Homework 2

Brandon Chung, Jiaxin Zheng, Andreina Arias, and Stephanie Chiang

Fall 2025

1. Download the classification output data set.

```
co_df <- read.csv("https://raw.githubusercontent.com/Chung-Brandon/621-Data-Mining/refs/heads/main/class
```

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?

ANSWER:

```
table(co_df$scored.class, co_df$class)
```

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

In the above confusion matrix outputted using `table()`, the columns represent the `class` or actual values and the rows are the predicted values from the `scored.class` column. The cells contain the counts for each combination:

- 119 true negatives → predicted 0, actually 0
- 30 false negatives → predicted 0, actually 1
- 5 false positives → predicted 1, actually 0
- 27 true positives → predicted 1, actually 1

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.

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

ANSWER:

The below is a function that takes as its parameters a dataframe and the names of two columns: actual values and predicted values. It then builds a confusion matrix, grabs the counts and calculates with the given accuracy formula.

```r
prediction_accuracy <- function(df, actual_col, predicted_col) {
  actual <- df[[actual_col]]
  predicted <- df[[predicted_col]]

  cm <- table(predicted, actual)

  TN <- cm["0", "0"]
  TP <- cm["1", "1"]
  FP <- cm["1", "0"]
  FN <- cm["0", "1"]

  accuracy <- (TP + TN) / (TP + FP + TN + FN)
  return(accuracy)
}

co_accuracy <- prediction_accuracy(co_df, "class", "scored.class")
co_accuracy
```

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

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.

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

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

ANSWER:

Using the same parameters as the previous function, the below calculates the error rate, followed by confirmation that our accuracy and errors equal 1.

```r
error_rate <- function(df, actual_col, predicted_col) {
  actual <- df[[actual_col]]
  predicted <- df[[predicted_col]]

  cm <- table(predicted, actual)

  TN <- cm["0", "0"]
  TP <- cm["1", "1"]
  FP <- cm["1", "0"]
  FN <- cm["0", "1"]

  errors <- (FP + FN) / (TP + FP + TN + FN)
  return(errors)
}

co_errors <- error_rate(co_df, "class", "scored.class")
co_errors
```

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

```
co_accuracy + co_errors == 1
```

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

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.

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

ANSWER:

```
pred_precision <- function(df, actual_col, predicted_col) {
  actual <- df[[actual_col]]
  predicted <- df[[predicted_col]]

  cm <- table(predicted, actual)
  TP <- cm["1", "1"]
  FP <- cm["1", "0"]

  precision <- (TP) / (TP + FP)
  return(precision)
}

co_prec <- pred_precision(co_df, "class", "scored.class")
co_prec
```

```
## [1] 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.

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

ANSWER:

```
sensitivity <- function(df, actual_col, predicted_col) {
  actual <- df[[actual_col]]
  predicted <- df[[predicted_col]]

  cm <- table(predicted, actual)
  TP <- cm["1", "1"]
  FN <- cm["0", "1"]

  result <- (TP) / (TP + FN)
  return(result)
}

co_recall <- sensitivity(co_df, "class", "scored.class")
co_recall
```

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

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.

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

ANSWER:

```
specificity <- function(df, actual_col, predicted_col) {
  actual <- df[[actual_col]]
  predicted <- df[[predicted_col]]

  cm <- table(predicted, actual)
  TN <- cm["0", "0"]
  FP <- cm["1", "0"]

  specificity <- (TN) / (TN + FP)
  return(specificity)
}

co_spec <- specificity(co_df, "class", "scored.class")
co_spec
```

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

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.

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

ANSWER:

```
#F1 Function
F1_Score=function(df, actual_col, predicted_col, positive){
  actual<-df[[actual_col]]
  predicted<-df[[predicted_col]]

  #True positive, false negative, and false positive for precision and recall
  TP<- sum(predicted==positive & actual==positive)
  FN <- sum(predicted!=positive & actual==positive)
  FP <- sum(predicted==positive & actual!=positive)

  #Precision and recall caulation, handle possible NaN  due to division by 0
  co_prec<-if ((TP+FP)==0)0 else TP/ (TP + FP)
  co_recall<-if ((TP+FN)==0)0 else TP/(TP + FN)


  #F1 score
```

```
  if(co_prec+co_recall == 0){
    return(0)
  }else {
    return((2* co_prec *co_recall)/(co_prec+co_recall))
  }
}


#Calculate F1
F1_co_df<-F1_Score(co_df, "class", "scored.class", positive = 1)
print(F1_co_df)
```

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

```
#check work
F1Manually<-(2*.8437*.4737)/(.8437+.4737)
print(F1Manually)
```

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

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 $0 < \quad < 1$ and $0 < \quad < 1$ then $\quad < \quad$.)

ANSWER:

F1=0 when either the precision or sensitivity is 0, F1=1 when both prescision and sensitivity are 1. F1 would be a score in between 0 and 1 when the is a balance between precision and recall. Since precision and sensitive are non negative F1 score will be non negative when prescisiona nd sensitive are multiplied or added together. The hint $0 < \quad < 1$ and $0 < \quad < 1$ then $\quad < \quad$, shows that the harmonic mean is less than or equal to the arthmetic mean, and less than or equal to the minimum of it components; the product of two values less than 1 is smaller than either value individually, keeping F1 less than or equal to both precision and sensitivity.

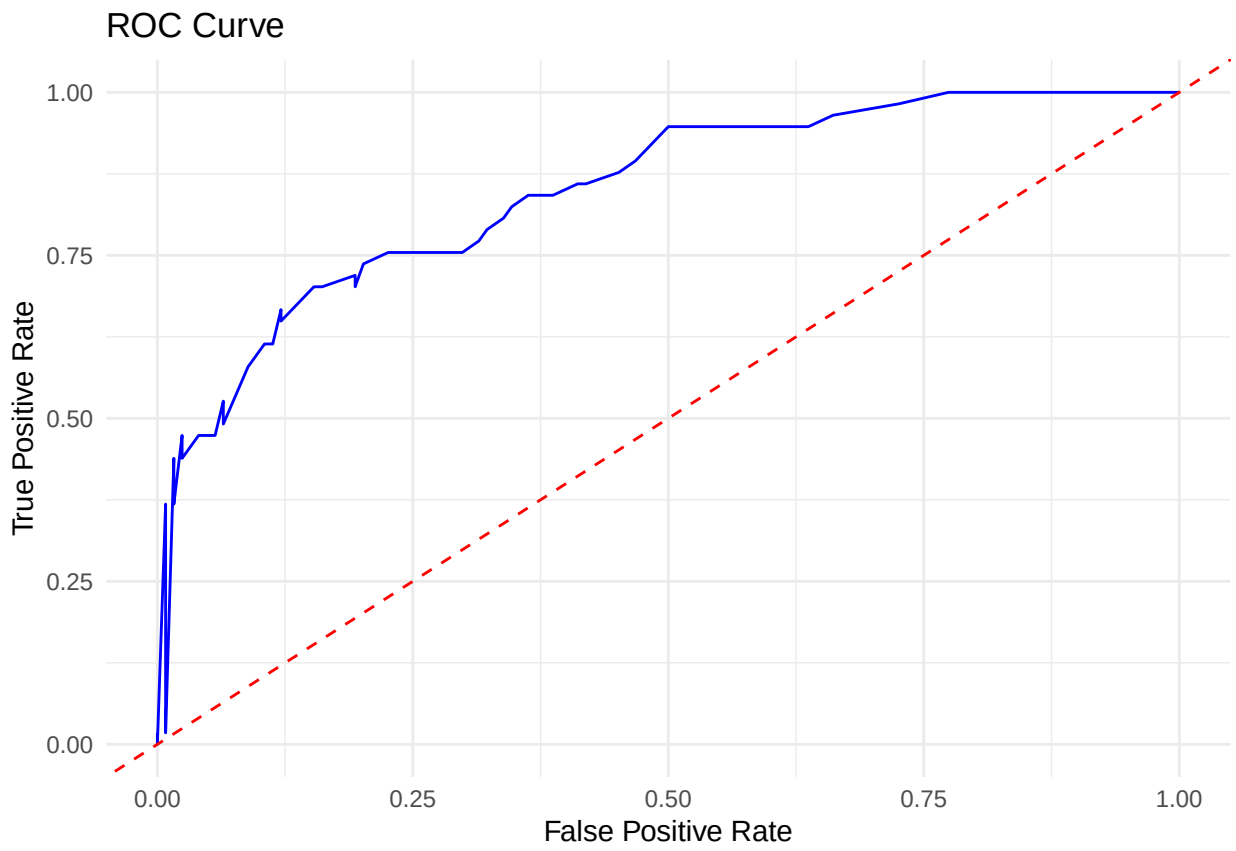
```
#prescision max of 1 and a min of 1, sensitivity max of 1 and a min of 0
presc<- runif(50, min=0, max=1)
sensi<-runif(50, min=0, max=1)
f1_bound<-(2*presc*sensi)/(presc+sensi)
summary(f1_bound)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01999 0.25369 0.42409 0.44265 0.65268 0.96153
```

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.

ANSWER:

```r
generate_roc <- function(df, class_col = "class", prob_col = "scored.probability") {

  # Extract true labels and predicted probabilities
  true_labels <- df[[class_col]]
  predicted_probs <- df[[prob_col]]

  # Initialize vectors for TPR and FPR
  thresholds <- seq(0, 1, by = 0.01)
  tpr <- numeric(length(thresholds))
  fpr <- numeric(length(thresholds))

  for (i in seq_along(thresholds)) {
    threshold <- thresholds[i]
    predicted_class <- ifelse(predicted_probs >= threshold, 1, 0)

    tp <- sum(predicted_class == 1 & true_labels == 1)
    fp <- sum(predicted_class == 1 & true_labels == 0)
    fn <- sum(predicted_class == 0 & true_labels == 1)
    tn <- sum(predicted_class == 0 & true_labels == 0)

    tpr[i] <- ifelse((tp + fn) == 0, 0, tp / (tp + fn))
    fpr[i] <- ifelse((fp + tn) == 0, 0, fp / (fp + tn))
  }

# Calculate AUC using trapezoidal rule

roc_df <- data.frame(FPR = fpr, TPR = tpr)
roc_df <- roc_df[order(roc_df$FPR), ]  # Ensure FPR is increasing
auc <- sum(diff(roc_df$FPR) * (head(roc_df$TPR, -1) + tail(roc_df$TPR, -1)) / 2)

# Create ROC plot

roc_plot <- ggplot(roc_df, aes(x = FPR, y = TPR)) +
  geom_line(color = "blue") +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "red") +
  labs(title = "ROC Curve", x = "False Positive Rate", y = "True Positive Rate") +
  theme_minimal()

return(list(plot = roc_plot, auc = auc))
}


result <- generate_roc(co_df)
print(result$plot)
```




```
print(result$auc)
```

```
## [1] 0.8484012
```

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

ANSWER:

```
classification_metrics <- function(df, actual_col, predicted_col, prob_col, positive = 1, threshold = 0
  # Metrics
  accuracy    <- prediction_accuracy(df, actual_col, predicted_col)
  error_rate  <- error_rate(df, actual_col, predicted_col)
  precision   <- pred_precision(df, actual_col, predicted_col)
  recall      <- sensitivity(df, actual_col, predicted_col)
  specificity <- specificity(df, actual_col, predicted_col)
  f1_score    <- F1_Score(df, actual_col, predicted_col, positive = 1)

  # AUC
  roc_result <- generate_roc(df, actual_col,prob_col)

  return(list(
    accuracy    = accuracy,
    error_rate  = error_rate,
```

```
    precision   = precision,
    sensitivity = recall,
    specificity = specificity,
    f1_score    = f1_score,
    auc         = roc_result$auc
  ))
}

metrics <- classification_metrics(co_df,"class","scored.class","scored.probability")
metrics
```

```
## $accuracy
## [1] 0.8066298
##
## $error_rate
## [1] 0.1933702
##
## $precision
## [1] 0.84375
##
## $sensitivity
## [1] 0.4736842
##
## $specificity
## [1] 0.9596774
##
## $f1_score
## [1] 0.6067416
##
## $auc
## [1] 0.8484012
```

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?

ANSWER: When comparing the sensitivity and specficity results from the caret package compare to our created functions, we saw the values were the same.

```
#Need to be factors to use confusionMatrix
co_df$scored.class<-as.factor(co_df$scored.class)
co_df$class<-as.factor(co_df$class)

b<-confusionMatrix(co_df$scored.class, co_df$class, positive="1")

caret_fn<-c(b$byClass["Sensitivity"], b$byClass["Specificity"])
Our_function<-c(sensitivity(co_df, "class", "scored.class"), specificity(co_df, "class", "scored.class"))

#will use cbind to create a table, using kable from knitr
c<-cbind(caret_fn, Our_function)

kable(c)
```

```
## Warning in attr(x, "align"): 'xfun::attr()' is deprecated.
```

```
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

```
## Warning in attr(x, "format"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

	caret_fn	Our_function
Sensitivity	0.4736842	0.4736842
Specificity	0.9596774	0.9596774

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?

ANSWER: The pROC package area under the curve is 0.850, our value of the area under the curve is 0.8484012. The pROC result seems to match well with our result. This 1% difference is likely due to natural variation between the two methods and falls within an acceptable range.

```
library(pROC)
```

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

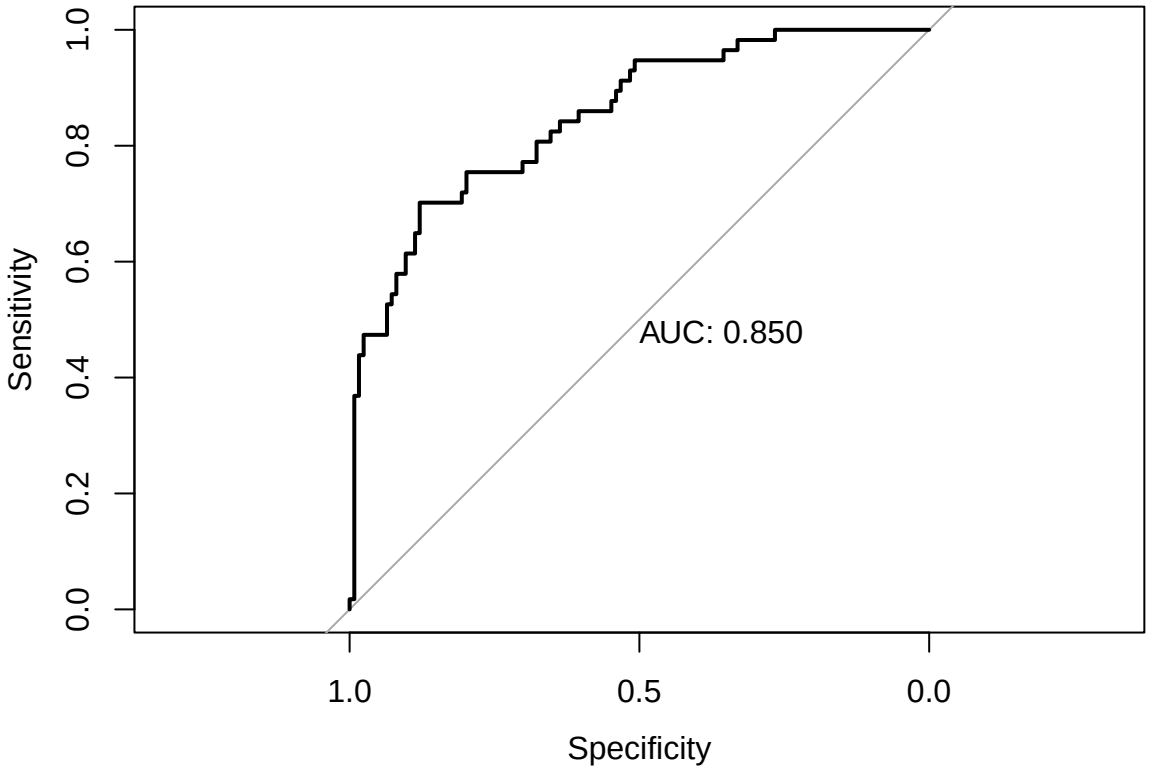
```
##
## Attaching package: 'pROC'
```

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

```
proc_plot <- roc(co_df$class,co_df$scored.probability, plot = T, print.auc = T)
```

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

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




```
proc_plot
```

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