Homework-2

Waheeb Algabri, Joe Garcia, Lwin Shwe, Mikhail Broomes

1. Import the classification output data set

```r
class_df <- as.data.frame(read.csv('https://raw.githubusercontent.com/waheeb123/Data-621/main/Homeworks/
kable(head(class_df))
```

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

The dataset provided includes several attributes to predict whether or not a patient has diabetes.

The data set has three key columns we will use:

```
a. class: the actual class for the observation

b. scored.class: the predicted class for the observation (based on a threshold of $0.5$)

c. 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?

2. Getting the raw confusion matrix

The fct_recode function can be used from the forcats package for factor recoding.

```
##                    scored.class
## class               Predicted Negative Predicted Positive
##   Actual Negative                  119                  5
##   Actual Positive                   30                 27
```

The output of confusion matrix such as the counts of *TN* True Negative are 119, *TP* True Positive are 27, the *FP* False Positive are 5; and *FN* False Negative value are 30.

3. Finding Accuracy of the Predictions

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

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

```r
Accuracy <- function(class_df){
  TN <- sum(class_df$class == 0 & class_df$scored.class ==0)
  TP <- sum(class_df$class == 1 & class_df$scored.class ==1)
  FP <- sum(class_df$class == 0 & class_df$scored.class ==1)
  FN <- sum(class_df$class == 1 & class_df$scored.class ==0)
  return((TN+TP)/(TN + TP + FP + FN))
}
Accuracy(class_df)
```

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

4. Calculate the classification error rate of the predictions

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.

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

```r
Classification_Error_Rate <- function(class_df){
  TN <- sum(class_df$class == 0 & class_df$scored.class ==0)
  TP <- sum(class_df$class == 1 & class_df$scored.class ==1)
  FP <- sum(class_df$class == 0 & class_df$scored.class ==1)
  FN <- sum(class_df$class == 1 & class_df$scored.class ==0)
  return((FP + FN)/(TN + TP + FP + FN))
}
Classification_Error_Rate(class_df)
```

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

5. Compute the precision of the predictions

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

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

```r
Precision <- function(class_df){
  TP <- sum(class_df$class == 1 & class_df$scored.class ==1)
  FP <- sum(class_df$class == 0 & class_df$scored.class ==1)
  return((TP)/(TP + FP))
}
Precision(class_df)
```

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

6. Find the sensitivity of the predictions

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.

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

```r
Sensitivity <- function(class_df){
  TP <- sum(class_df$class == 1 & class_df$scored.class ==1)
  FN <- sum(class_df$class == 1 & class_df$scored.class ==0)
  return((TP)/(TP + FN))
}
Sensitivity(class_df)
```

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

7. Find the specificity of the predictions

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

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

```r
Specificity <- function(class_df){
  TN <- sum(class_df$class == 0 & class_df$scored.class ==0)
  FP <- sum(class_df$class == 0 & class_df$scored.class ==1)
  return((TN)/(TN + FP))
}
Specificity(class_df)
```

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

8. Find the F1 score of the predictions

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.

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

```r
F1_score <- function(class_df){
  Precision <- Precision(class_df)
  Sensitivity <- Sensitivity(class_df)
  return((2 * Precision * Sensitivity)/(Precision + Sensitivity))
}

F1_score(class_df)
```

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

9. What are the bounds on the 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 $ab < a$.)

Precision is the proportion of true positive predictions among all positive predictions. The (TP) is a count of predicted positive samples, and the sum of true positives and false positives is positive, so precision is bounded by $[0, 1]$.

$$0 < \text{Precision} = \frac{TP}{TP + FP} < 1$$

Sensitivity is the ratio of true positive predictions among all actual positive instances. Similarly, the sum of true positives and false negatives is also positive value, so sensitivity is bounded by $[0, 1]$.

$$0 < \text{Sensitivity} = \frac{TP}{TP + FN} < 1$$

The F1 score is defined as:

$$0 < F1\ \text{Score} = \frac{2 \times \text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}} < 1$$

Result of the F1 function given that the maximum value for precision and sensitivity is 1 if every single value were correctly predicted:

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

Alternatively, the worst case scenario for the metrics from a classification model would if every single prediction was incorrect:

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

Let's show graphically that even if one of the scores was perfect/imperfect the maximum value as assumed before would be 1.

```r
x <- seq(0, 1, by=0.01)

# Calculate the function values
y <- (2* x * 1)/(x+1)

# Create a data frame with x and y values
df <- data.frame(x = x, y = y)

ggplot(data = df, aes(x = x, y = y)) +
  geom_point() +
  labs(x = "x", y = "f(x)",title = "F1 Score between 0 and 1") +
  theme_light()
```

This verify that the F1 score will always fall within $[0, 1]$.

10. Plot 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.

```r
ROC <- function(x, y) {
  # Sort the data by decreasing order of the probability column
  x <- x[order(y, decreasing = TRUE), ]

  # Calculate True Positive Rate (Sensitivity) and False Positive Rate (1-Specificity)
  TPR <- cumsum(x$class) / sum(x$class)
  FPR <- cumsum(!x$class) / sum(!x$class)

  # Create a data frame with TPR, FPR, and the probability column
  xy <- data.frame(TPR, FPR, x$scored.probability)

  # Calculate the area under the ROC curve using the trapezoidal method
  FPR_df <- c(diff(xy$FPR), 0)
  TPR_df <- c(diff(xy$TPR), 0)
  AUC <- round(sum(xy$TPR * FPR_df) + sum(TPR_df * FPR_df) / 2, 4)

  # Plot the ROC curve
  plot(xy$FPR, xy$TPR, type = "l",
       main = "ROC Curve",
       xlab = "False Positive Rate (Specificity)",
       ylab = "True Positive Rate (Sensitivity)")

  # Add a red dashed line representing a random model
  abline(a = 0, b = 1, col = "red", lty = 2)

  # Add a legend with the calculated AUC
  legend(0.6, 0.4, legend = paste("AUC =", AUC), title = "AUC", col = "black", lty = 1)
}

# Call the ROC function with the true class column and predicted probabilities
ROC(class_df, class_df$scored.probability)
```

11. Use your created R functions and the provided classification output data set

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

```r
# Calculate the classification metrics
conf_matrix <- table(class_df$class, class_df$scored.class)
accuracy <- Accuracy(class_df)
error_rate <- Classification_Error_Rate(class_df)
precision <- Precision(class_df)
sensitivity <- Sensitivity(class_df)
specificity <- Specificity(class_df)
```

```
f1_score <- F1_score(class_df)

# Create a data frame with the metrics
metrics_df <- data.frame(
  Metric = c("Accuracy", "Error Rate", "Precision", "Sensitivity", "Specificity", "F1 Score"),
  Value = c(accuracy, error_rate, precision, sensitivity, specificity, f1_score)
)

# Print the confusion matrix
print("Confusion Matrix:")
```

```
## [1] "Confusion Matrix:"
```

```
print(conf_matrix)
```

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

```
# Print the metrics data frame
print("Classification Metrics:")
```

```
## [1] "Classification Metrics:"
```

```
print(metrics_df)
```

```
##        Metric     Value
## 1    Accuracy 0.8066298
## 2  Error Rate 0.1933702
## 3   Precision 0.8437500
## 4 Sensitivity 0.4736842
## 5 Specificity 0.9596774
## 6    F1 Score 0.6067416
```

```
# Plot ROC curve
ROC(class_df, class_df$scored.probability)
```

12. Explore the functions of confusionMatrix using caret Package

Investigate the caret package. In particular, consider the functions confusion Matrix, sensitivity, and specificity. Apply the functions to the data set. How do the results compare with your own functions?

```
library(caret)

# Creates vectors having data points
expected_value <- factor(c(class_df %>% pull(class)))
```

```r
predicted_value <- factor(c(class_df %>% pull(scored.class)))

#Perform confusion matrix
caret_confusion_matrix <- confusionMatrix(data=predicted_value, reference = expected_value, positive='1

#Display results
caret_confusion_matrix
```

```
## 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
##
```

All the calculated values and the results from using caret package and the functions, confusionMatrix are the same.

13. Create 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?

```r
library(pROC)
pROC_obj <- roc(class_df$class,class_df$scored.probability,
            smoothed = TRUE,
            # arguments for ci
            ci=TRUE, ci.alpha=0.01, stratified=FALSE,
            # arguments for plot
            plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE, grid=TRUE,
            print.auc=TRUE, show.thres=TRUE)
```

```
sens.ci <- ci.se(pROC_obj)
plot(sens.ci, type="shape", col="yellow")
```

The overall results shows that the manual ROC curve is consistent with the curve generated by pROC package for the dataset. The ‘roc()’ function generates a ROC curve with more thresholds, resulting in a smoother curve. The thresholds are unevenly spaced, and the plot uses specificity on the x-axis instead of 1-specificity. Despite these differences, the AUC values from manual plot and the pROC function are comparably similar, with a small discrepancy likely due to slight variations in threshold calculations.