# Home Work Assignment - 02

*Critical Thinking Group 5*
*Arindam Barman*
*Mohamed Elmoudni*
*Shazia Khan*
*Kishore Prasad*

## Contents

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

```
fp <- "https://raw.githubusercontent.com/kishkp/data621-ctg5/master/HW2/classification-output-data.csv"
class_data <- read.csv(fp)
```

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?

```
cm<- with(class_data, table("Actual"=class, "Predicted"= scored.class))

if (rownames(cm)[1]==0 ) (rownames(cm)[1]='NO')
if (rownames(cm)[2]==1)  (rownames(cm)[2]='YES')
if (colnames(cm)[1]==0 ) (colnames(cm)[1]='NO')
if (colnames(cm)[2]==1 ) (colnames(cm)[2]='YES')
```

```
(ftable(cm, quote = FALSE, method="row.compact"))
```

```
##        Predicted  NO YES
## Actual
## NO               119   5
## YES               30  27
```

Since we have given the table command as table(class_data$class, $class_data$scored.class), in the output above:
- Rows represent the Actual class
- Columns represent predicted class

**Explanation:**

There are two possible predicted classes: "yes" and "no". The classifier made a total of 181 predictions (e.g., 181 were being tested for the presence of that disease, in this case diabetes). Out of those 181 cases, the classifier predicted "yes" 32 times, and "no" 149 times. In reality, 57 patients in the sample have the disease, and 124 patients do not.

In addition:
• 119 represents true negatives (TN) where we predicted no, and they don't have the disease.
• 27 represents true positives (TP): These are cases in which we predicted yes (they have the disease), and they do have the disease.
• 30 represents false negatives (FN): We predicted no, but they actually do have the disease.
• 5 represents false positives (FP): We predicted yes, but they don't actually have the disease.

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.

```
get_truth_table <- function(ds, actual, predicted) {
    # check if object exists
    if (!(exists("ds")))   return("Object not found")

    #check if object is data frame
    if (!(is.data.frame(ds))) return("Invalid datatype: Not a data frame")

    #check if data frameis empty
    if (is.data.frame(ds) && nrow(ds)==0)
        return("dataset is Empty")
    else
        cm <- table(select(ds, get(actual), get(predicted)))   # ds not class_data?

    return (list(TP = cm[2,2], TN = cm[1,1], FN = cm[2,1], FP = cm[1,2]))
}


accuracy <- function(ds, actual, predicted) {

    tt <- get_truth_table(ds, actual, predicted)

    TP <- tt$TP
    TN <- tt$TN
    FN <- tt$FN
    FP <- tt$FP

    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.

```
class_err_rate <- function(ds, actual, predicted) {

    tt <- get_truth_table(ds, actual, predicted)

    TP <- tt$TP
    TN <- tt$TN
    FN <- tt$FN
```

```
    FP <- tt$FP
    return ((FP + FN) / (TP + FP + TN + FN))
}
```

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

**The sum of accuracy and error rate is as below:** 1

```
accuracy(class_data, "class", "scored.class") + class_err_rate(class_data, "class", "scored.class")
```

```
## [1] 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 <- function(ds, actual, predicted) {
    tt <- get_truth_table(ds, actual, predicted)

    TP <- tt$TP
    TN <- tt$TN
    FN <- tt$FN
    FP <- tt$FP

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

Testing the precision function: 0.84375

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 <- function(ds, actual, predicted) {

    tt <- get_truth_table(ds, actual, predicted)

    TP <- tt$TP
    TN <- tt$TN
    FN <- tt$FN
    FP <- tt$FP

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

Testing the sensitivity function: 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 <- function(ds, actual, predicted) {

    tt <- get_truth_table(ds, actual, predicted)

    TP <- tt$TP
    TN <- tt$TN
    FN <- tt$FN
    FP <- tt$FP

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

Testing the specifity function: 0.9596774

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 <- function(ds, actual, predicted) {

    tt <- get_truth_table(ds, actual, predicted)

    TP <- tt$TP
    TN <- tt$TN
    FN <- tt$FN
    FP <- tt$FP

    pre <- (TP / (TP + FP))
    sen <- (TP / (TP + FN))

    return ((2*pre*sen) / (pre+sen))
}
```

Testing the F1 score function: 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 $0 < a < 1$ and $0 < b < 1$ then $ab < 0$.)

We have:

$$precision = \frac{TP}{TP + FP} \qquad Eq.1$$

$$sensitivity = \frac{TP}{TP + FN} \qquad Eq.2$$

From Eq.1, let's find the bounds of the precision relative to TP:
First, we will find the bound when TP approaches 0:

$$\lim_{TP \to 0} \frac{TP}{TP + FP} = \frac{0}{0 + FP}$$
$$\lim_{TP \to 0} \frac{TP}{TP + FP} = 0$$

Then, we will find the bound when TP approaches infinity:

$$\lim_{TP \to \infty} \frac{TP}{TP + FP} = \frac{\frac{TP}{TP}}{\frac{TP}{TP} + \frac{FP}{TP}}$$
$$\lim_{TP \to \infty} \frac{TP}{TP + FP} = \frac{1}{1 + \frac{FP}{TP}}$$
$$\lim_{TP \to \infty} \frac{TP}{TP + FP} = \frac{1}{1 + 0}$$
$$\lim_{TP \to \infty} \frac{TP}{TP + FP} = 1$$

from Eq.1, let's find the bounds of the precision relative to FP:
First, we will find the bound when FP approaches 0:

$$\lim_{FP \to 0} \frac{TP}{TP + FP} = \frac{TP}{0 + TP}$$
$$\lim_{FP \to 0} \frac{TP}{TP + FP} = 1$$

Then, we will find the bound when FP approaches infinity:

$$\lim_{FP \to \infty} \frac{TP}{TP + FP} = \frac{\frac{TP}{FP}}{\frac{TP}{FP} + \frac{FP}{FP}}$$
$$\lim_{FP \to \infty} \frac{TP}{TP + FP} = \frac{0}{0 + \frac{FP}{FP}}$$
$$\lim_{FP \to \infty} \frac{TP}{TP + FP} = \frac{0}{0 + 1}$$
$$\lim_{FP \to \infty} \frac{TP}{TP + FP} = 0$$

Hence, the precision $\frac{TP}{TP+FP}$ has bounds of [0,1]

Please note that following the steps above used for the precision, we can deduce that the sensitivity $\frac{TP}{TP+FN}$ also has bounds of [0,1]

Therefore, we can state the following:

$$0 \leq precision \leq 1$$

$$0 \leq sensitivity \leq 1$$

$$0 \leq precision * sensitivity \leq 1 * sensitivity$$

$$0 \leq 2 * precision * sensitivity \leq 2 * sensitivity \qquad Eq.3$$

Also, since the max sensitivity is 1 and max precision is 1 as shown above, we can state:

$$0 \leq precision + sensitivity \leq 2 \quad Eq.4$$

Now divide Eq.3 by Eq.4 we get:

$$0 \leq (2 * precision * sensitivity)/(precision + sensitivity) \leq 2 * sensitivity/2$$
$$0 \leq (2 * precision * sensitivity)/(precision + sensitivity) \leq sensitivityqquad Eq.5$$

We know that the sensitivity max value 1. Hence, Eq.5 has a lower bound of 0 and upper bound of 1.

Therefore, 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.

```
roc_curve <- function(ds, actual, pred.prob, threshold) {
    true_positive_rate <- function(ds, actual, pred.prob, threshold) {
        sum(select(class_data, get(pred.prob)) < threshold &
                select(class_data, get(actual)) == 0) / sum(select(class_data, get(actual)) == 0)
    }

    false_positive_rate <- function(ds, actual, pred.prob, threshold) {
        sum(select(class_data, get(pred.prob)) < threshold &
                select(class_data, get(actual)) == 1) / sum(select(class_data, get(actual)) == 1)
    }

    roc <- data.frame(threshold = seq(0,1, by = 0.01), tpr=NA, fpr=NA)
    roc$tpr <- sapply(roc$threshold, function(th) true_positive_rate(ds, actual, pred.prob, th))
    roc$fpr <- sapply(roc$threshold, function(th) false_positive_rate(ds, actual, pred.prob, th))

    roc$next_fpr <- lead(roc$fpr,1)
    roc$area_curve <- (roc$next_fpr-roc$fpr) * roc$tpr


    idx_threshold = which.min(abs(roc$threshold-0.5))

    p_roc <- ggplot(roc, aes(fpr,tpr)) +
    geom_line(color=rgb(0,0,1,alpha=0.3)) +
    coord_fixed() +
    geom_line(aes(threshold,threshold), color=rgb(0,0,1,alpha=0.5)) +
    labs(title = sprintf("ROC")) + xlab("FPR") + ylab("TPR") +
    geom_hline(yintercept=roc[idx_threshold,"tpr"], alpha=0.5, linetype="dashed") +
    geom_vline(xintercept=roc[idx_threshold,"fpr"], alpha=0.5, linetype="dashed")
```

```
    lst_roc_auc <- list(p_roc, sum(roc$area_curve, na.rm = TRUE))
    return(lst_roc_auc)
}
```

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

Below are the requested details using our functions:

- The Accuracy of the provided dataset is : 0.8066298
- The Classification Error Rate of the provided dataset is : 0.1933702
- The precision of the provided dataset is : 0.84375
- The Sensitivity of the provided dataset is : 0.4736842
- The Specificity of the provided dataset is : 0.9596774
- The F1 Score of the provided dataset is : 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?

The caret package (short for Classification And REgression Training) is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for:

- data splitting
- pre-processing
- feature selection
- model tuning using resampling
- variable importance estimation
- as well as other functionality.

Below, lets compare the output of caret package with our built functions:
First lets look at the confusion matrix generated by the caret package:

```
#Reference: http://topepo.github.io/caret/other.html

#Caret Confusion Matrix
ccm <- confusionMatrix(class_data$scored.class, class_data$class, positive='1')

#Information from Confusion matrix
#ccm

#Confusion matrix table
ccm$table
```

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

This table is similar to the one generated by us.

Please Note: The caret package gives a transposed output of the our confusion matrix. So we have the predicted on the rows and the actuals on the columns in the caret version of confusion matrix.

Next we look at the values for sensitivity and specificity generated from the caret package. Below are the values for the same:

```
#Caret Sensitivity
ccm$byClass["Sensitivity"]
```

```
## Sensitivity
##   0.4736842
```

```
#Caret Specificity
ccm$byClass["Specificity"]
```

```
## Specificity
##   0.9596774
```

Lets see how our function fares with these values. Below are the values generated by our functions:
- The Sensitivity of the provided dataset is :

```
sensitivity(class_data, "class", "scored.class")
```

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

- The Specificity of the provided dataset is :

```
specificity(class_data, "class", "scored.class")
```

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

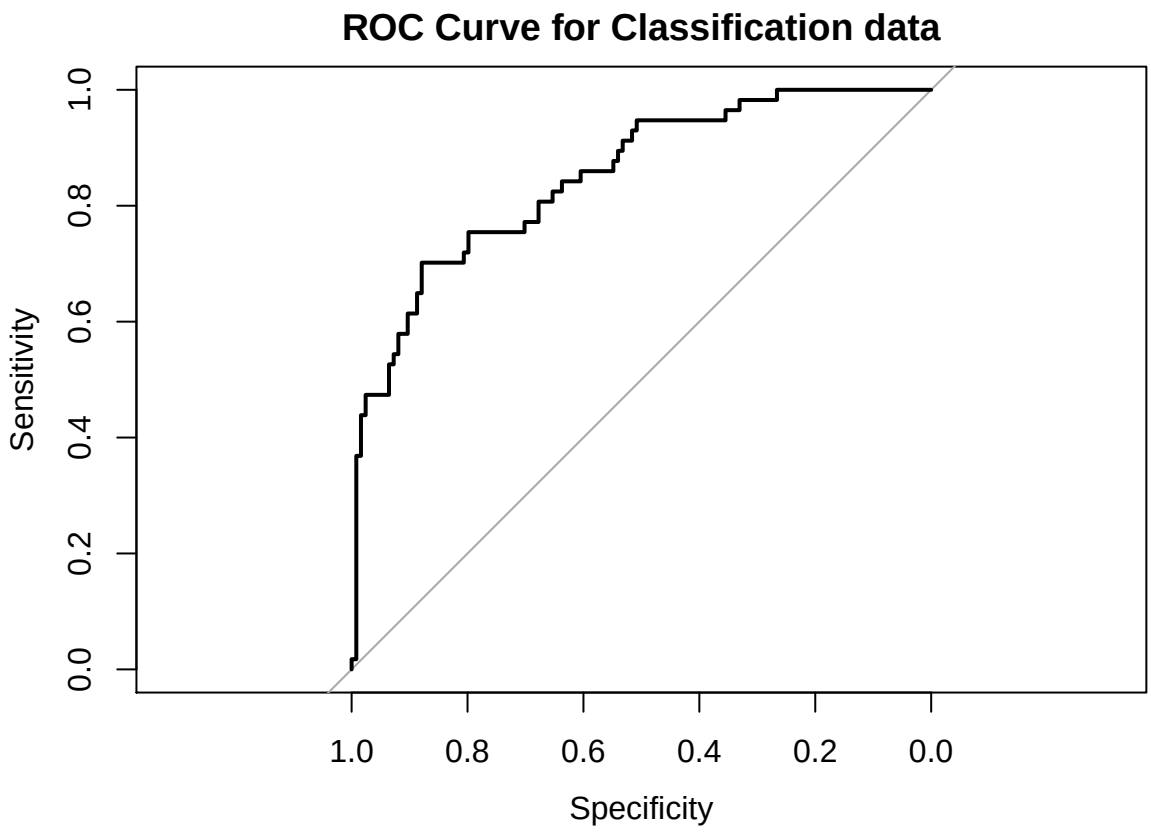
We see that it matches well with the caret package.

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?

Below are the roc curve generated by the pROC package along with the auc for the same:

```
#library(pROC)

#plot myRoc
myRoc <- roc(class~scored.probability, class_data)
plot(myRoc, main="ROC Curve for Classification data")
```




```
##
## Call:
## roc.formula(formula = class ~ scored.probability, data = class_data)
##
## Data: scored.probability in 124 controls (class 0) < 57 cases (class 1).
## Area under the curve: 0.8503
```

```
#Area under the curve
myRoc$auc
```

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

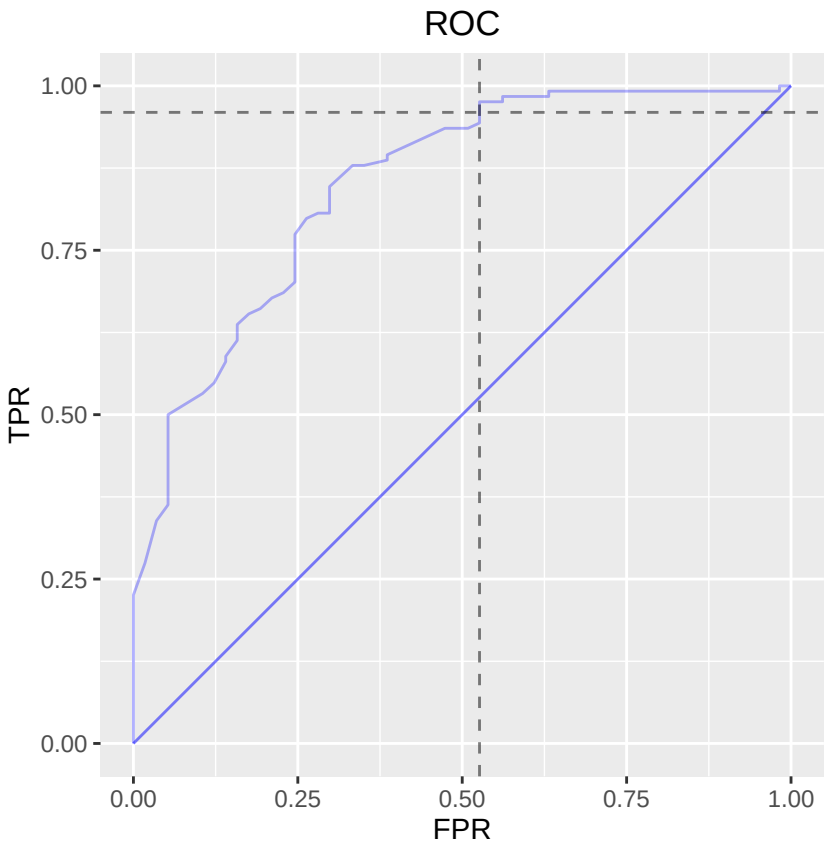
Next, lets check the roc curve and auc generated by our functions:

```
ds<- class_data
actual <- "class"
pred.prob   <- "scored.probability"
threshold <- 0.5

# output from our function
a<-roc_curve(ds, actual, pred.prob, 0.5)

# ROC curve
a[1]
```

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




```
# auc value
a[2]
```

```
## [[1]]
## [1] 0.8438031
```

```
# output from built in functon
#plot(roc(class_data$class, class_data$scored.probability))
```

We see that our ROC curve, is a little rough edged and not quite as smooth as the pROC output. Also, our auc value is a little lower than the pROC value.

The reason for this difference is that we have used an interval of 0.01 on the sequence of thresholds from 0 to 1. If we were to decrease this to 0.001 or 0.0001, then we will have a better estimate and curve. The more we decrease the interval the closer it gets to the pROC output. However, this will also take toll on the time taken to run our function.