DATA 621 – Business Analytics and Data Mining

Homework 2: Critical Thinking Group 2

E.Azrilyan, C.Joseph, M.Kivenson, S.Mehta, V.Patel

2020-03-14

Step 1.

Download the classification output data set.

```
df <- read.csv("https://raw.githubusercontent.com/che10vek/Data621/master/classification-output-data.csv
kable(head(df,10), booktabs = T)
```

pregnant	glucose	diastolic	skinfold	insulin	bmi	pedigree	age	class	scored.class	scored.probability
7	124	70	33	215	25.5	0.161	37	0	0	0.3284523
2	122	76	27	200	35.9	0.483	26	0	0	0.2731904
3	107	62	13	48	22.9	0.678	23	1	0	0.1096604
1	91	64	24	0	29.2	0.192	21	0	0	0.0559984
4	83	86	19	0	29.3	0.317	34	0	0	0.1004907
1	100	74	12	46	19.5	0.149	28	0	0	0.0551546
9	89	62	0	0	22.5	0.142	33	0	0	0.1071154
8	120	78	0	0	25.0	0.409	64	0	0	0.4599474
1	79	60	42	48	43.5	0.678	23	0	0	0.1170237
2	123	48	32	165	42.1	0.520	26	0	0	0.3153632

Step 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?

```
cmtable<-table(df$class, df$scored.class)
cmtable
```

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

Interpreting the output of this Confusion Matrix: The rows represent Actual Values, and the columns respesent Predicted Values. Let's rename the rows and columns to to make it clearer.

```
colnames(cmtable) <- c("Predicted No", "Predicted Yes")
rownames(cmtable) <- c("Actual No", "Actual Yes")
kable(cmtable, booktabs = T)
```

	Predicted No	Predicted Yes
Actual No	119	5
Actual Yes	30	27

Step 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_accuracy <- function(actual, predicted)
{
  cmtable <- as.matrix(table(predicted, actual))
  TN <- cmtable[1,1]
  FN <- cmtable[1,2]
  FP <- cmtable[2,1]
  TP <- cmtable[2,2]
  return ((TP + TN) / (TN + FN + TP + FP))
}
get_accuracy(df$class, df$scored.class)
```

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

Step 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.

```
get_class_err_rate <- function(actual, predicted)
{
  cmtable <- as.matrix(table(predicted, actual))
  TN <- cmtable[1,1]
  FN <- cmtable[1,2]
  FP <- cmtable[2,1]
  TP <- cmtable[2,2]
  return ((FP + FN) / (TN + FN + TP + FP))
}
get_class_err_rate(df$class, df$scored.class)
```

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

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

Note: Accuracy is used when the dataset is balanced. It means the dataset has equal number of classes. if the dataset is imbalanced, we have to use other metrics like precision, recall and f1 score.

```r
get_accuracy(df$class, df$scored.class) + get_class_err_rate(df$class, df$scored.class)
```

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

Step 5.

Question 5 - Precision

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

Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the precision of the predictions.

Note: False positive rate of the classifer is otherwise called Type-1 Error.

$FalsePositiveRate = \frac{FP}{FP+TN}$

E.g of type1 error: Suppose a spam detection model predicted as spam, but in reality it is not. The impact of such event is customer is going to miss the email.

In such use cases, we should focus on reducing false positive rate and we use the metric Precision to measure this. For precision, the more closer to 1, the better would be the tolerance of the classifer towards the false positive rate.

```r
get_precision <- function(actual, predicted){
  cm <- as.matrix.data.frame(table(predicted, actual))
  TN <- cm[1,1]
  FN <- cm[1,2]
  FP <- cm[2,1]
  TP <- cm[2,2]
  return (TP / (TP + FP))
}
prec<- get_precision(df$class, df$scored.class)
prec
```

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

Step 6.

Question 6 - Sensitivity (Recall) 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.

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

Note: Sensitivity/Recall indicate the False negative rate of the classifer which is otherwise called Type-2 error.

e.g of type2 error: $FalseNegativeRate = \frac{FN}{FN+TP}$

E.g of type1 error: A person has a cancer in reality, but the model predicted as false. The impact of such event is bad and patient will die without the treatment.

In such use cases, we should focus on reducing false negative rate and we use the metric Recall or Sensitivityto measure this. For Sensitivity, the more closer to 1, the better would be the tolerance of the classifer towards the false Negative rate.

The goal is to reduce both Type1 and Type2 error. Hence we have another metric called "F-beta" Score to combine both of these metrics. beta is value chosen depends on tolerance of how much False positive and False negative rate is allowed.

F1 Score is calculated when beta is 1.

```
get_sensitivity <- function(actual, predicted){
  cm <- as.matrix.data.frame(table(predicted, actual))
  TN <- cm[1,1]
  FN <- cm[1,2]
  FP <- cm[2,1]
  TP <- cm[2,2]
  return (TP / (TP + FN))
}

recall <- get_sensitivity(df$class, df$scored.class)
recall
```

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

Step 7.

Question 7 - Specficity

$Specficity = \frac{TN}{TN+FP}$

Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the specificity of the predictions.

```
get_specficity <- function(actual, predicted){
  cm <- as.matrix.data.frame(table(predicted, actual))
  TN <- cm[1,1]
  FN <- cm[1,2]
  FP <- cm[2,1]
  TP <- cm[2,2]
  return (TN / (TN + FP))
}

get_specficity(df$class, df$scored.class)
```

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

Step 8.

Question 8 - F1 Score

$F1Score = \frac{2*precision*Sensitivity}{precision+Sensitivity}$

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.

```r
get_f1score <- function(prec, recall){
  return (2* prec * recall/ (prec + recall))
}

get_f1score(prec, recall)
```

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

Step 9

Question 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.

As we know, there are 2 types of Error. They are type-1 and type-2

Type-1 errors are False positive which will be dealt with the metric Precision.

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

Type-2 errors are False negative which will be dealt with the metric Recall or sensitivity.

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

Our goal is reduce the Type-1 and Type-2 errors.

When FP is 0, we have precision is equal to 1.

when FN is 0, we have recall is equal to 1

Therefore, the maximum value of precision is equal to 1.

When TP =0, we dont have any True positive cases and the precision and recall would be 0.

Therefore, the minimum value of recall is equal to 0.

$F1Minimum = \frac{2*0*0}{0+0} = 0$

$F1Maximum = \frac{2*1*1}{1+1} = 1$

hence the f1_score ranges from 0 to 1.

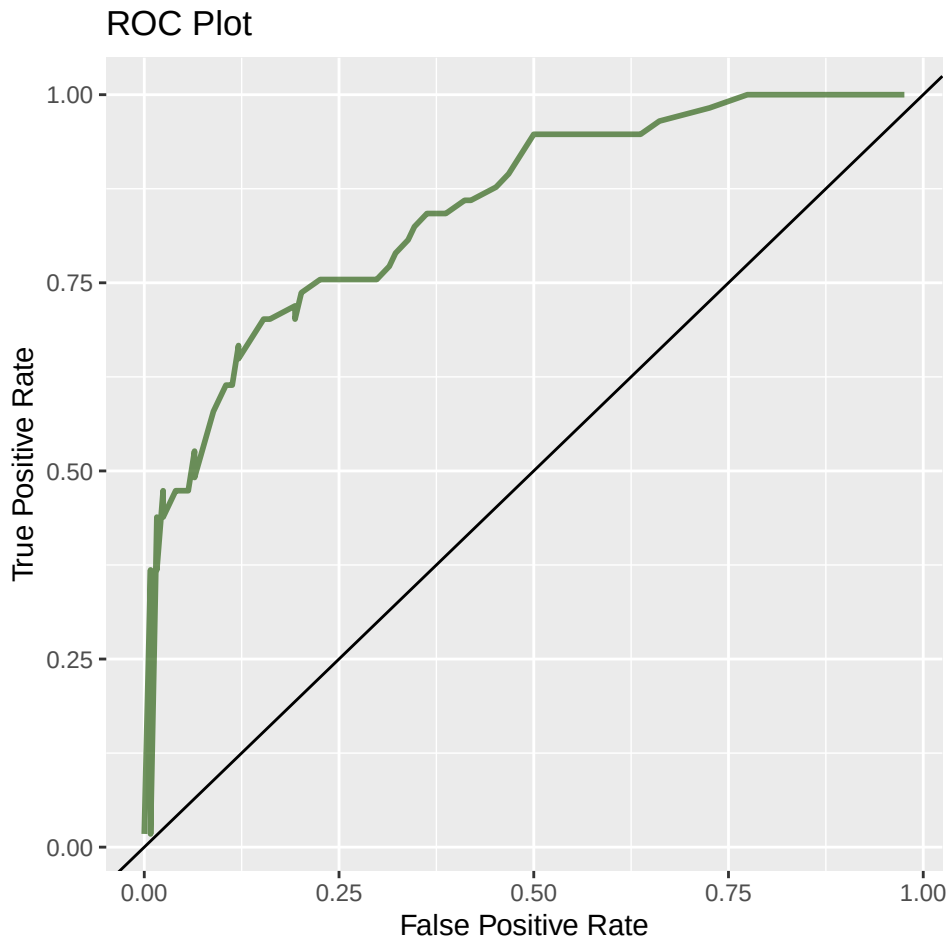
Step 10

Question 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

```r
library(ggplot2)
#ROC Curve
my_curve <- function(data) {
  thresholds <- seq(0, 1, by=0.01)
  X <- c()
  Y <- c()

  for (i in 1:length(thresholds)){
```

```
    scored_class <- ifelse(data$scored.probability >= thresholds[i], 1, 0)
    cur_df <- data.frame(scored.class = scored_class, class = data$class)
    df_table <- with(cur_df, table(scored.class, class))
    X[i] <- (df_table[4])/(df_table[4] + df_table[3])
    Y[i]<- (df_table[2]/(df_table[2] + df_table[1]))
  }

  plot_df <- data.frame(TRUE_POSITIVE = X, FALSE_POSITIVE = Y)
  # In order to roughly calculate the area under the curve, we must remove the NA values
  plot_df <- na.omit(plot_df)

  ROC_plot <- ggplot(plot_df, aes(x=FALSE_POSITIVE, y=TRUE_POSITIVE)) +
    geom_line(col=rgb(0.2,0.4,0.1,0.7) , lwd=1) +
    geom_abline(intercept = 0, slope = 1) +
    labs(title="ROC Plot", x = "False Positive Rate", y = "True Positive Rate")

  # Now to calculate the AUC
  x <- abs(diff(plot_df$FALSE_POSITIVE))
  y <- plot_df$TRUE_POSITIVE

  area_under_curve <- sum(x*y)

  ROC_plot2 <- ggplot(plot_df, aes(x=FALSE_POSITIVE, y=TRUE_POSITIVE)) +
    ggtitle('ROC Curve') +
    geom_line(col="blue") +
    geom_abline(intercept = 0, slope = 1) +
    geom_point() +
    xlab('False Positive Rate') +
    ylab('True Positive Rate')

  return(list(ROC_plot, area_under_curve))
}

result = my_curve(df)
ROC_plot <-result[[1]]
ROC_curve <-result[[2]]

result
```

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




```
##
## [[2]]
## [1] 0.8299377
```

Step 11.

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

```
my_Classification <- data.frame('Accuracy' = get_accuracy(df$class, df$scored.class),
                   'Error rate' = get_class_err_rate(df$class, df$scored.class),
                   'Precision' = get_precision(df$class, df$scored.class),
                   'Sensitivity' = get_sensitivity(df$class, df$scored.class),
                   'Specficity' = get_specficity(df$class, df$scored.class),
                   'F_Score' = get_f1score(prec, recall)
                   )


kable(my_Classification, booktabs = T)
```

Accuracy	Error.rate	Precision	Sensitivity	Specficity	F_Score
0.8066298	0.1933702	0.84375	0.4736842	0.9596774	0.6067416

Step 12.

Investigate the caret package. In particular, consider the functions confusionMatrix, sensitivity, and specificity. Apply the functions to the data set.

Note: the functions from the caret package ask for the Predicted values to be formatted as rows and the True Positives to be in the top left cell of the table. So we re-factor our variables before passing them into the functions.

```
truth <- factor(df$class)
levels(truth) = list("Yes" = 1, "No" = 0)
pred <- factor(df$scored.class)
levels(pred) = list("Yes" = 1, "No" = 0)

confusionMatrix(pred, truth)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  27   5
##        No   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 : 0.00004976
##
##             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 : Yes
##
```

```
sensitivity(pred, truth)
```

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

```
specificity(pred, truth)
```

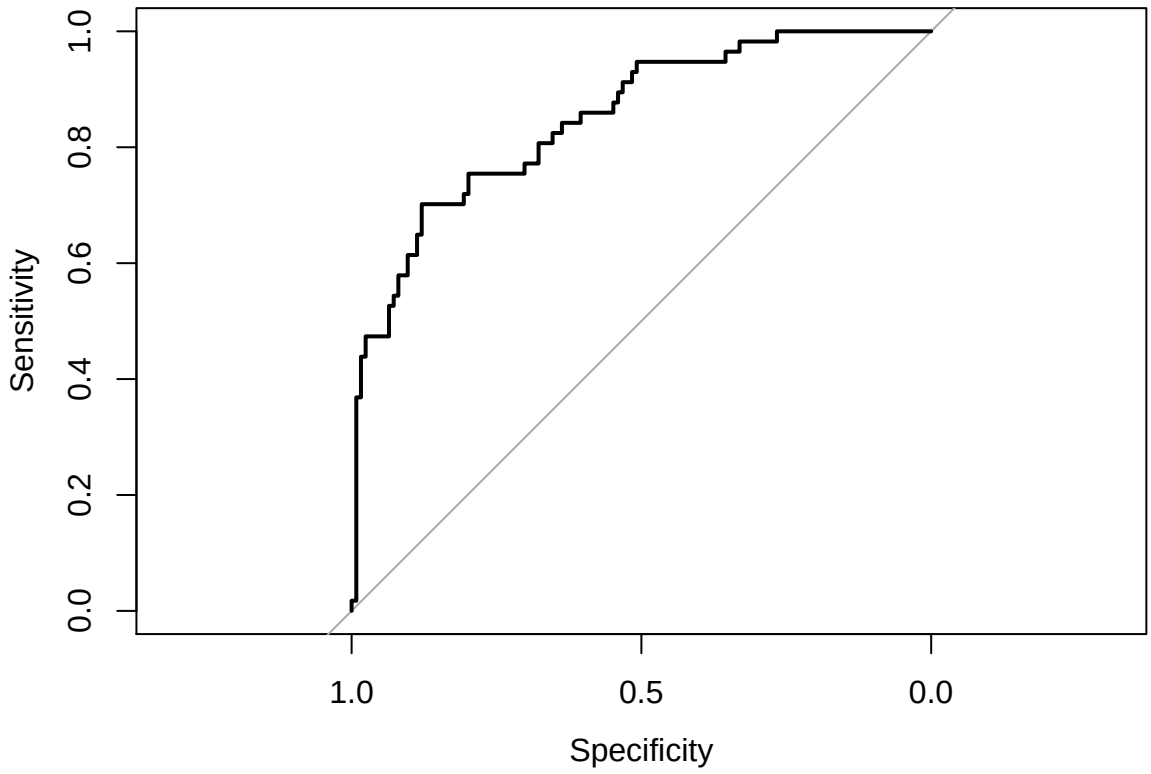
```
## [1] 0.9596774
```

We can see above that the caret functions give the same results as our manually created functions.

Step 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_obj <- roc(df$class, df$scored.probability, quiet = T)
plot(roc_obj)
```




```
print(roc_obj)
```

```
##
## Call:
## roc.default(response = df$class, predictor = df$scored.probability,     quiet = T)
##
## Data: df$scored.probability in 124 controls (df$class 0) < 57 cases (df$class 1).
## Area under the curve: 0.8503
```

We can see the pROC results are very similar to our manually created functions.