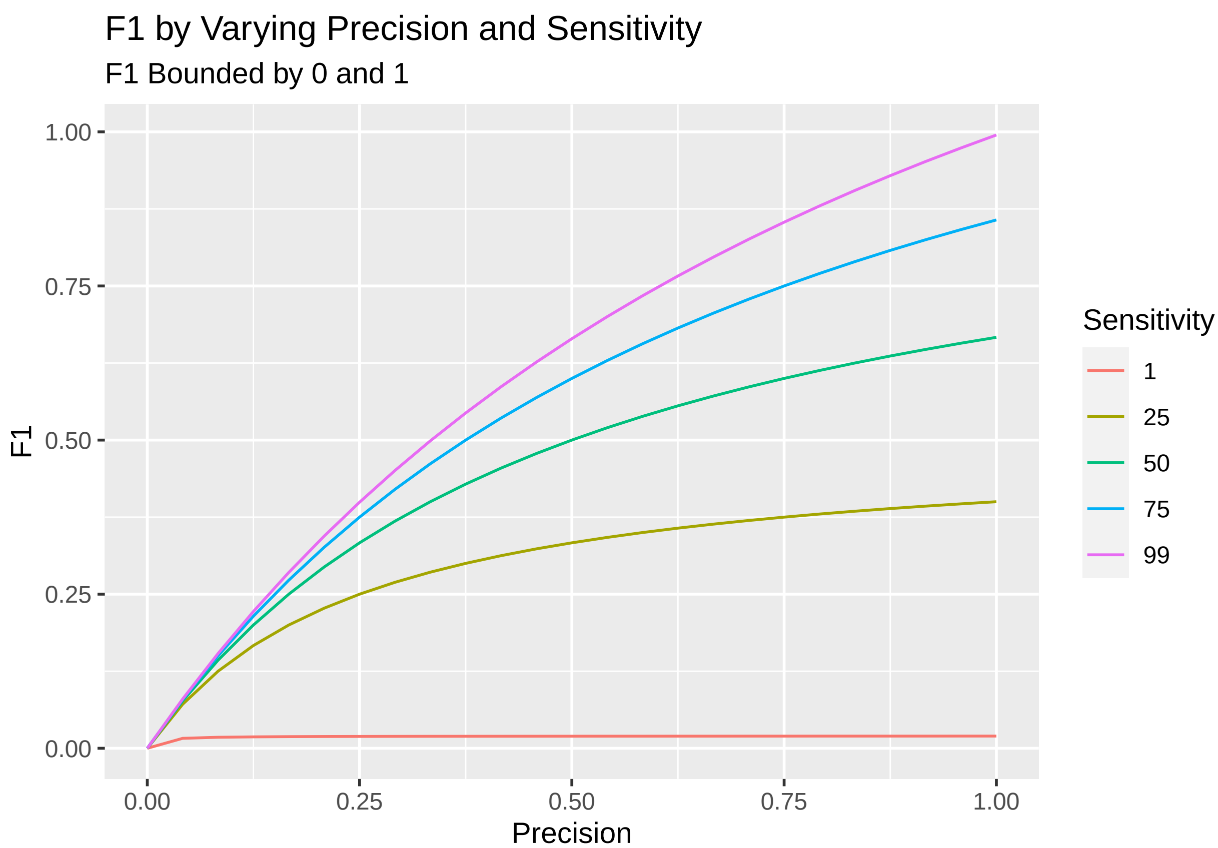
Step 2. Create a sequence of precision values and calculate f1 when sensitivity equals 1, 25, 75, and 99%

```
# to repeat for sensitivity 1, 25, 75, 99 percent
f1_df <- f1_df %>%
  mutate(f1_1 = (2 * precision_seq * 0.01)/(precision_seq + 0.01),
         f1_25 = (2 * precision_seq * 0.25)/(precision_seq + 0.25),
         f1_75 = (2 * precision_seq * 0.75)/(precision_seq + 0.75),
         f1_99 = (2 * precision_seq * 0.99)/(precision_seq + 0.99))
head(f1_df)
```

```
##   precision_seq      f1_50       f1_1      f1_25      f1_75      f1_99
## 1    0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## 2    0.04166667 0.07692308 0.01612903 0.07142857 0.07894737 0.07996769
## 3    0.08333333 0.14285714 0.01785714 0.12500000 0.15000000 0.15372671
## 4    0.12500000 0.20000000 0.01851852 0.16666667 0.21428571 0.22197309
## 5    0.16666667 0.25000000 0.01886792 0.20000000 0.27272727 0.28530259
## 6    0.20833333 0.29411765 0.01908397 0.22727273 0.32608696 0.34422809
```

Step 3. Create a line graph showing how F1 score changes over varying values of Sensitivity and Specificity.

```
f1_df %>% pivot_longer(cols = -precision_seq, names_to = "Sensitivity", names_prefix = "f1_", values_to
  ggplot(aes(x = precision_seq, y = f1, color = Sensitivity)) +
  geom_line() +
  labs(y = "F1", x = "Precision", title = "F1 by Varying Precision and Sensitivity", subtitle = "F1 Bour
```



No matter how high precision is or how high sensitivity is, because F1 is a harmonic mean of precision and sensitivity and because precision and sensitivity are bounded by 0 and 1, F1 can only ever be bounded by 0 and 1.

10) ROC curve

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.

Used ChatGPT for assistance in creating function below generate_ROC_curve

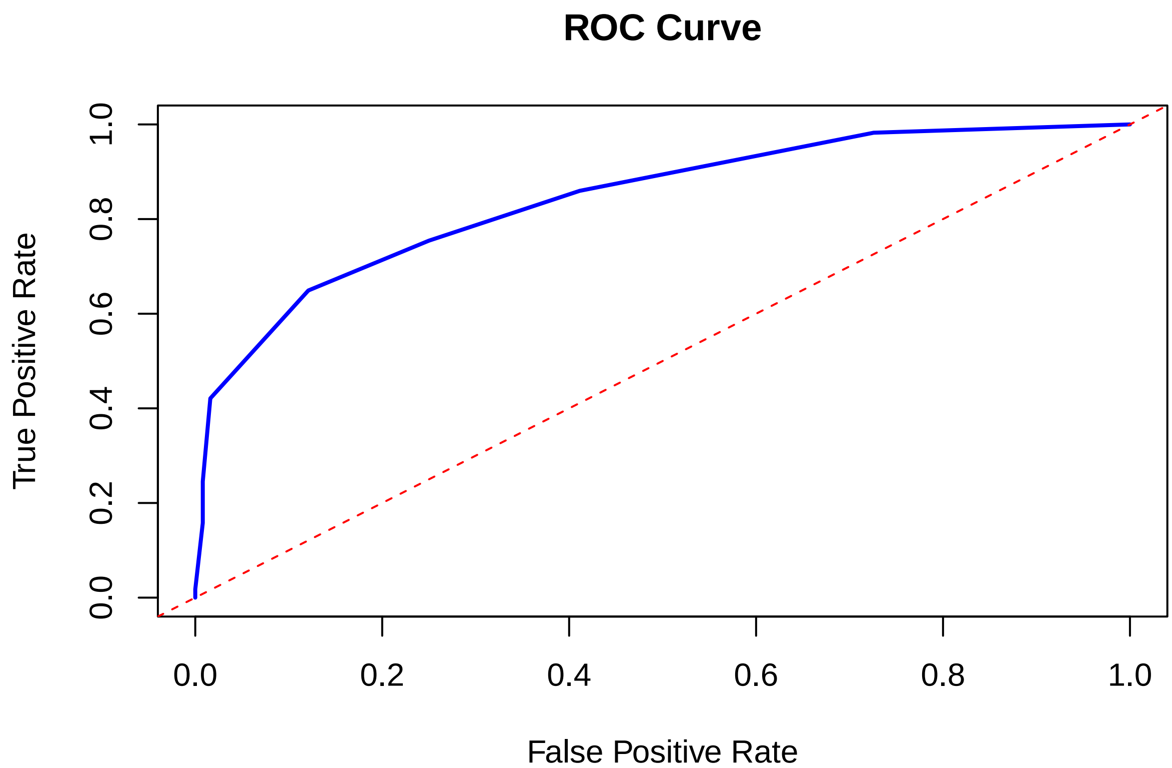
```
# Function to generate ROC curve without external packages
generate_ROC_curve <- function(data, true_class_col, prob_col) {

  # Initialize vectors to store True Positive Rate (Sensitivity) and False Positive Rate (1-Specificity
  tpr_vector <- numeric()
  fpr_vector <- numeric()

  # Iterate through thresholds between 0 and 1 by .1 increments
  for (t in seq(0, 1, .1) ) {
      # Initialize variables
    tp <- 0  # True Positives
    fp <- 0  # False Positives
    n <- sum(data[[true_class_col]] == 0)  # Actual Class 0
```

```r
    p <- sum(data[[true_class_col]] == 1)  # Actual Class 1
      # Iterate through sorted data
    for (i in 1:nrow(data)) {

      if (data[[prob_col]][i] >= t) {      ## If Probability >= Threshold t
        if (data[[true_class_col]][i] == 1) {
          tp <- tp + 1
        }
        else {
          fp <- fp + 1
        }
      }
    }
          # Calculate True Positive Rate (Sensitivity) and False Positive Rate (1-Specificity)
    tpr <- tp / p
    fpr <- fp / n

    # Append to vectors
    tpr_vector <- c(tpr_vector, tpr)
    fpr_vector <- c(fpr_vector, fpr)
  }
  # Plot ROC curve
  plot(fpr_vector, tpr_vector, type = "l", col = "blue", lwd = 2,
       main = "ROC Curve", xlab = "False Positive Rate", ylab = "True Positive Rate")
  abline(a = 0, b = 1, col = "red", lty = 2)

  auc <- auc(fpr_vector, tpr_vector)
  return(auc)
}
generate_ROC_curve(df_classif, "class", "scored.probability")
```

```
## Warning in roc.default(response, predictor, auc = TRUE, ...): 'response' has
## more than two levels. Consider setting 'levels' explicitly or using
## 'multiclass.roc' instead
```

```
## Setting levels: control = 0, case = 0.00806451612903226
```

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




```
## Area under the curve: 1
```

AUC calculated from predicted classification using .5 threshold to compare later

AUC_calc will take any set of actual and predicted values and calculate the AUC (Area Under the Curve) using the True Positive Rate, True Negative, Concordance, Discordance, and Percent of Ties.

AUC= Concordance Between Pairs + 0.5 x Percent of Ties

```
actual <- ls_class
predicted <- ls_scr_class

AUC_calc <- function (actual, predicted){
  df <- data.frame(actual = actual, predicted = predicted)
  # Calculate total number of pairs to check - permutation of how many 1's and 0's exist in the actual
  totalPairs <- nrow(subset(df, actual == "1")) * nrow(subset(df, actual == "0"))
  # Calculate concordance = number of pairs where actual and predicted AGREE
  df <- df %>% mutate(agreement = ifelse(actual == predicted, 1, 0))
  # Calculate discordance = number of pairs where actual and predicted DISAGREE
  df <- df %>% mutate(disagreement = ifelse(actual != predicted, 1, 0))

  conc <- sum(df$agreement)

  sum(df$disagreement)
```

```
  conc <- c(vapply(ones$Predicted,
                  function(x) {
                        ((x > zeros$Predicted))
                        },
                      FUN.VALUE=logical(nrow(zeros))))
  concordance <- conc/nrow(df)
  discordance <- disc/totalPairs
  tiesPercent <- (1-concordance-discordance)
  AUC = concordance + 0.5*tiesPercent
  return(list("Concordance"=concordance, "Discordance"=discordance,
              "Tied"=tiesPercent, "AUC"=AUC))
}

auc(as.numeric(actual), as.numeric(predicted))
```

```
## Setting levels: control = 1, case = 2

## Setting direction: controls < cases

## Area under the curve: 0.7167
```

11) Classification Metrics

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

```
paste("CONFUSION MATRIX")
```

```
## [1] "CONFUSION MATRIX"
```

```
conf_mat(actual, predicted)
```

```
##          actual
## predicted   1   0
##         1  27   5
##         0  30 119
```

```
paste("True positives:",tp_calc(actual, predicted))
```

```
## [1] "True positives: 27"
```

```
paste("True negatives:",tn_calc(actual, predicted))
```

```
## [1] "True negatives: 119"
```

```
paste("False positives:",fp_calc(actual, predicted))
```

```
## [1] "False positives: 5"
```

```r
paste("False negatives:",fn_calc(actual, predicted))
```

```
## [1] "False negatives: 30"
```

```r
paste("Accuracy:",accuracy_calc(df_classif, "class", "scored.class"))
```

```
## [1] "Accuracy: 0.806629834254144"
```

```r
paste("Precision:",precision_calc(df_classif, "class", "scored.class"))
```

```
## [1] "Precision: 0.84375"
```

```r
paste("Classificaiton Error Rate:",class_error_rate(df_classif, "class", "scored.class"))
```

```
## [1] "Classificaiton Error Rate: 0.193370165745856"
```

```r
paste("Specificity:",specificity_calc(df_classif, "class", "scored.class"))
```

```
## [1] "Specificity: 0.959677419354839"
```

```r
paste("Sensitivity:",sensitivity_calc(df_classif, "class", "scored.class"))
```

```
## [1] "Sensitivity: 0.473684210526316"
```

```r
paste("F1:",f1(df_classif))
```

```
## [1] "F1: 0.606741573033708"
```

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?

As shown below, the functions we created to assess the data match the values given when using the function from the caret package.

```r
confusionMatrix(data=ls_scr_class, reference = ls_class)
```

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

```
##                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
##
```

13) Investigate the pROC package

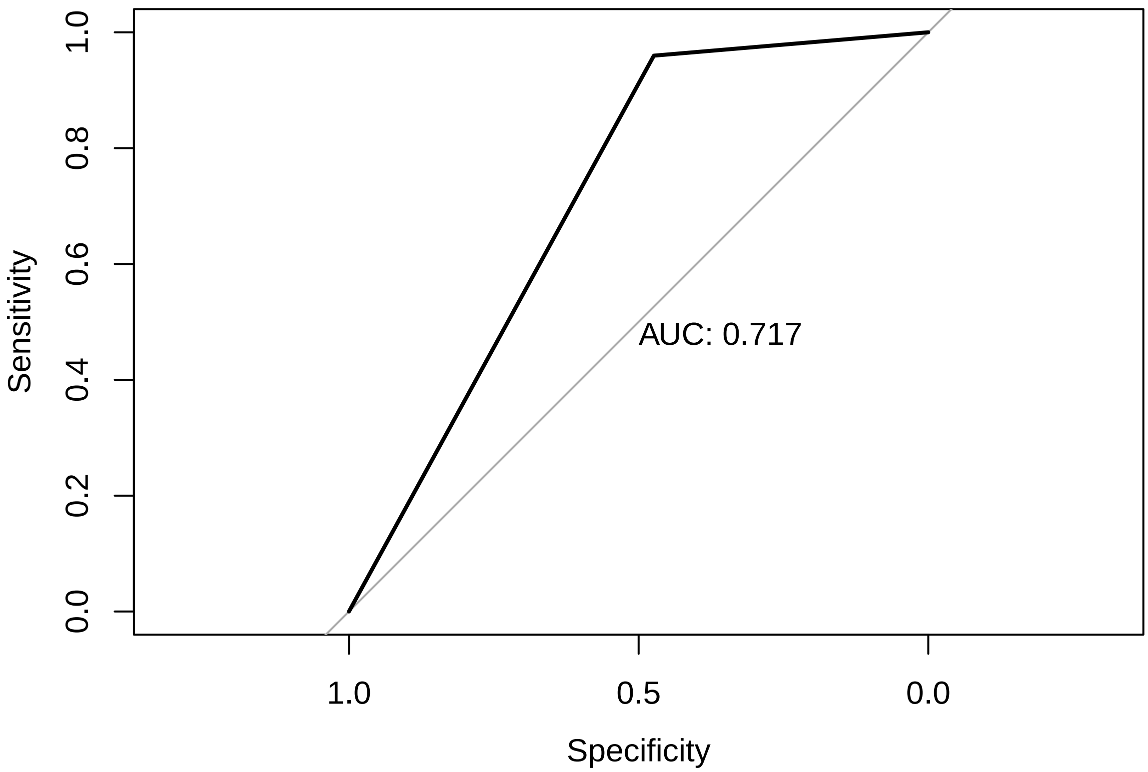
Use it to generate an ROC curve for the data set. How do the results compare with your own functions?

Seen below is the ROC curve when the threshold is .5 plotting actual (class) and predicted (scored.class) values, whereas our ROC curve created in number 10 explored different thresholds between 0 and 1 incrementing by .1 by using the scored.probability. This is why both the AUC and the ROC curve plot differ.

```r
roc(as.numeric(actual), as.numeric(predicted), plot = TRUE, print.auc = TRUE)
```

```
## Setting levels: control = 1, case = 2
```

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




```
##
## Call:
## roc.default(response = as.numeric(actual), predictor = as.numeric(predicted),     plot = TRUE, print
##
## Data: as.numeric(predicted) in 57 controls (as.numeric(actual) 1) < 124 cases (as.numeric(actual) 2)
## Area under the curve: 0.7167
```