Data-621 Homework-2

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

- **Load Data**

```
input.df = read.csv("D:/MSDS/MSDSQ4/Data621/HW-2/classification-output-data.csv", stringsAsFacto
rs = FALSE)
```

- **View Data**

```
head(input.df)
```

```
##   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
##   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
```

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?

```
conf.mat = table(input.df$class, input.df$scored.class)
conf.mat
```

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

- **Above table shows the confusion matrix. In the above table rows represents the Actual class and columns represent the predicted class**

- **Create a dataframe for above metrics**

```
create.metrics = function(actclass, predclass)
{
  TN = sum(actclass == 0 & predclass == 0)
  FP = sum(actclass == 0 & predclass == 1)
  FN = sum(actclass == 1 & predclass == 0)
  TP = sum(actclass == 1 & predclass == 1)
  metrics.df = data.frame(TN=TN, FN = FN, TP = TP, FP = FP)
  return(metrics.df)

}

metrics.df = create.metrics(input.df[,'class'], input.df[,'scored.class'])
```

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)

- **Function to calculate Accuracy**

```
calc.accuracy = function(df)
{
  Accuracy = (df$TP + df$TN)/ (df$TP + df$FP + df$TN + df$FN)
  return(Accuracy)
}

print(paste('Accuracy = ', calc.accuracy(metrics.df)))
```

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

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)

- **Function to calculate Classification Error**

```
calc.error = function(df)
{
  Error = (df$FP + df$FN)/ (df$TP + df$FP + df$TN + df$FN)
  return(Error)
}

print(paste('Classification Error = ', calc.error(metrics.df)))
```

```
## [1] "Classification Error =  0.193370165745856"
```

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

```
print(calc.accuracy(metrics.df) + calc.error(metrics.df))
```

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

- **Above calculation proves that Accuracy and Error rate 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 = TP/(TP + FP)

```
calc.precision = function(df)
{
  Precision = (df$TP)/ (df$TP + df$FP)
  return(Precision)
}

print(paste('Precision = ', calc.precision(metrics.df)))
```

```
## [1] "Precision =  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 = TP/(TP + FN)

```
calc.sensitivity = function(df)
{
  Sensitivity = (df$TP)/ (df$TP + df$FN)
  return(Sensitivity)
}

print(paste('Sensitivity = ', calc.sensitivity(metrics.df)))
```

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

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)

```
calc.specificity = function(df)
{
  Specificity = (df$TN)/ (df$TN + df$FP)
  return(Specificity)
}

print(paste('Specificity = ', calc.specificity(metrics.df)))
```

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

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 * Precision * Sensitivity)/ (Precision + Sensitivity)

```
calc.f1score = function(df)
{
  precision = calc.precision(df)
  sensitivity = calc.sensitivity(df)
  f1.score = (2 * precision * sensitivity)/(precision + sensitivity)
  return(f1.score)
}

print(paste('F1 Score = ', calc.f1score(metrics.df)))
```

```
## [1] "F1 Score =  0.606741573033708"
```

9. Before we move on, let's consider a question that was asked: What are the bounds on the F1 score? Showthat the F1 score will always be between 0 and 1.

- **F1 score is calculated based on Precision and Sensitivity. Precision is nothing but how many are true positives out of identified positives and sensitivity is nothing but how many true positives are identified out of total available positives. Based on these definitions and above formulae we can conclude that both precision and sensitivity values can range from 0 to 1**
- **Below is the simple simulation of F1 score range when precision and sensitivity values varies from 0 to 1**

```
precision = c(0.001, 0.1, 0.5, 0.9, 1)
recall = c(0.001, 0.1, 0.5, 0.9, 1)

f1.score = (2*precision * recall)/(precision + recall)
print(f1.score)
```

```
## [1] 0.001 0.100 0.500 0.900 1.000
```

- **Above simulation shows that when precision and sensitivity values varies from 0 to 1 F1 Score takes a range of values from 0 to 1. From the above simulation we can conclude that F1 score value can be within 0 to 1 range, with 0 indicating poor model fit and 1 indicating best model fit**

10. Write a function that generates an ROC curve from a data set with a true classification column (class in ourexample) 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.

```
simple_auc <- function(TPR, FPR){
  # inputs already sorted, best scores first
  dFPR <- c(diff(FPR), 0)
  dTPR <- c(diff(TPR), 0)
  auc = sum(TPR * dFPR) + sum(dTPR * dFPR)/2
  return(abs(auc))
}
calc.roc = function()
{
  library(ggplot2)
  threshold = seq(0,1,0.01)
  class = input.df$class
  spec = c()
  sens = c()
  for(t in threshold)
  {
    scored.class = ifelse(input.df$scored.probability > t, 1, 0)
    df = data.frame(class = class, scored.class = scored.class)

    metrics.df = create.metrics(df[,'class'], df[,'scored.class'])
    spec = c(spec, calc.specificity(metrics.df))
    sens = c(sens, calc.sensitivity(metrics.df))
  }

  plt = ggplot2::qplot(1-spec, sens, xlim = c(0, 1), ylim = c(0, 1),
    xlab = "false positive rate", ylab = "true positive rate", geom='line')
   auc = simple_auc(sens, 1-spec)
   return (list(plt, auc))
}

lst = calc.roc()
```

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

- **Accuracy**

```
print(paste('Accuracy = ', calc.accuracy(metrics.df)))
```

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

- **Classification Error**

```
print(paste('Classification Error = ', calc.error(metrics.df)))
```

```
## [1] "Classification Error =  0.193370165745856"
```

- **Precision**

```
print(paste('Precision = ', calc.precision(metrics.df)))
```

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

- **Sensitivity**

```
print(paste('Sensitivity = ', calc.sensitivity(metrics.df)))
```

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

- **Specificity**

```
print(paste('Specificity = ', calc.specificity(metrics.df)))
```

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

- **F1 Score**

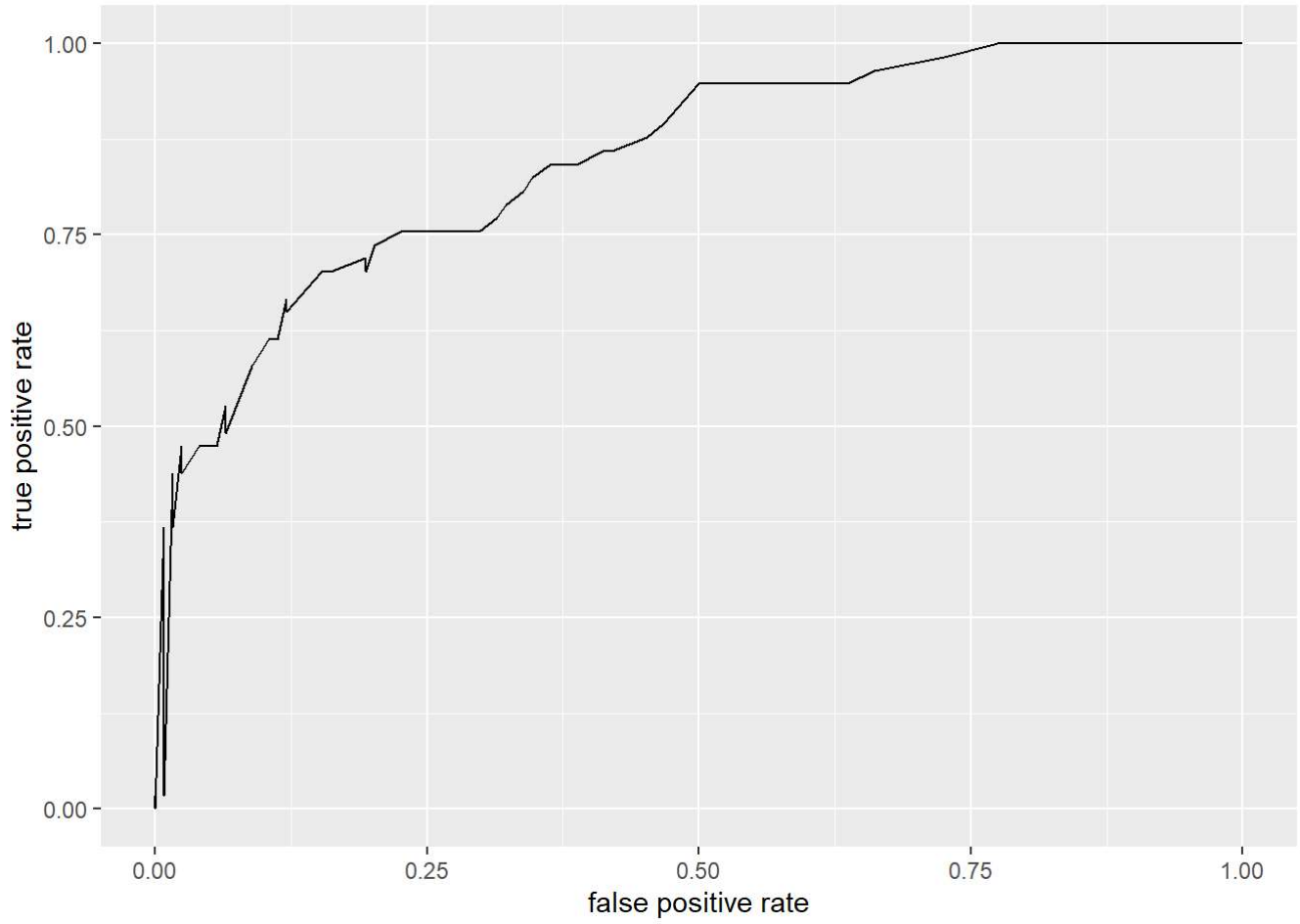
```
print(paste('F1 Score = ', calc.f1score(metrics.df)))
```

```
## [1] "F1 Score =  0.606741573033708"
```

- **ROC Curve**

```
calc.roc()[1]
```

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



- **AUC**

```
print(paste('AUC = ', calc.roc()[2]))
```

```
## [1] "AUC =  0.848896434634974"
```

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?

```
library(caret)
```

```
## Loading required package: lattice
```

```
confusionMatrix(as.factor(input.df$scored.class), as.factor(input.df$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
##
```

```
sensitivity(as.factor(input.df$scored.class), as.factor(input.df$class), positive = 1)
```

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

```
specificity(as.factor(input.df$scored.class), as.factor(input.df$class), negative=0)
```

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

We can see that above results from caret package calls matches exactly with our own function call. No difference is identified between results obtained from caret package and our own function call for important metrics like confusionmatrix, sensitivity and specificity

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?

```
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 3.6.1
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

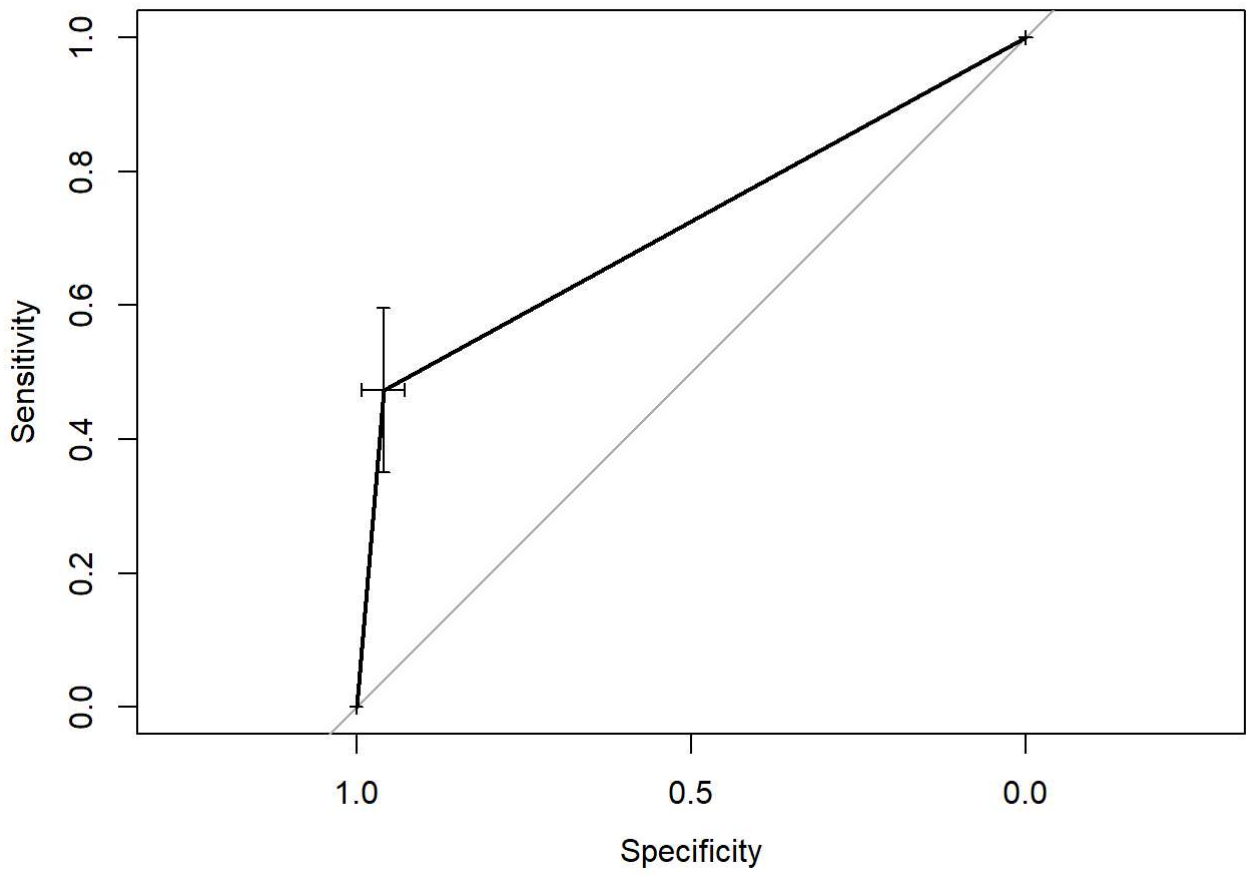
```
roccurve = roc(input.df$class, input.df$scored.class, ci=TRUE, of="thresholds")
```

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

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

```
## Warning in coords.roc(roc, x = thresholds, input = "threshold", ret
## = "threshold", : An upcoming version of pROC will set the 'transpose'
## argument to FALSE by default. Set transpose = TRUE explicitly to keep the
## current behavior, or transpose = FALSE to adopt the new one and silence
## this warning. Type help(coords_transpose) for additional information.
```

```
plot.roc(roccurve)
```



```
auc(roccurve)
```

```
## Area under the curve: 0.7167
```

ROC obtained from pROC package looks little diffrent than ROC obtained from our own function. Area under curve is also slightly different. AUC returned from our own function is 0.84 and AUC returned from pROC package is 0.71

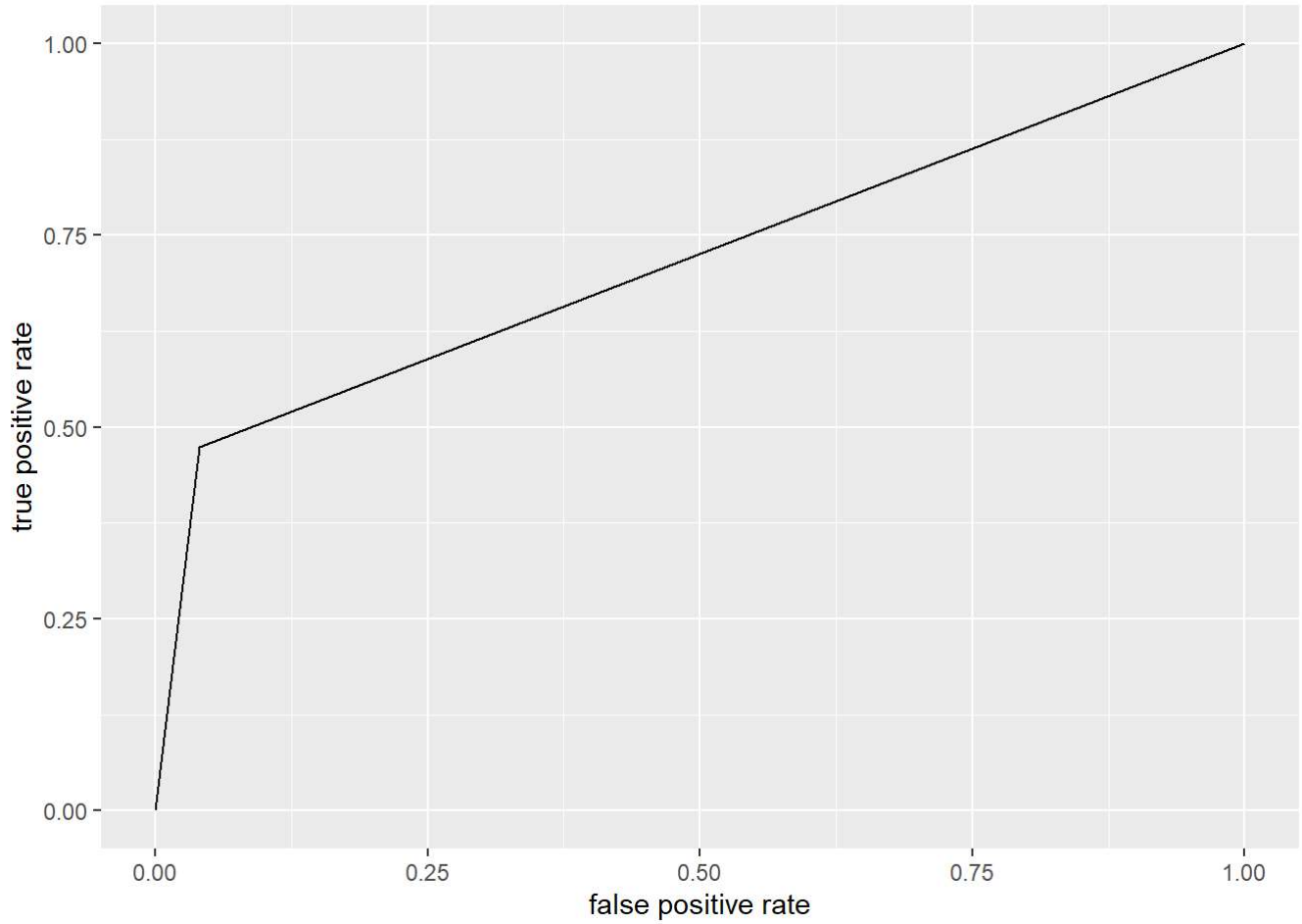
After further exploration, we found that this difference is due to different threshold increment values. When we modify our function to use higher interval threshold values (min-0, max-1, increment-0.5) the ROC curve returned from our function matches exactly with pROC package. Even the AUC value is exactly same between our own function and pROC package after threchold increment is tweaked

ROC plot obtained from our own function after tweaking threshold increments

```
calc.roc = function()
{
  library(ggplot2)
  threshold = seq(0,1,0.5)
  class = input.df$class
  spec = c()
  sens = c()
  for(t in threshold)
  {
    scored.class = ifelse(input.df$scored.probability > t, 1, 0)
    df = data.frame(class = class, scored.class = scored.class)

    metrics.df = create.metrics(df[,'class'], df[,'scored.class'])
    spec = c(spec, calc.specificity(metrics.df))
    sens = c(sens, calc.sensitivity(metrics.df))
  }

  plt = ggplot2::qplot(1-spec, sens, xlim = c(0, 1), ylim = c(0, 1),
    xlab = "false positive rate", ylab = "true positive rate", geom='line')
   auc = simple_auc(sens, 1-spec)
   return (list(plt, auc))
}

lst = calc.roc()
```

```
lst[1]
```

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



AUC value obtained from our own function after tweaking threshold increments

```
lst[2]
```

```
## [[1]]
## [1] 0.7166808
```