DATA 621 - Homework 2: Classification Prediction Model Analysis

Deepak Mongia & Soumya Ghosh

March 15, 2020

Libraries

```
library(kableExtra)
library(ggplot2)
library(dplyr)
library(MASS)
library(pROC)
library(caret)
```

Introduction

As a part of this homework assignment, we have been given a dataset called classification-output-data.csv, which has a set of independent variables or features and a observed class attribute, along with a predictive classification model scored class probability and scored class based on the scored probability. We have to use the below 3 key columns to derive some key classification model metrics -

- **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

Steps Followed:

1. Download the classification output data set

Loading the dataset into R:

```
hw_dataset <- read.csv("https://raw.githubusercontent.com/deepakmongia/Data621/master/HW-2/Data/classifi
```

Sample snapshot of data set:

```
head(hw_dataset, 20) %>% kable() %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed
```

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
5	88	78	30	0	27.6	0.258	37	0	0	0.1251892
5	108	72	43	75	36.1	0.263	33	0	0	0.2706248
13	76	60	0	0	32.8	0.180	41	0	0	0.2098096
0	100	70	26	50	30.8	0.597	21	0	0	0.0935859
7	194	68	28	0	35.9	0.745	41	1	1	0.8848457
12	92	62	7	258	27.6	0.926	44	1	0	0.3966522
0	173	78	32	265	46.5	1.159	58	0	1	0.8913949
3	171	72	33	135	33.3	0.199	24	1	1	0.5345490
8	196	76	29	280	37.5	0.605	57	1	1	0.9463342
5	99	74	27	0	29.0	0.203	32	0	0	0.1449162

Statistical Summary:

```
summary(hw_dataset)
```

```
##     pregnant         glucose        diastolic        skinfold
##  Min.   : 0.000   Min.   : 57.0   Min.   : 38.0   Min.   : 0.0
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.0   1st Qu.: 0.0
##  Median : 3.000   Median :112.0   Median : 70.0   Median :22.0
##  Mean   : 3.862   Mean   :118.3   Mean   : 71.7   Mean   :19.8
##  3rd Qu.: 6.000   3rd Qu.:136.0   3rd Qu.: 78.0   3rd Qu.:32.0
##  Max.   :15.000   Max.   :197.0   Max.   :104.0   Max.   :54.0
##     insulin            bmi          pedigree          age
##  Min.   :  0.00   Min.   :19.40   Min.   :0.0850   Min.   :21.00
##  1st Qu.:  0.00   1st Qu.:26.30   1st Qu.:0.2570   1st Qu.:24.00
##  Median :  0.00   Median :31.60   Median :0.3910   Median :30.00
##  Mean   : 63.77   Mean   :31.58   Mean   :0.4496   Mean   :33.31
##  3rd Qu.:105.00   3rd Qu.:36.00   3rd Qu.:0.5800   3rd Qu.:41.00
##  Max.   :543.00   Max.   :50.00   Max.   :2.2880   Max.   :67.00
##      class         scored.class    scored.probability
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.02323
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.11702
##  Median :0.0000   Median :0.0000   Median :0.23999
##  Mean   :0.3149   Mean   :0.1768   Mean   :0.30373
##  3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.43093
##  Max.   :1.0000   Max.   :1.0000   Max.   :0.94633
```

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?

```
# Subset original data set
classdata <- hw_dataset %>% dplyr::select(scored.class,class)

# Raw Confusion Matrix
cmat <- table(classdata)

cmat
```

```
##              class
## scored.class   0   1
##            0 119  30
##            1   5  27
```

We have formatted above rax confusion matrix to define Event & Non-Event. Here we have coded '1' values in the raw data set as **"Event"** (i.e. True value) and '0' values as **"Non-Event"**(i.e. False value).

```
cmat <- cmat[order(rownames(cmat),decreasing = T), order(colnames(cmat),decreasing = T)]
rownames(cmat) <- c("Event","Non-Event")
colnames(cmat) <- c("Event","Non-Event")

cmat %>%  kable %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"))
```

	Event	Non-Event
Event	27	5
Non-Event	30	119

The rows in the confusion matrix represent the predicted classes, **"Event"** and **"Non-Event"**, from the classification model. The columns represent the actual classes represented in the data set.

3. Calculating the accuracy:

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

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

```
Accuracy_func <- function(input_df){
  True_positive_plus_negative <- sum(input_df$class == input_df$scored.class)
  False_positive_plus_negative <- sum(input_df$class != input_df$scored.class)

  Accuracy <- True_positive_plus_negative / (True_positive_plus_negative + False_positive_plus_negative)
  return(Accuracy)
}
```

Accuracy Score for the given prediction Output:

```
Accuracy_func (hw_dataset)
```

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

4. Calculating the classification error rate:

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.

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

```
Class_error_rt_func <- function(input_df){
  True_positive_plus_negative <- sum(input_df$class == input_df$scored.class)
  False_positive_plus_negative <- sum(input_df$class != input_df$scored.class)

  error_rate <- False_positive_plus_negative / (True_positive_plus_negative + False_positive_plus_negati
  return(error_rate)
}
```

Classification Error rate for the given prediction Output:

```
Class_error_rt_func (hw_dataset)
```

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

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

```
Accuracy_func(hw_dataset) + Class_error_rt_func(hw_dataset)
```

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

5. Calculating Precision:

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

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

```
Precision_func <- function(input_df) {
  True_positive <- sum(input_df$class == 1 & input_df$scored.class == 1)
  False_positive <- sum(input_df$class == 0 & input_df$scored.class == 1)

  Precision_val <- True_positive / (True_positive + False_positive)

  return(Precision_val)
}
```

Precision Score for the given prediction Output:

```
Precision_func (hw_dataset)
```

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

6. Calculating Sensitivity:

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.

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

```
Sensitivity_func <- function(input_df) {
  True_positive <- sum(input_df$class == 1 & input_df$scored.class == 1)
  False_negative <- sum(input_df$class == 1 & input_df$scored.class == 0)

  Sensitivity_val <- True_positive / (True_positive + False_negative)

  return(Sensitivity_val)
}
```

Sensitivity Score for the given prediction Output:

```
Sensitivity_func (hw_dataset)
```

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

7. Calculating Specificity:

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

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

```
Specificity_func <- function(input_df) {
  True_negative <- sum(input_df$class == 0 & input_df$scored.class == 0)
  False_positive <- sum(input_df$class == 0 & input_df$scored.class == 1)

  Specificity_val <- True_negative / (True_negative + False_positive)

  return(Specificity_val)
}
```

Specificity Score for the given prediction Output:

```
Specificity_func (hw_dataset)
```

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

8. Calculating F1 score:

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.

$F1 \quad Score = \frac{2*Precision*Sensitivity}{Precision + Sensitivity}$

```
f1score_func <- function(input_df) {

  Precision_val <- Precision_func(input_df)
  Sensitivity_val <- Sensitivity_func(input_df)

  f1score_val <- ( 2 * Precision_val * Sensitivity_val ) / (Precision_val + Sensitivity_val)
  return(f1score_val)

}
```

F1 Score for the given prediction Output:

```
f1score_func (hw_dataset)
```

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

9. Bounds of F1 Score

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 $ab < a$.)

Expanding the definition of F1 Score as below -

$F1 \quad Score \quad = \quad \frac{2*Precision*Sensitivity}{Precision \quad + \quad Sensitivity}$

$= \quad \frac{2*\frac{TP}{TP+FP}*\frac{TP}{TP+FN}}{\frac{TP}{TP+FP}+\frac{TP}{TP+FN}}$

$= \quad \frac{\frac{2*TP^2}{(TP+FP)(TP+FN)}}{\frac{TP(2TP+FP+FN)}{(TP+FP)(TP+FN)}}$

$= \quad \frac{2TP}{2TP+FP+FN}$

In the equation above, TP, FP & FN are all positive integers. $TP, FP, FN \quad \in \quad N \quad where \quad N \quad = \{0, 1, 2, 3, ...\}$

So mathematically, 2TP $\leq$ 2TP+FP+FN. Hence Numerator is at most equal to denominator. Hence F1 Score ≤ 1.

The fraction, $\frac{2TP}{2TP+FP+FN}$ will have a minimum value of Zero when TP = 0 and maximum value of 1 when FP = FN = 0.

Hence it can be conculded that $0 \quad \leq \quad F1 \quad \leq \quad 1$.

10. Building the 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.

```
ROC_func <- function(input_df) {
  roc_df <- data.frame(matrix(ncol = 3, nrow = 0))
  names(roc_df) <- c("Probability","TPR","FPR")
  input_df_int <- data.frame(input_df)
  for (threshold in seq(0, 1, by = 0.01))
  {
    input_df_int$scored.class <- ifelse(input_df_int$scored.probability > threshold, 1, 0)
    tpr <- Sensitivity_func(input_df_int)
    fpr <- 1 - Specificity_func(input_df_int)
    row <- data.frame(probability = threshold, TPR = tpr, FPR = fpr)

    roc_df <- rbind(roc_df, row)

  }

  ## Compute the area
  roc_df$area <- -diff(roc_df$FPR)*roc_df$TPR
```

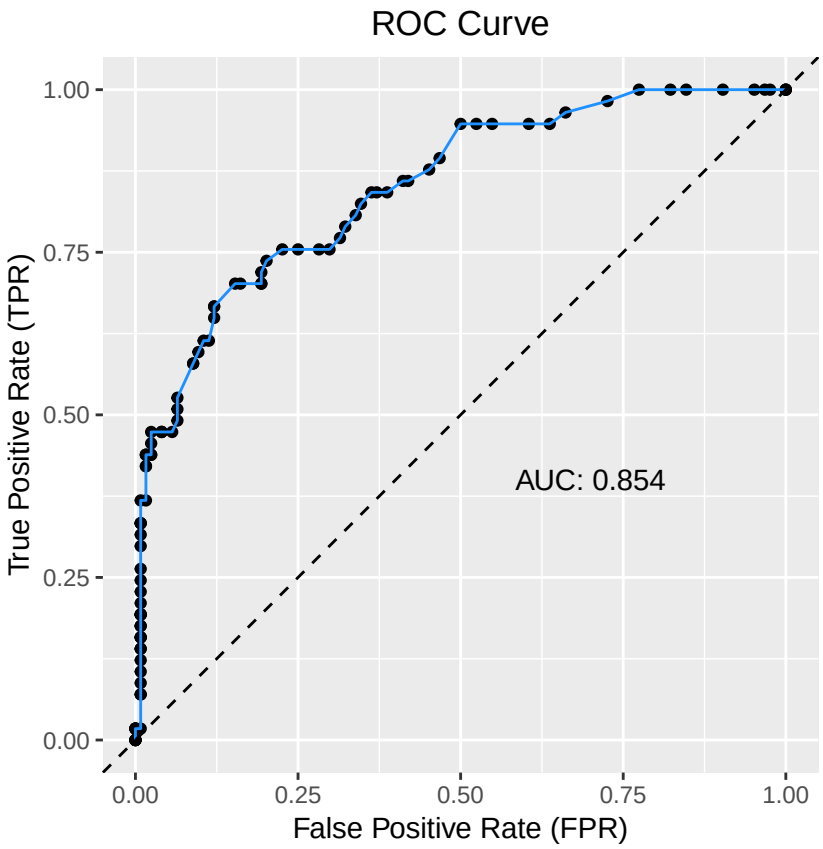
```
  ## Filled missing values with 0
  roc_df <- roc_df %>% mutate(area = ifelse(is.na(area),0,area))

  ## Calculate AUC as sum of the area of all rectangles
  AUC <- round(sum(roc_df$area),4)


  roc_curve <- ggplot(data = roc_df, aes(x = FPR, y = TPR)) +
                geom_point() +
                geom_path(color = "dodger blue") +
                ggtitle("ROC Curve") +
                xlab("False Positive Rate (FPR)") +
                ylab("True Positive Rate (TPR)") +
                theme(
                    plot.title = element_text(hjust = 0.5))+
                geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
                coord_equal(ratio = 1) +
                annotate(geom = "text", x = 0.7, y = 0.4, label = paste("AUC:", AUC))

  return(list(Plot = roc_curve, AUC = AUC))
}
```

Plotting the ROC curve:

```
ROC_func(hw_dataset)
```

```
## Warning in -diff(roc_df$FPR) * roc_df$TPR: longer object length is not a
## multiple of shorter object length
```

```
## $Plot
```




```
##
## $AUC
## [1] 0.854
```

11. Custom Function Model Outputs:

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

```
model_metrics <- c(Accuracy_func (hw_dataset), Class_error_rt_func (hw_dataset), Precision_func (hw_data
names(model_metrics) <- c("Accuracy", "Classification Error Rate", "Precision", "Sensitivity", "Specifi

model_metrics %>% kable(col.names = "Metrics") %>% kable_styling(bootstrap_options = c("striped", "hove
```

	Metrics
Accuracy	0.8066298
Classification Error Rate	0.1933702
Precision	0.8437500
Sensitivity	0.4736842
Specificity	0.9596774
F1 Score	0.6067416

12. Caret Package Function Outputs:

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?

```
hw_dataset_1 <- hw_dataset %>% dplyr::select(scored.class, class) %>%
  mutate(scored.class = as.factor(scored.class),
         class = as.factor(class))

caret_op <- caret::confusionMatrix(data = hw_dataset_1$scored.class,
                reference = hw_dataset_1$class,
                positive = "1")

caret_package <- c(caret_op$overall["Accuracy"], caret_op$byClass["Sensitivity"], caret_op$byClass["Spe
our_function <- c(Accuracy_func (hw_dataset), Sensitivity_func (hw_dataset), Specificity_func (hw_datas

model_comp <- cbind(caret_package, our_function)

model_comp %>% kable() %>% kable_styling(bootstrap_options = c("striped", "hover", "condensed", "respons
```

	caret_package	our_function
Accuracy	0.8066298	0.8066298
Sensitivity	0.4736842	0.4736842
Specificity	0.9596774	0.9596774

As per the table above the Accuracy, Sensitivity and Specitivity scores generated by the Caret package are exactly identical to the corresponding model metrics calculated by our custom functions.

13. ROC Curve using pROC Package:

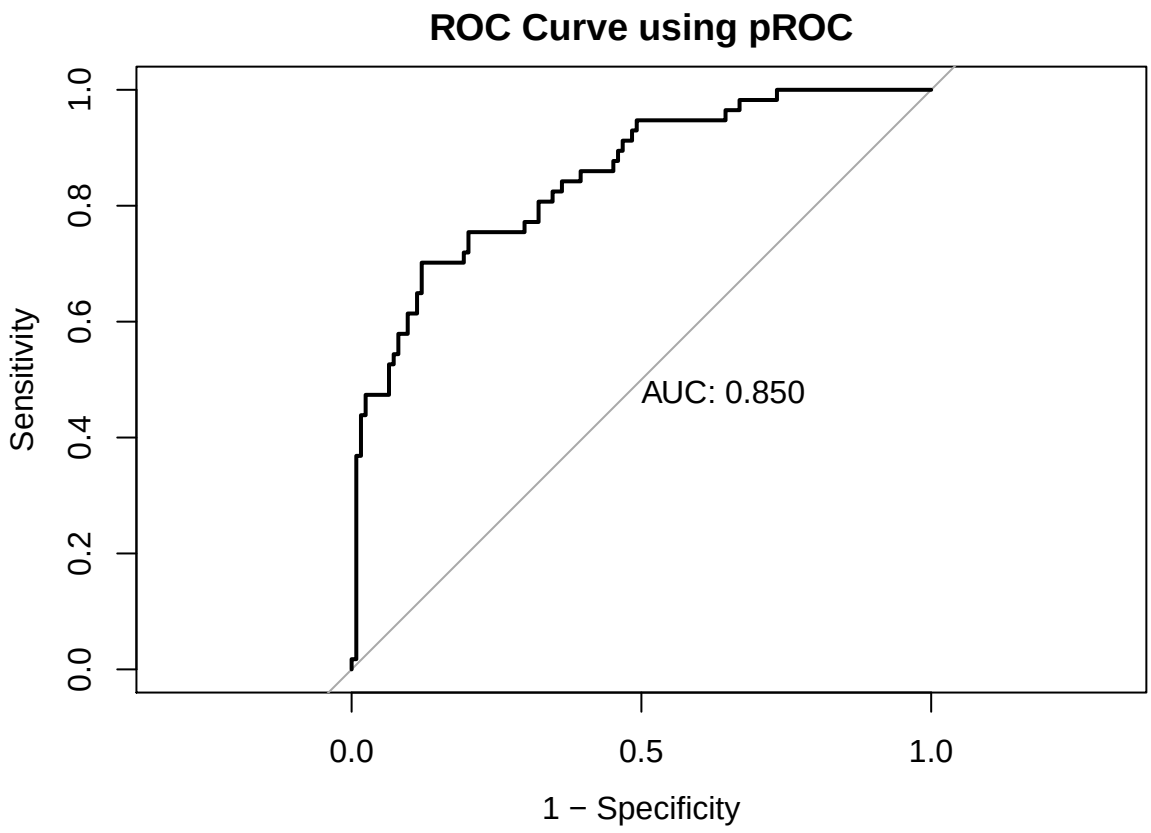
Investigate the **pROC** package. Use it to generate an ROC curve for the data set. How do the results compare with your own functions?

```
rocCurve <- roc(response = hw_dataset$class,
                predictor = hw_dataset$scored.probability)
```

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

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

```
## Plot the ROC Curve:
plot(rocCurve, legacy.axes = TRUE, main = "ROC Curve using pROC",print.auc = TRUE)
```

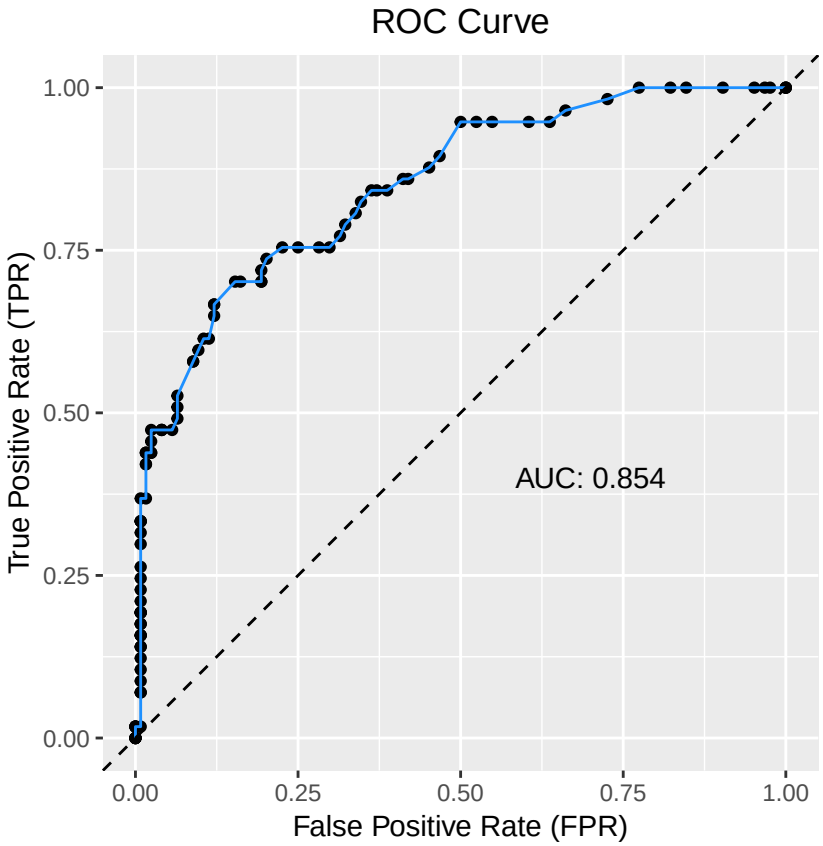



```
## Area Under the Curve (AUC):
auc(rocCurve)
```

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

```
## Plot ROC with Custom Function
ROC_func(hw_dataset)$Plot
```

```
## Warning in -diff(roc_df$FPR) * roc_df$TPR: longer object length is not a
## multiple of shorter object length
```




```
## Area Under the Curve (AUC) using Custom Function:
ROC_func(hw_dataset)$AUC
```

```
## Warning in -diff(roc_df$FPR) * roc_df$TPR: longer object length is not a
## multiple of shorter object length
```

```
## [1] 0.854
```

Conclusion: The shape of the ROC cusrve generated by pROC package and our function are almost identical. The AUC values are comparable but slightly different.