Data 621 Homework 2: Appendix A

Group 2: William Aiken, Donald Butler, Michael Ippolito, Bharani Nittala, and Leticia Salazar

10-09-2022

Contents

Below find the code used:

Part 1:

Download the classification output data set.

```
# load data set
class_df <- read.csv("https://raw.githubusercontent.com/letisalba/Data_621/master/Homework_2/csv/classi
head(class_df, n = 4)
```

```
##   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
##   scored.class scored.probability
## 1            0         0.32845226
## 2            0         0.27319044
## 3            0         0.10966039
## 4            0         0.05599835
```

Getting an overview of the data: 181 observations of 11 variables

```
summary(class_df)
```

```
##     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
```

```
str(class_df)
```

```
## 'data.frame':    181 obs. of  11 variables:
##  $ pregnant          : int  7 2 3 1 4 1 9 8 1 2 ...
##  $ glucose           : int  124 122 107 91 83 100 89 120 79 123 ...
##  $ diastolic         : int  70 76 62 64 86 74 62 78 60 48 ...
##  $ skinfold          : int  33 27 13 24 19 12 0 0 42 32 ...
##  $ insulin           : int  215 200 48 0 0 46 0 0 48 165 ...
##  $ bmi               : num  25.5 35.9 22.9 29.2 29.3 19.5 22.5 25 43.5 42.1 ...
##  $ pedigree          : num  0.161 0.483 0.678 0.192 0.317 0.149 0.142 0.409 0.678 0.52 ...
##  $ age               : int  37 26 23 21 34 28 33 64 23 26 ...
##  $ class             : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ scored.class      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ scored.probability: num  0.328 0.273 0.11 0.056 0.1 ...
```

Part 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 data set. Make sure you understand the output. In particular, do the rows represent the actual or predicted class? The columns?

- The rows represent the actual class and columns represents the predicted class.

```
class_df %>%
  select(class, scored.class) %>%
  # re-coding to label the 0's and 1's
  mutate(class = recode(class,
                        '0' = 'Actual Negative',
                        '1' = 'Actual Positive'),
         scored.class = recode(scored.class,
                               '0' = 'Predicted Negative',
                               '1' = 'Predicted Positive')) %>%
  table()
```

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

Part 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.

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

```
func.accuracy <- function (class_df, actual, predict) {
  accuracy <- sum(class_df[actual] == class_df[predict]) / nrow(class_df)
  return (accuracy)
}

paste0('The accuracy of the predictions is ', round(func.accuracy(class_df,"class","scored.class"),5))
```

```
## [1] "The accuracy of the predictions is 0.80663"
```

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

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

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

```r
func.error_rate <- function(data) {
  df <- data %>%
    mutate(TP = ifelse(class == 1 & scored.class == 1,1,0),
           TN = ifelse(class == 0 & scored.class == 0,1,0),
           FP = ifelse(class == 0 & scored.class == 1,1,0),
           FN = ifelse(class == 1 & scored.class == 0,1,0))
  TP = sum(df$TP)
  TN = sum(df$TN)
  FP = sum(df$FP)
  FN = sum(df$FN)

  return((FP+FN)/(TP+FP+TN+FN))
}

error_rate <- func.error_rate(class_df)
#accuracy <- func.accuracy(class_df)
accuracy <- func.accuracy(class_df,"class","scored.class")

# Verify that you get an accuracy and an error rate that sums to one.
paste0('The accuracy and error rate sums to ', (accuracy + error_rate))
```

```
## [1] "The accuracy and error rate sums to 1"
```

Part 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.

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

```r
func.precision <- function(data) {
  df <- data %>%
    mutate(TP = ifelse(class == 1 & scored.class == 1,1,0),
           FP = ifelse(class == 0 & scored.class == 1,1,0))
  TP = sum(df$TP)
  FP = sum(df$FP)

  return(TP/(TP+FP))
}

paste0('The precision of the predictions is ',round(func.precision(class_df),5))
```

```
## [1] "The precision of the predictions is 0.84375"
```

Part 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.

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

- True Positive (TP) is the number of observations where the prediction and reference are both positive. False Negative (FN) is the number of observations where the prediction is negative, but the reference is positive.

```
func.sensitivity <- function(data) {
  df <- data %>%
    mutate(TP = ifelse(class == 1 & scored.class == 1,1,0),
           FN = ifelse(class == 1 & scored.class == 0,1,0))
  TP = sum(df$TP)
  FN = sum(df$FN)

  return(TP/(TP+FN))
}

paste0('The sensitivity of the predictions is ', round(func.sensitivity(class_df),5))
```

```
## [1] "The sensitivity of the predictions is 0.47368"
```

Part 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.

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

- True Negative (TN) is the number of observations where the prediction and reference are both negative. False Positive (FP) is the number of observations where the prediction is positive, but the reference is negative.

```
func.specificity <- function(data) {
  df <- data %>%
    mutate(TN = ifelse(class == 0 & scored.class == 0,1,0),
           FP = ifelse(class == 0 & scored.class == 1,1,0))
  TN = sum(df$TN)
  FP = sum(df$FP)

  return(TN/(TN+FP))
}

paste0('The specificity of the predictions is ', round(func.specificity(class_df),5))
```

```
## [1] "The specificity of the predictions is 0.95968"
```

Part 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.

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

```r
# Calculate precision using dynamic column names
calcPrecision <- function(df, actualCol, predictedCol) {
  TP <- nrow(df %>% dplyr::filter((!!sym(actualCol)) == 1 & (!!sym(predictedCol)) == 1))
  FP <- nrow(df %>% dplyr::filter((!!sym(actualCol)) == 0 & (!!sym(predictedCol)) == 1))
  return(TP / (TP + FP))
}

# Calculate sensitivity using dynamic column names
calcSensitivity <- function(df, actualCol, predictedCol) {
  TP <- nrow(df %>% dplyr::filter((!!sym(actualCol)) == 1 & (!!sym(predictedCol)) == 1))
  FN <- nrow(df %>% dplyr::filter((!!sym(actualCol)) == 1 & (!!sym(predictedCol)) == 0))
  return(TP / (TP + FN))
}

# Calculate F1 score
calcF1 <- function(df, actualCol, predictedCol) {
  tmp_precision <- calcPrecision(df, actualCol, predictedCol)
  tmp_sensitivity <- calcSensitivity(df, actualCol, predictedCol)
  return((2 * tmp_precision * tmp_sensitivity) / (tmp_precision + tmp_sensitivity))
}
print(paste0('The F1 score is ', round(calcF1(class_df, 'class', 'scored.class'), 3)))
```

```
## [1] "The F1 score is 0.607"
```

Part 9:

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)

There are four cases to consider:

1) Both precision and sensitivity approach unity.

$\lim_{p \to 1} \lim_{s \to 1} \frac{2ps}{p+s} = \frac{(2)(1)(1)}{(1+1)} = \frac{2}{2} = 1$

2) Precision approaches zero; sensitivity approaches unity.

$\lim_{p \to 0} \lim_{s \to 1} \frac{2ps}{p+s} = \frac{(2)(0)(1)}{(0+1)} = \frac{0}{1} = 0$

3) Precision approaches unity; sensitivity approaches zero.

$\lim_{p \to 1} \lim_{s \to 0} \frac{2ps}{p+s} = \frac{(2)(1)(0)}{(1+0)} = \frac{0}{1} = 0$

4) Both precision and sensitivity approach zero.

$\lim_{p \to 0} \lim_{s \to 0} \frac{2ps}{p+s}$

Using the rule

if 0 < p < 1 and 0 < s < 1, then ps < p

ps < p

and therefore

2ps < 2p

And using the rule

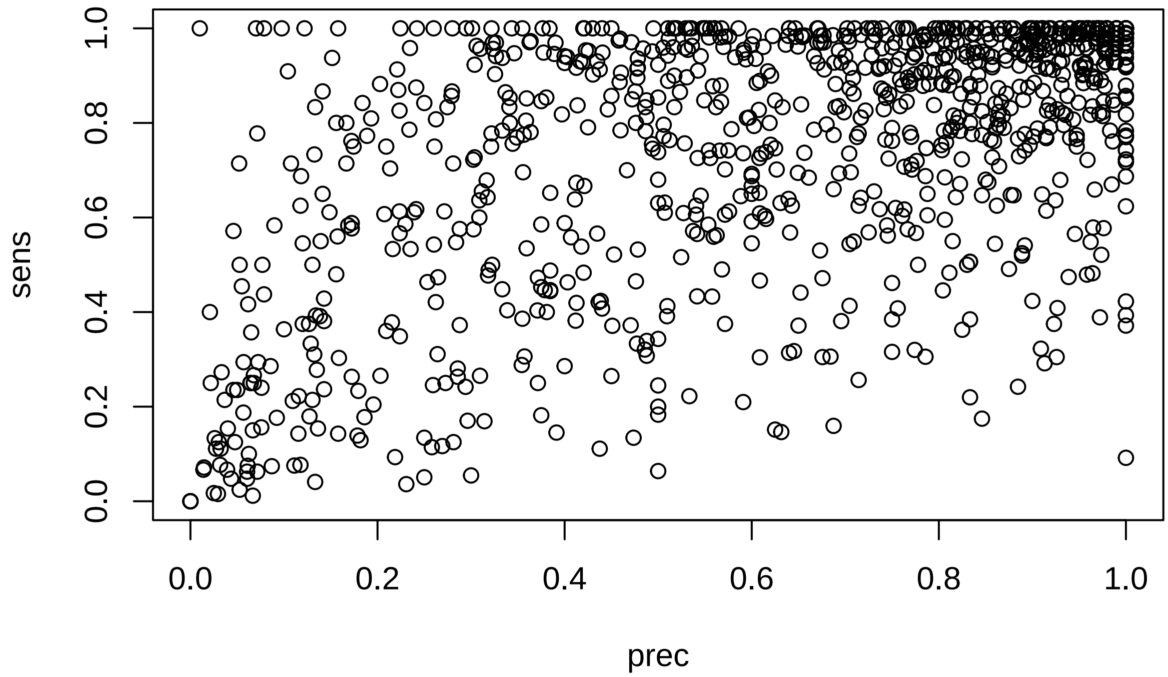
if 0 < p < 1 and 0 < s < 1, then ps < 1 and ps < (p + s)

then

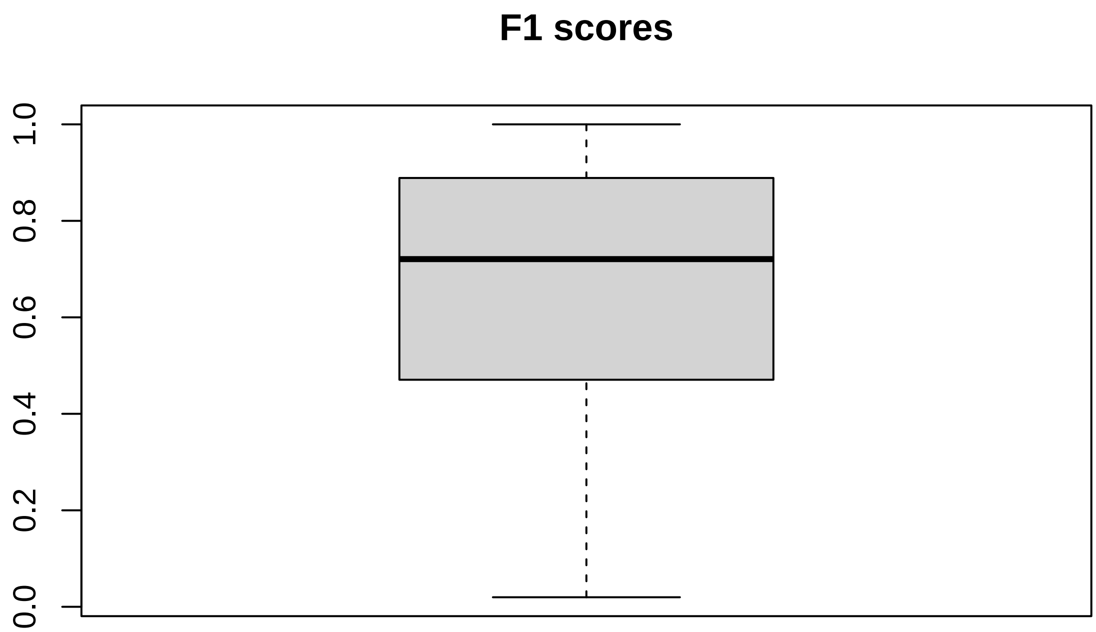
2ps < 2p and ps < p + s

$\frac{2ps}{p+s} < \frac{2p}{p+s} < \frac{something<ps}{something>ps} < 1$

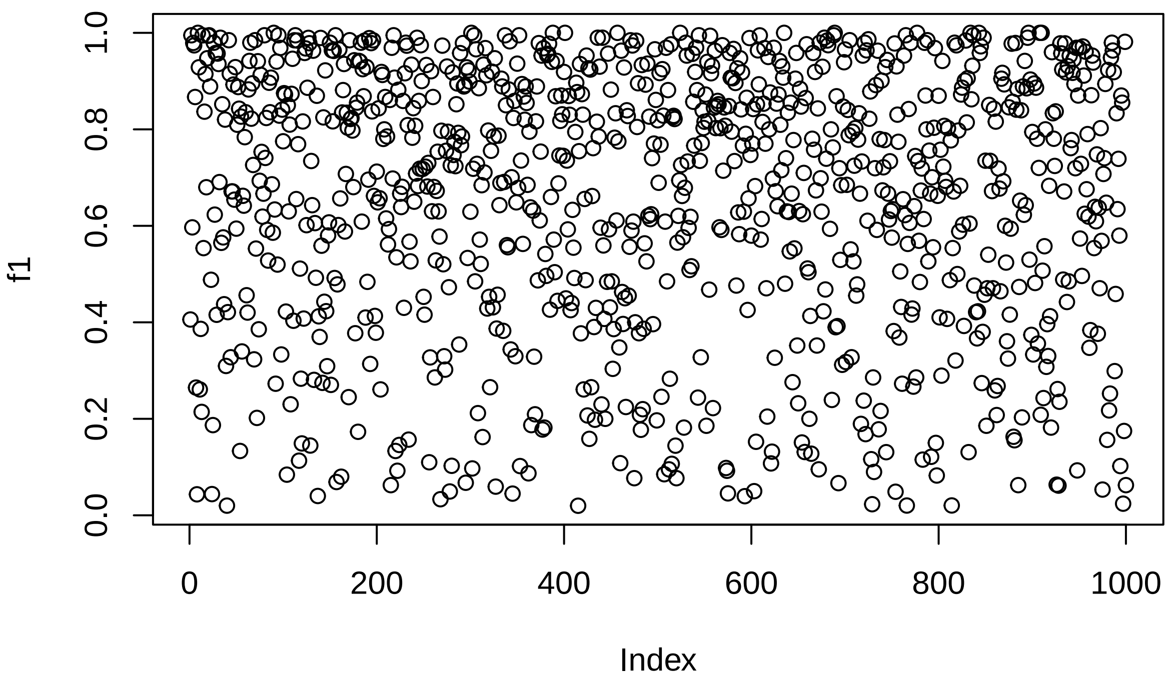
```
# Showing this empirically

# Create values of precision and sensitivity
n <- 100
prec <- c()
sens <- c()
for (i in seq(1, 1000)) {
  TP <- sample(seq(0, 100), size=1)
  FP <- sample(seq(0, n - TP), size=1)
  FN <- sample(seq(0, n - TP - FP), size=1)
  TN <- 100 - TP - FP - FN
  tmp_prec = TP / (TP + FP)
  tmp_sens = TP / (TP + FN)
  if (TP + TN + FP + FN != 100 | (TP + FP == 0 & TP + FN == 0)) {
    print(paste0('TP=', TP, ', TN=', TN, ', FP=', FP, ', FN=', FN, ', n=', TP + TN + FP + FN))
  }
  prec <- c(prec, tmp_prec)
  sens <- c(sens, tmp_sens)
}
plot(prec, sens)
```



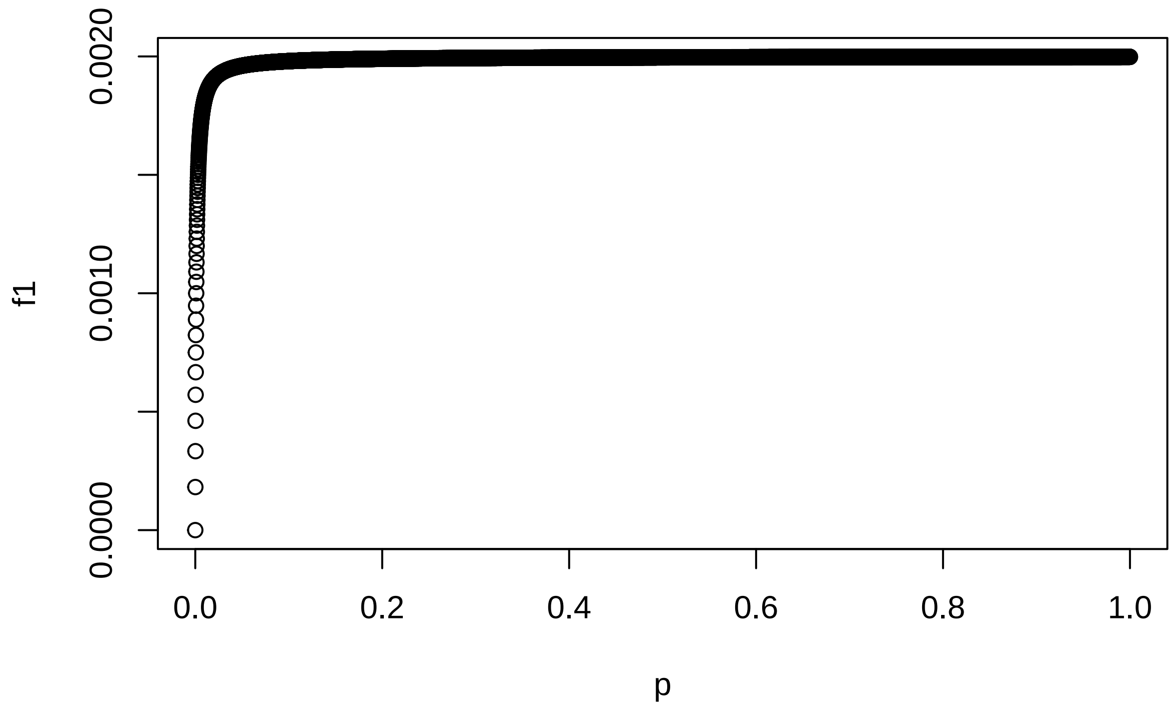

```
f1 <- (2*prec*sens)/(prec + sens)
boxplot(f1, main='F1 scores')
```




```
plot(f1)
```




```r
# Show that as precision approaches 0 while sensitivity is close to zero, the f1 score approaches zero.
p <- seq(0, 1, 0.0001)
s <- rep(0.001, length(p))
f1 <- (2 * p * s) / (p + s)
plot(p, f1)
```



Part 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).

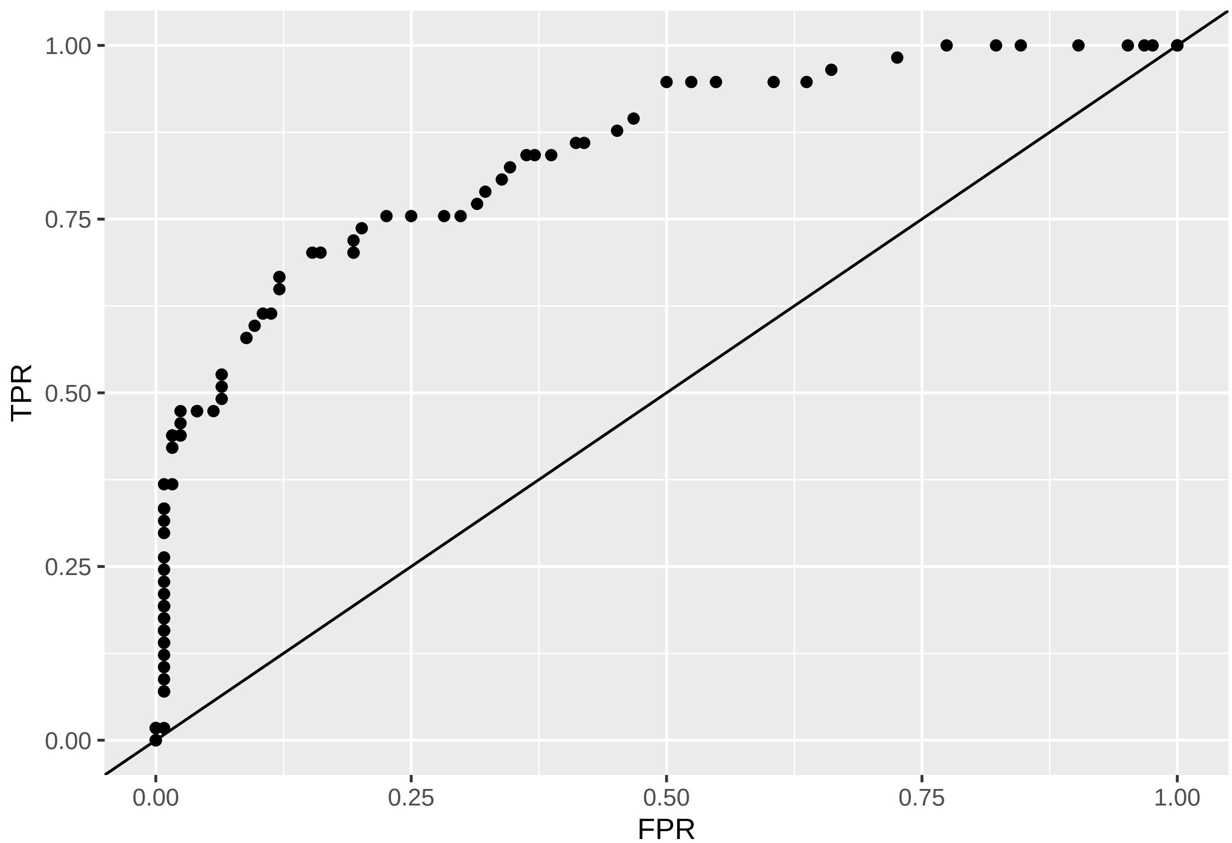
Note that I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.

```
roc_func <- function(data){
  temp_x <- rep(0, 101)
  temp_y <- rep(0, 101)
  temp_seq <- seq(from = 0, to = 1, by = 0.01)
  for (i in 1:length(temp_seq)){
   df <- data %>% mutate(scored.class = as.numeric(scored.probability > temp_seq[i])) %>%
    mutate(TP = ifelse(class == 1 & scored.class == 1,1,0),
           FP = ifelse(class == 0 & scored.class == 1,1,0),
           FN = ifelse(class == 1 & scored.class == 0,1,0),
           TN = ifelse(class == 0 & scored.class == 0,1,0))
  TPR = sum(df$TP)/(sum(df$TP) + sum(df$FN))
  FPR = sum(df$FP)/(sum(df$FP) + sum(df$TN))
```

```
    temp_x[i] <- FPR
    temp_y[i] <- TPR
  }
  temp_df <- bind_cols(temp_x, temp_y) %>% as.data.frame()
  names(temp_df) <- c("FPR", "TPR")
  plt <- ggplot2::ggplot(data = temp_df, aes(x = FPR, y = TPR)) + geom_point() + geom_abline()
  AUC <- pracma::trapz(temp_x,temp_y)
  output <- list(plt, AUC)
  return(output)
}

roc_func(class_df)
```

```
## New names:
## * `` -> `...1`
## * `` -> `...2`
```

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




```
##
## [[2]]
## [1] -0.8488964
```

Part 11:

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

```r
Accuracy <- round(func.accuracy(class_df,"class","scored.class"),5)
Accuracy
```

Accuracy

```
## [1] 0.80663
```

```r
Class_Error_Rate <- round(func.error_rate(class_df),5)
Class_Error_Rate
```

Classification Error Rate

```
## [1] 0.19337
```

```r
Precision <- round(func.precision(class_df),5)
Precision
```

Precision

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

```r
Sensitivity <- round(func.sensitivity(class_df),5)
Sensitivity
```

Sensitivity

```
## [1] 0.47368
```

```r
Specificity <- round(func.specificity(class_df),5)
Specificity
```

Specificity

```
## [1] 0.95968
```

```
F1_Score <- round(calcF1(class_df, 'class', 'scored.class'),5)
F1_Score
```

F1 Score

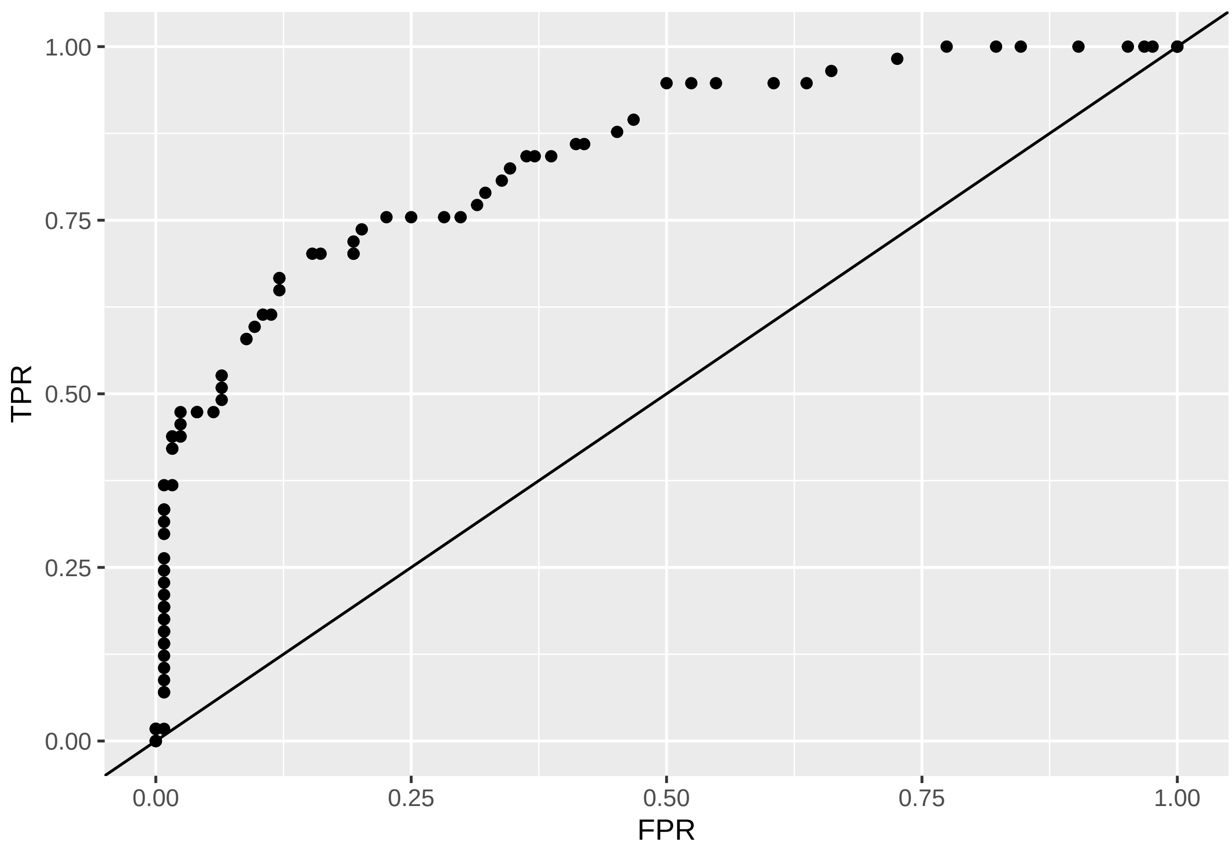
```
## [1] 0.60674
```

```
roc_func(class_df)
```

ROC and AUC

```
## New names:
## * `` -> `...1`
## * `` -> `...2`
```

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




```
##
## [[2]]
## [1] -0.8488964
```

Part 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?

```
caret <- confusionMatrix(as.factor(class_df$scored.class), as.factor(class_df$class), positive = "1")
caret
```

```
## 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
##
```

```
Sensitivity == caret$byClass["Sensitivity"]
```

These will load once Part 11 is completed:

```
## Sensitivity
##       FALSE
```

```
Specificity == caret$byClass["Specificity"]
```

```
## Specificity
##       FALSE
```

```
Accuracy == caret$overall["Accuracy"]
```

```
## Accuracy
##    FALSE
```

Part 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?

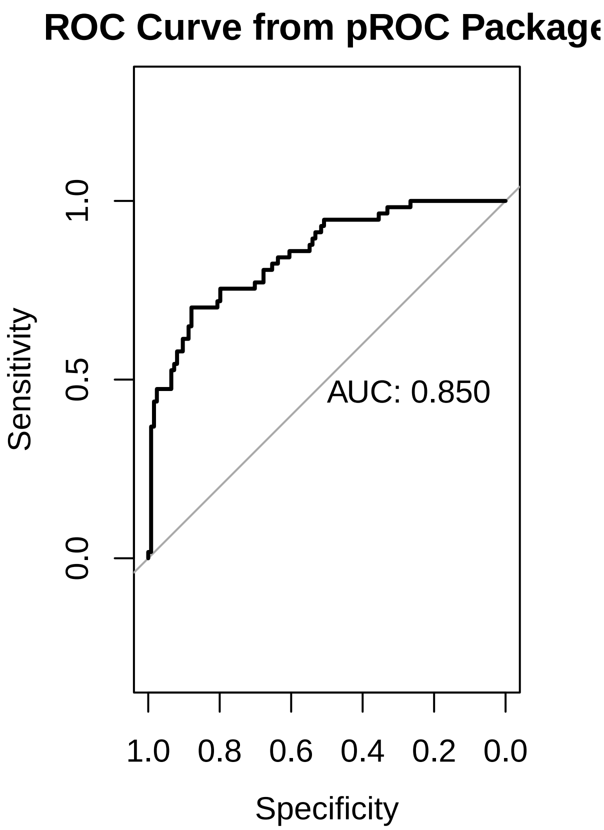
```
par(mfrow = c(1,2))

roc <- plot(roc(class_df$class, class_df$scored.probability), print.auc = TRUE, main = "ROC Curve from p
```

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

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

```
roc
```

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


ROC Curve from pROC Package
Sensitivity
1.0
0.5
0.0
AUC: 0.850
1.0
0.8
0.6
0.4
0.2
0.0
Specificity