DATA 621 Business Analytics and Data Mining

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

Last edited October 15, 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 $0 <$ < 1 and $0 <$ < 1 then $<$.)

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

Load the data

```
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)
```

Confusion Matrix

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

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 (Event = 1, but it was putting Nonevent = 0 first), then you've flipped the TP, TN, FN, and FP. *If you do not relevel 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**.

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

2. 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?

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
```

Accuracy Function

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. $Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$ **This was (1) not written as a function and (2) was not written to take a dataframe as the sole input. This has been corrected.**

```
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
```

Classification Error Rate Function

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. **See comment on Q3**

```
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
```

Precision Function

$Precision = \frac{TP}{TP+FP}$ 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. **Same comment**

```
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
```

Sensitivity Function

Sensitivity+$\frac{TP}{TP+FN}$ 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. **Same comment**

```
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
```

Specificity Function

Specifity=$\frac{TN}{TN+FN}$ 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. **Same comment**

```
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
```

F1 score Function

$F1\ Score = \frac{2 \times Precision \times Sensitivity}{Presicion + Sensitivity}$ 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. **Same comment**

```
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. 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 < .)

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))
```

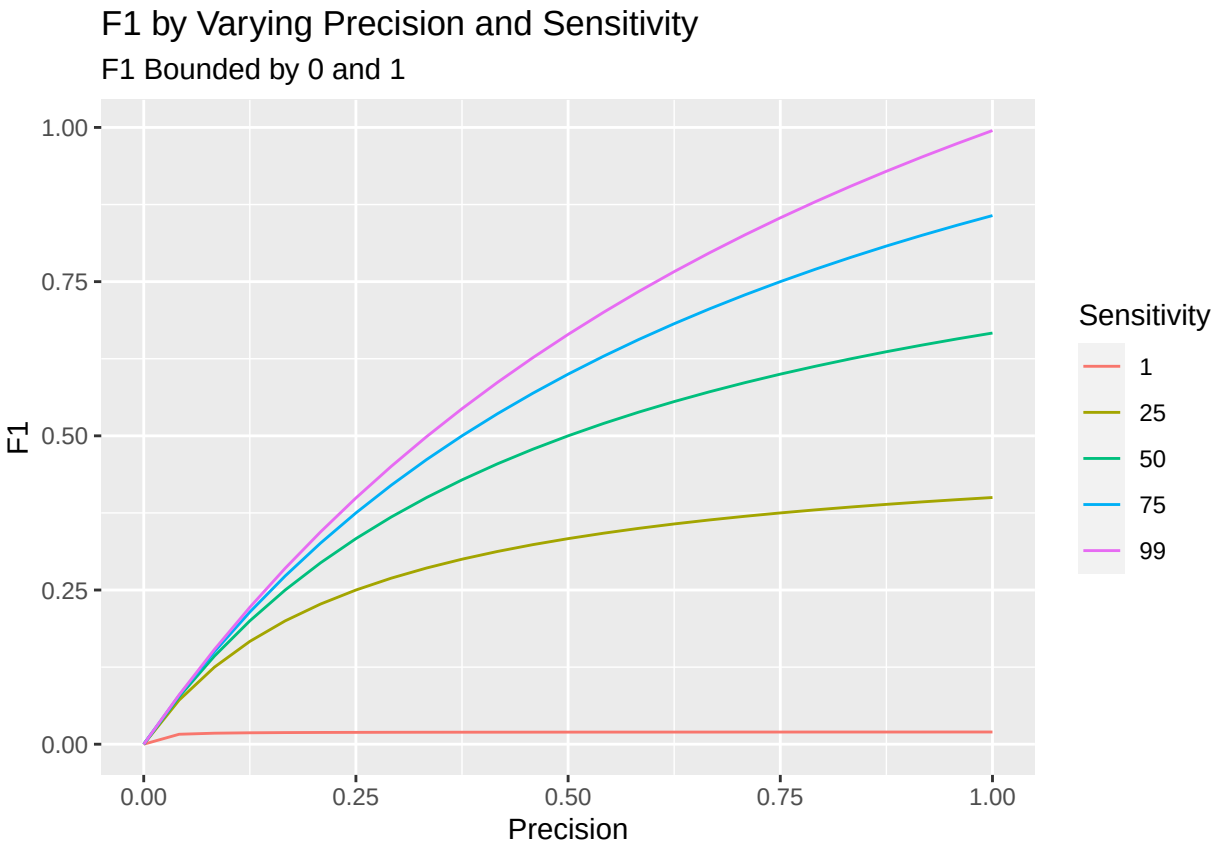
Step 2. Create a sequence of precision values and calculate f1 when sensitivity equals 1, 25, 75, and 99%

```
# 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 Bou
```



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.

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))))
  # disc <- sum(c(vapply(ones$Predicted,
  #                      function(x) {(x < zeros$Predicted)},
  #                      FUN.VALUE = logical(nrow(zeros)))), na.rm = T)
  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
```

```
nrow(df_classif)
```

```
## [1] 181
```

11. 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:")
```

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

```
tp_calc(actual, predicted)
```

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

```
paste("True negatives:")
```

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

```
tn_calc(actual, predicted)
```

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

```
paste("False positives:")
```

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

```
fp_calc(actual, predicted)
```

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

```
paste("False negatives:")
```

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

```
fn_calc(actual, predicted)
```

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

```
paste("Accuracy:")
```

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

```
accuracy_calc(df_classif, "class", "scored.class")
```

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

```
paste("Precision:")
```

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

```
precision_calc(df_classif, "class", "scored.class")
```

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

```
paste("Calssificaiton Error Rate:")
```

```
## [1] "Calssificaiton Error Rate:"
```

```
class_error_rate(df_classif, "class", "scored.class")
```

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

```
paste("Specificity:")
```

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

```
specificity_calc(df_classif, "class", "scored.class")
```

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

```
paste("Sensitivity:")
```

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

```
sensitivity_calc(df_classif, "class", "scored.class")
```

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

```
paste("F1:")
```

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

```
f1(df_classif)
```

```
## [1] 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?

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

```
## 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
##
```

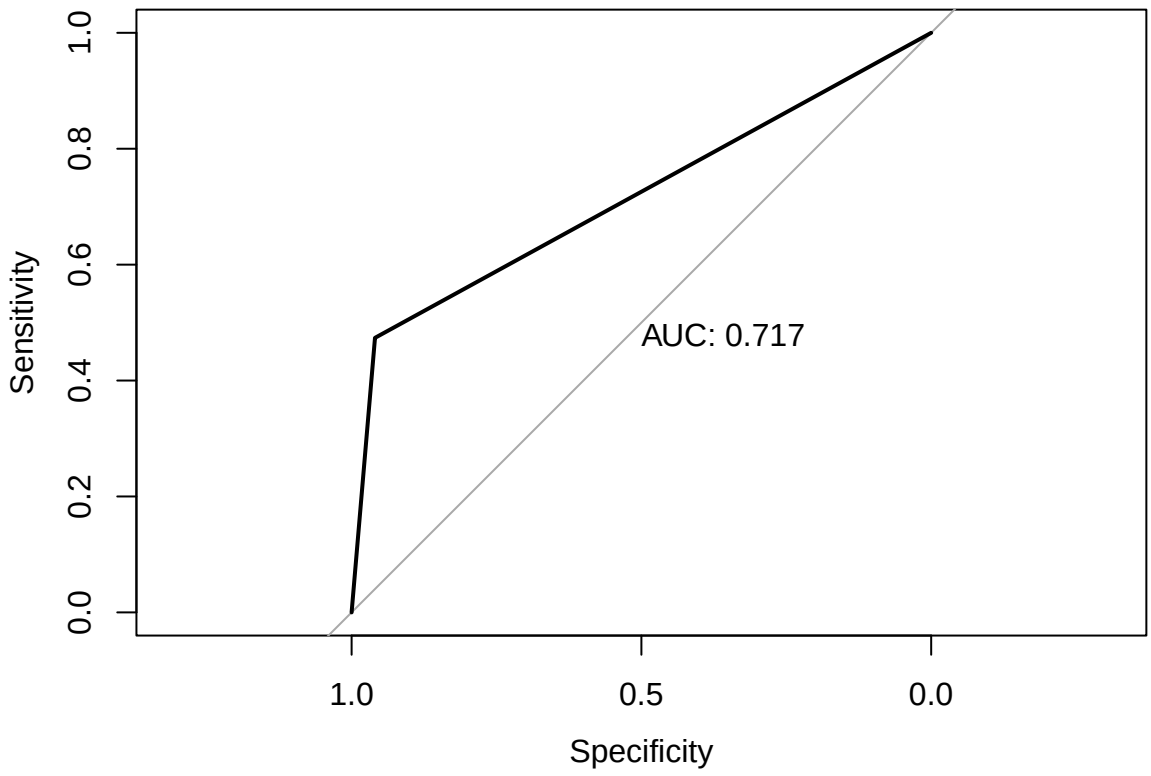
```
##                  Kappa : 0.4916
##
##  Mcnemar's Test P-Value : 4.976e-05
##
##             Sensitivity : 0.9597
##             Specificity : 0.4737
##          Pos Pred Value : 0.7987
##          Neg Pred Value : 0.8438
##              Prevalence : 0.6851
##          Detection Rate : 0.6575
##    Detection Prevalence : 0.8232
##       Balanced Accuracy : 0.7167
##
##        'Positive' Class : 0
##
```

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?

```
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 124 controls (as.numeric(actual) 1) < 57 cases (as.numeric(actual) 2)
## Area under the curve: 0.7167
```