Data621: HW2

Shoshanna Farber, Josh Forster, Glen Davis, Andrew Bowen, Charles Ugiagbe

2023-10-05

```
library(tidyverse)
library(geomtextpath)
```

Overview:

In this homework assignment, you will work through various classification metrics. You will be asked to create functions in R to carry out the various calculations. You will also investigate some functions in packages that will let you obtain the equivalent results. Finally, you will create graphical output that also can be used to evaluate the output of classification models, such as binary logistic regression.

Supplemental Material:

- Applied Predictive Modeling, Ch. 11 (provided as a PDF file).
- Web tutorials: http://www.saedsayad.com/model_evaluation_c.htm

Deliverables (100 Points):

- Upon following the instructions below, use your created R functions and the other packages to generate the classification metrics for the provided data set. A write-up of your solutions submitted in PDF format.

Instructions:

Complete each of the following steps as instructed:

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

```
class_df <- as.data.frame(read.csv('https://raw.githubusercontent.com/andrewbowen19/businessAnalyticsDa
head(class_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
```

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

2. 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?

```
class_df <- class_df |>
  mutate(class = factor(class, levels=c(1,0)),
         scored.class = factor(scored.class, levels=c(1,0)))
keep <- c("scored.class", "class")
class_subset <- class_df |>
    select(all_of(keep))
confusion_matrix <- table(class_subset)
confusion_matrix
```

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

The row labels in the confusion matrix represent the predicted class, while the column labels represent the actual class. The values in the confusion matrix correspond to the number of observations correctly/incorrectly predicted for this dataframe. There were 27 instances where a patient was accurately diagnosed with diabetes (true positive) and 119 instances where a patient without diabetes was classified as not having diabetes (true negative). There were 30 instances where a patient with diabetes was incorrectly classified as not having diabetes (false negative) and 5 instances of a patient being incorrectly diagnosed with diabetes (false positive).

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.

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

```
accuracy.func <- function(input_df) {
    accuracy <- input_df |> dplyr::mutate(correct=ifelse(class==scored.class,1,0)) |> summarise(total_c
}

acc_val <- accuracy.func(class_subset)
print(round(acc_val, 2))
```

```
##   accuracy
## 1     0.81
```

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.

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

```
error.func <- function(input_df) {
    accuracy <- input_df |> dplyr::mutate(incorrect=ifelse(class==scored.class,0,1)) |> summarise(total_
}

error_val <- error.func(class_subset)
print(round(error_val, 2))
```

```
##   error_rate
## 1       0.19
```

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

```
(as.numeric(acc_val) + as.numeric(error_val))
```

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

Confirmed. The accuracy and error rate values do add up 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.

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

```
precision.func <- function(input_df) {
    precision <- input_df |> dplyr::filter(scored.class==1) |> mutate(prec=ifelse(class==scored.class,1
        summarise(total_prec = sum(prec),precision_rate=total_prec/n()) |> select(c(precision_rate))
}

prec_val <- precision.func(class_subset)
print(round(prec_val, 2))
```

```
##   precision_rate
## 1           0.84
```

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.

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

```
sensitivity.func <- function(input_df) {
    sensitivity <- input_df |> dplyr::filter(class==1) |> mutate(sens=ifelse(class==scored.class,1,0))
        summarise(total_sens = sum(sens),sensitivity_rate=total_sens/n()) |> select(c(sensitivity_rate))
}

sens_val <- sensitivity.func(class_subset)
print(round(sens_val, 2))
```

```
##   sensitivity_rate
## 1             0.47
```

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.

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

```
specificity.func <- function(input_df) {
    specificity <- input_df |> dplyr::filter(class==0) |> mutate(spec=ifelse(class==scored.class,1,0))
        summarise(total_spec = sum(spec),specificity_rate=total_spec/n()) |> select(c(specificity_rate))
}

spec_val <- specificity.func(class_subset)
print(round(spec_val, 2))
```

```
##   specificity_rate
## 1             0.96
```

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.

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

```
f1.score <- function(input_df) {
    f1<-(2*precision.func(input_df)*sensitivity.func(input_df))/(precision.func(input_df)+sensitivity.fu
}

f1_val <- f1.score(class_subset)
colnames(f1_val) <- "f1_score"
print(round(f1_val, 2))
```

```
##   f1_score
## 1     0.61
```

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 $ab < a$.)

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:

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

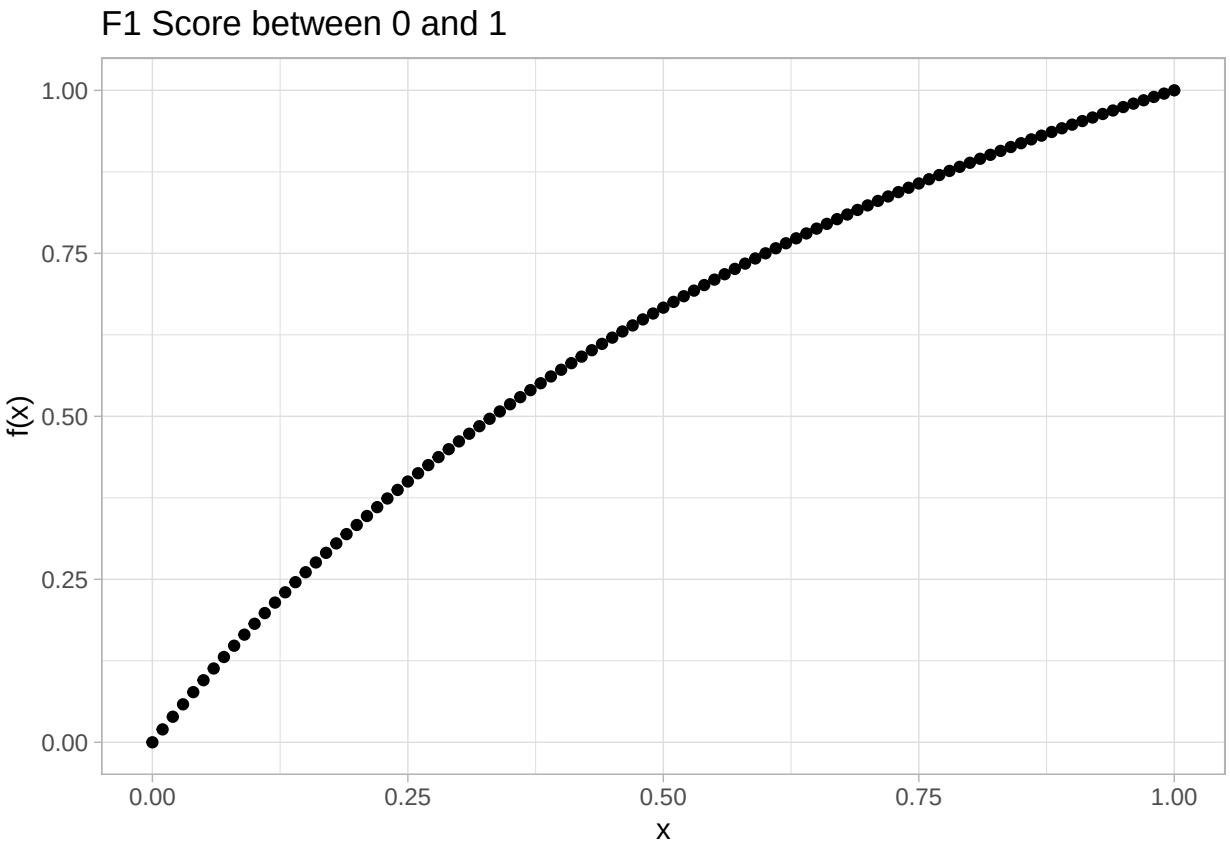
Let's show graphically that even if one of the scores was perfect/imperfect the maximum value as assumed before would be 1.

```
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()
```

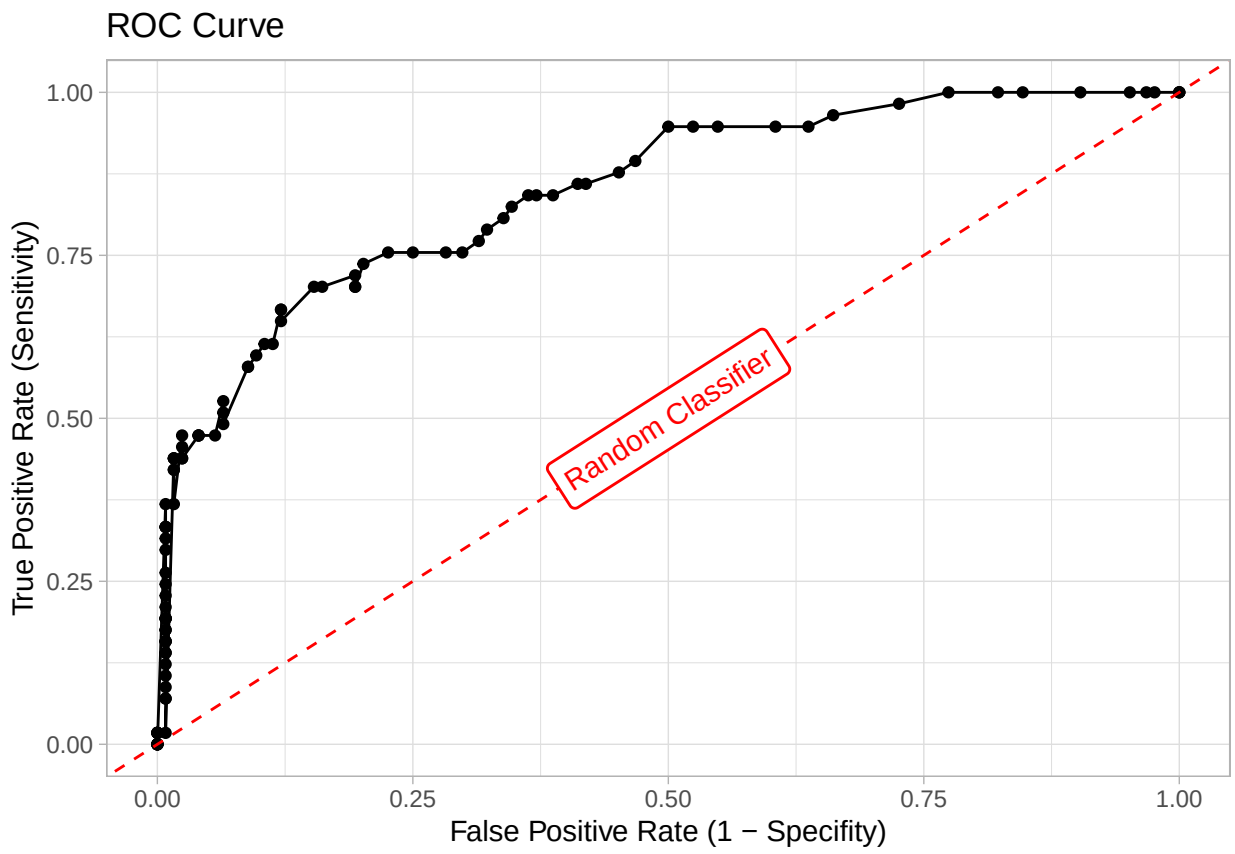


10. Write a function that generates an ROC curve from a data set with a true classification column (class in our example) and a probability column (scored.probability in our example). Your function should return a list that includes the plot of the ROC curve and a vector that contains the calculated area under the curve (AUC). Note that I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.

```
keep <- c("class", "scored.probability")
class_subset <- class_df |>
    select(all_of(keep))
roc.func <- function(input_df){
    plot_points_df <- as.data.frame(matrix(ncol = 3, nrow = 0))
    cols <- c("threshold", "fpr", "tpr")
    colnames(plot_points_df) <- cols
    thresholds <- seq(from = 0, to = 1, by = 0.01)
    copy <- input_df
    for (i in 1:length(thresholds)){
        threshold <- thresholds[i]
        copy <- copy |>
            mutate(scored.class = ifelse(scored.probability > threshold, 1, 0))
        fpr <- 1 - as.numeric(specificity.func(copy))
        tpr <- as.numeric(sensitivity.func(copy))
        new_row <- as.data.frame(t(c(threshold, fpr, tpr)))
        colnames(new_row) <- cols
        plot_points_df <- rbind(plot_points_df, new_row)
    }
    roc_plot <- ggplot(plot_points_df, aes(x = fpr, y = tpr)) +
        geom_point() +
        geom_line() +
        geom_labelabline(intercept = 0, slope = 1, label = "Random Classifier",
                         linetype = "dashed", color = "red") +
        labs(x = "False Positive Rate (1 - Specifity)",
             y = "True Positive Rate (Sensitivity)",
             title = "ROC Curve") +
        theme_light()
    auc <- 0
    plot_points_df <- plot_points_df |>
        arrange(fpr)
    for (k in 2:nrow(plot_points_df)){
        x2 <- plot_points_df[k, 2]
        x1 <- plot_points_df[k - 1, 2]
        y2 <- plot_points_df[k, 3]
        y1 <- plot_points_df[k - 1, 3]
        trap_area <- (x2 - x1) * ((y2 + y1)/2)
        auc <- auc + trap_area
    }
    roc_plot_and_auc <- list(ROC_Plot = roc_plot, AUC = round(auc, 2))
    return(roc_plot_and_auc)
}

roc_plot_and_auc <- roc.func(class_subset)
roc_plot_and_auc
```

```
## $ROC_Plot
```




```
##
## $AUC
## [1] 0.85
```

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

```
metrics_table <- cbind(acc_val, error_val, prec_val, sens_val, spec_val, f1_val) |>
  pivot_longer(accuracy:f1_score,
               names_to="metric",
               values_to="value")

knitr::kable(metrics_table)
```

metric	value
accuracy	0.8066298
error_rate	0.1933702
precision_rate	0.8437500
sensitivity_rate	0.4736842
specificity_rate	0.9596774
f1_score	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?

```r
# load package
library(caret)

# confusionMatrix
(conf_matrix <- confusionMatrix(class_df$scored.class,
                reference = class_df$class,
                positive = '1'))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   0
##          1  27   5
##          0  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 : 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
##
```

```r
# sensitivity
caret_sens <- sensitivity(conf_matrix$table)

# specificity
caret_spec <- specificity(conf_matrix$table)

# compare functions
caret_calcs <- as.data.frame(cbind(caret_sens, caret_spec))
colnames(caret_calcs) <- c("sensitivity_rate", "specificity_rate")
caret_calcs |>
  pivot_longer(cols=c("sensitivity_rate", "specificity_rate"),
               names_to = "metric",
               values_to = "caret value") |>
  left_join(metrics_table, by="metric") |>
  knitr::kable(col.names = c("Metric", "Caret Values", "Function Values"))
```

Metric	Caret Values	Function Values
sensitivity_rate	0.4736842	0.4736842
specificity_rate	0.9596774	0.9596774

The values from our functions and the values from the sensitivity and specificity functions are the same.

Let's check the rest of our functions against the values from the confusionMatrix function. Sensitivity and specificity can also be taken from here.

```
caret_acc <- conf_matrix$overall[1]
caret_err <- 1-caret_acc
caret_prec <- conf_matrix$byClass[5]
caret_sens <- conf_matrix$byClass[1]
caret_spec <- conf_matrix$byClass[2]
caret_f1 <- conf_matrix$byClass[7]

caret_results <- as.data.frame(cbind(caret_acc, caret_err, caret_prec, caret_sens, caret_spec, caret_f1
colnames(caret_results) <- metrics_table$metric
caret_results |>
  pivot_longer(accuracy:f1_score,
               names_to="metric",
               values_to="caret_vals") |>
  right_join(metrics_table) |>
  knitr::kable(col.names = c("Metric", "Function Values", "Caret Values"))
```

```
## Joining with `by = join_by(metric)`
```

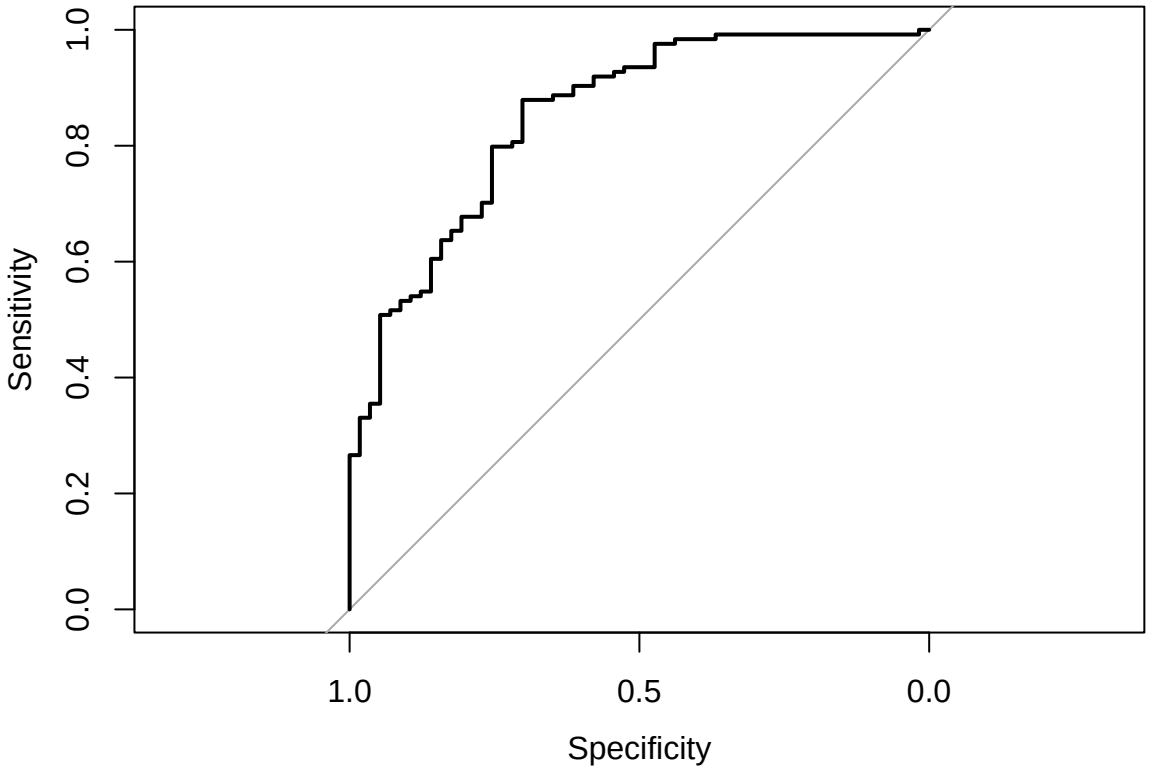
Metric	Function Values	Caret Values
accuracy	0.8066298	0.8066298
error_rate	0.1933702	0.1933702
precision_rate	0.8437500	0.8437500
sensitivity_rate	0.4736842	0.4736842
specificity_rate	0.9596774	0.9596774
f1_score	0.6067416	0.6067416

The values for each of our functions are the same as the values for the built in functions from 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?

```
library(pROC)

roc_from_pROC <- roc(class_df$class,
    class_df$scored.probability,
    plot=T)
```



Examining the list of information produced by the roc() function from the pROC package, including the plot, we can see this function uses more thresholds than our function, and these thresholds aren't evenly spaced. As the thresholds get smaller, the differences between one threshold and the next get smaller as well. The curve from this function appears to step as a result, and it has no apparent jagged dips like ours. The AUC returned is the same though, and the main difference is in the plot itself. $Specificity$ is plotted on the x-axis instead of $1 - Specificity$, so the curve runs from 1 to 0 on the x-axis instead of from 0 to 1.