HW2

Sam Reeves

1. Load the dataset.

```
df <- read.csv('classification-output-data.csv')
```

2. Use the table() function to get the raw confusion matrix for this scored dataset. What does it represent?

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

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

Here the upper left are true negative predictions, the bottom right are true positives. The rows are predicted values, and the columns are real.

3. Write a function that takes a data set as a dataframe, with actual and predicted values, and returns the accuracy.

```
return.accuracy <- function(df) {
  conf <- table(df$scored.class, df$class)

  neg.t <- conf[1,1]
  neg.f <- conf[1,2]
  pos.t <- conf[2,2]
  pos.f <- conf[2,1]

  accuracy <- (pos.t + neg.t) / (pos.t + pos.f + neg.t + neg.f)
  return(accuracy)
}
```

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.

```r
return.error <- function(df) {
  conf <- table(df$scored.class, df$class)

  neg.t <- conf[1,1]
  neg.f <- conf[1,2]
  pos.t <- conf[2,2]
  pos.f <- conf[2,1]

  error <- (pos.f + neg.f) / (pos.t + pos.f + neg.t + neg.f)
  return(error)
}
```

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.

```r
return.precision <- function(df) {
  conf <- table(df$scored.class, df$class)

  neg.t <- conf[1,1]
  neg.f <- conf[1,2]
  pos.t <- conf[2,2]
  pos.f <- conf[2,1]

  precision <- (pos.t) / (pos.t + pos.f)
  return(precision)
}
```

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.

```r
return.recall <- function(df) {
  conf <- table(df$scored.class, df$class)

  neg.t <- conf[1,1]
  neg.f <- conf[1,2]
  pos.t <- conf[2,2]
  pos.f <- conf[2,1]

  recall <- (pos.t) / (pos.t + neg.f)
  return(recall)
}
```

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.

```
return.specificity <- function(df) {
  conf <- table(df$scored.class, df$class)

  neg.t <- conf[1,1]
  neg.f <- conf[1,2]
  pos.t <- conf[2,2]
  pos.f <- conf[2,1]

  specificity <- (neg.t) / (pos.f + neg.t)
  return(specificity)
}
```

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.

```
return.f1 <- function(df = df) {
  prec <- return.precision(df)
  sens <- return.recall(df)

  f1 <- (2 * prec * sens) / (prec + sens)
  return(f1)
}
```

```
return.f1(df)
```

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

9. What are the bounds on the F1 score? Show that the F1 score will always be between 0 and 1.

The numerator will always be smaller than the denominator, except in the case where the numerator is 2. Always. These to complimentary fractions precision and sensitivity multiplied together with 2 will always be smaller than the two values added together.

So, this value cannot exceed 1 or become negative. These are the limits, the maximum is $\frac{2}{1}$ and minimum is $\frac{2}{\infty}$.

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.

```
return.rocd <- function(df) {
  class <- df$class
  prob <- round(df$scored.probability, 2)

  rocd <- data.frame(table(class, prob))
```

```
  rocd <- reshape(rocd, timevar = 'class',
                    idvar = 'prob', direction = 'wide')

  rocd$spec <- cumsum(rocd$Freq.0) / sum(rocd$Freq.0)
  rocd$sens <- cumsum(rocd$Freq.1) / sum(rocd$Freq.1)

  fig <- plot(1 - rocd$spec, 1 - rocd$sens, type = 'l')

  rocd$x <- 0
  rocd$y <- 0
  rocd$auc <- 0
  rocd$`1 - spec` <- 1 - rocd$spec

  for(i in 1:(dim(rocd)[1]-1)) {
    rocd$x[i] <- abs(rocd$`1 - spec`[i+1] - rocd$`1 - spec`[i])
    rocd$y[i] <- abs(rocd$sens[i+1] - rocd$sens[i])
    rocd$auc[i] <- rocd$x[i] * (rocd$sens[i] + rocd$sens[i+1]) / 2
  }

  auc <- sum(rocd$auc)
  return(list(rocd, fig, auc))
}
```

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

```
return.accuracy(df)
```

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

```
return.error(df)
```

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

```
return.precision(df)
```

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

```
return.recall(df)
```

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

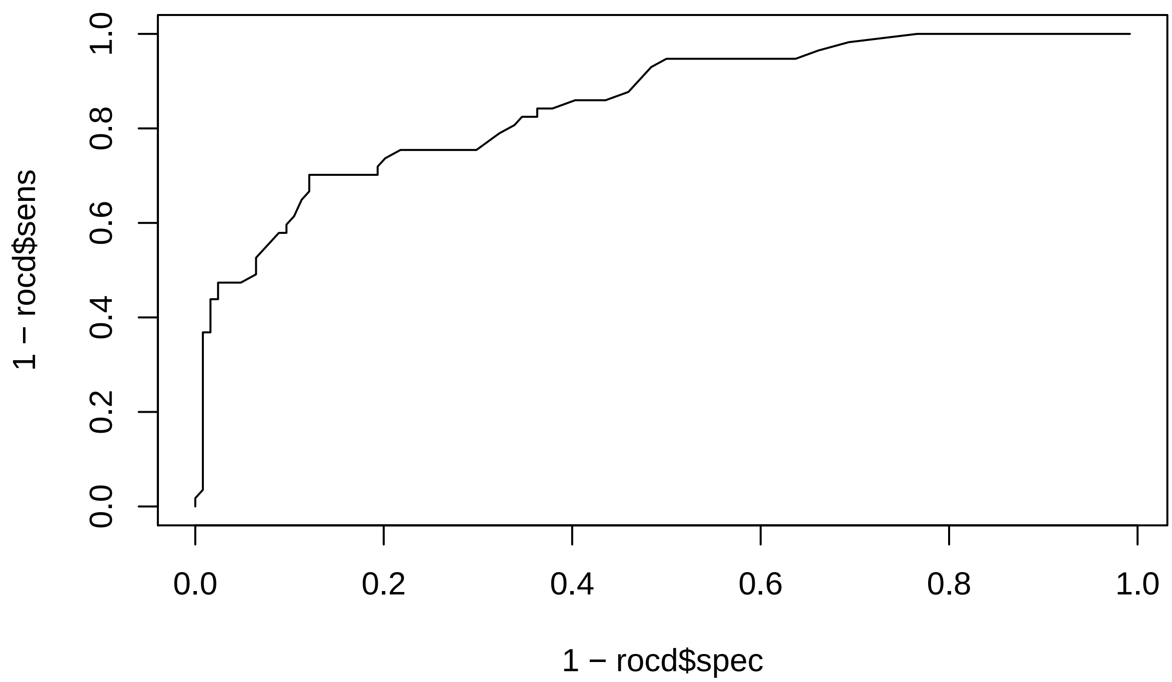
```
return.specificity(df)
```

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

```
return.f1(df)
```

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

```
return.rocd(df)
```




```
## [[1]]
##     prob Freq.0 Freq.1        spec       sens           x          y
## 1   0.02      1      0 0.008064516 0.00000000 0.024193548 0.00000000
## 3   0.03      3      0 0.032258065 0.00000000 0.040322581 0.00000000
## 5   0.05      5      0 0.072580645 0.00000000 0.048387097 0.00000000
## 7   0.06      6      0 0.120967742 0.00000000 0.032258065 0.00000000
## 9   0.07      4      0 0.153225806 0.00000000 0.048387097 0.00000000
## 11  0.08      6      0 0.201612903 0.00000000 0.032258065 0.00000000
## 13  0.09      4      0 0.233870968 0.00000000 0.072580645 0.01754386
## 15   0.1      9      1 0.306451613 0.01754386 0.032258065 0.01754386
## 17  0.11      4      1 0.338709677 0.03508772 0.024193548 0.01754386
## 19  0.12      3      1 0.362903226 0.05263158 0.048387097 0.00000000
## 21  0.13      6      0 0.411290323 0.05263158 0.056451613 0.00000000
## 23  0.14      7      0 0.467741935 0.05263158 0.032258065 0.00000000
## 25  0.15      4      0 0.500000000 0.05263158 0.016129032 0.01754386
## 27  0.16      2      1 0.516129032 0.07017544 0.024193548 0.05263158
## 29  0.17      3      3 0.540322581 0.12280702 0.024193548 0.01754386
## 31  0.18      3      1 0.564516129 0.14035088 0.016129032 0.00000000
```

```
## 33  0.19      2      0 0.580645161 0.14035088 0.016129032 0.00000000
## 35   0.2      2      0 0.596774194 0.14035088 0.024193548 0.01754386
## 37  0.21      3      1 0.620967742 0.15789474 0.016129032 0.00000000
## 39  0.22      2      0 0.637096774 0.15789474 0.000000000 0.01754386
## 41  0.23      0      1 0.637096774 0.17543860 0.016129032 0.00000000
## 43  0.24      2      0 0.653225806 0.17543860 0.008064516 0.01754386
## 45  0.25      1      1 0.661290323 0.19298246 0.016129032 0.01754386
## 47  0.26      2      1 0.677419355 0.21052632 0.024193548 0.03508772
## 49  0.27      3      2 0.701612903 0.24561404 0.008064516 0.00000000
## 51  0.28      1      0 0.709677419 0.24561404 0.016129032 0.00000000
## 53  0.29      2      0 0.725806452 0.24561404 0.048387097 0.00000000
## 55   0.3      6      0 0.774193548 0.24561404 0.008064516 0.00000000
## 57  0.31      1      0 0.782258065 0.24561404 0.016129032 0.01754386
## 59  0.32      2      1 0.798387097 0.26315789 0.008064516 0.01754386
## 61  0.33      1      1 0.806451613 0.28070175 0.000000000 0.01754386
## 63  0.34      0      1 0.806451613 0.29824561 0.016129032 0.00000000
## 65  0.35      2      0 0.822580645 0.29824561 0.024193548 0.00000000
## 67  0.36      3      0 0.846774194 0.29824561 0.032258065 0.00000000
## 69  0.37      4      0 0.879032258 0.29824561 0.000000000 0.03508772
## 71  0.38      0      2 0.879032258 0.33333333 0.008064516 0.01754386
## 73   0.4      1      1 0.887096774 0.35087719 0.008064516 0.03508772
## 75  0.41      1      2 0.895161290 0.38596491 0.008064516 0.01754386
## 77  0.42      1      1 0.903225806 0.40350877 0.000000000 0.01754386
## 79  0.43      0      1 0.903225806 0.42105263 0.008064516 0.00000000
## 81  0.44      1      0 0.911290323 0.42105263 0.008064516 0.01754386
## 83  0.45      1      1 0.919354839 0.43859649 0.016129032 0.03508772
## 85  0.46      2      2 0.935483871 0.47368421 0.000000000 0.01754386
## 87  0.47      0      1 0.935483871 0.49122807 0.000000000 0.01754386
## 89  0.48      0      1 0.935483871 0.50877193 0.016129032 0.01754386
## 91  0.49      2      1 0.951612903 0.52631579 0.008064516 0.00000000
## 93   0.5      1      0 0.959677419 0.52631579 0.016129032 0.00000000
## 95  0.52      2      0 0.975806452 0.52631579 0.000000000 0.01754386
## 97  0.53      0      1 0.975806452 0.54385965 0.000000000 0.01754386
## 99  0.55      0      1 0.975806452 0.56140351 0.008064516 0.00000000
## 101 0.56      1      0 0.983870968 0.56140351 0.000000000 0.01754386
## 103 0.59      0      1 0.983870968 0.57894737 0.000000000 0.05263158
## 105 0.62      0      3 0.983870968 0.63157895 0.008064516 0.00000000
## 107 0.63      1      0 0.991935484 0.63157895 0.000000000 0.03508772
## 109 0.64      0      2 0.991935484 0.66666667 0.000000000 0.01754386
## 111 0.66      0      1 0.991935484 0.68421053 0.000000000 0.03508772
## 113 0.68      0      2 0.991935484 0.71929825 0.000000000 0.01754386
## 115 0.69      0      1 0.991935484 0.73684211 0.000000000 0.01754386
## 117  0.7      0      1 0.991935484 0.75438596 0.000000000 0.01754386
## 119 0.71      0      1 0.991935484 0.77192982 0.000000000 0.03508772
## 121 0.72      0      2 0.991935484 0.80701754 0.000000000 0.01754386
## 123 0.76      0      1 0.991935484 0.82456140 0.000000000 0.01754386
## 125 0.78      0      1 0.991935484 0.84210526 0.000000000 0.01754386
## 127 0.81      0      1 0.991935484 0.85964912 0.000000000 0.01754386
## 129 0.83      0      1 0.991935484 0.87719298 0.000000000 0.03508772
## 131 0.85      0      2 0.991935484 0.91228070 0.000000000 0.01754386
## 133 0.86      0      1 0.991935484 0.92982456 0.000000000 0.03508772
## 135 0.88      0      2 0.991935484 0.96491228 0.008064516 0.01754386
## 137 0.89      1      1 1.000000000 0.98245614 0.000000000 0.01754386
## 139 0.95      0      1 1.000000000 1.00000000 0.000000000 0.00000000
```

```
##              auc    1 - spec
## 1   0.0000000000 0.991935484
## 3   0.0000000000 0.967741935
## 5   0.0000000000 0.927419355
## 7   0.0000000000 0.879032258
## 9   0.0000000000 0.846774194
## 11  0.0000000000 0.798387097
## 13  0.0006366723 0.766129032
## 15  0.0008488964 0.693548387
## 17  0.0010611205 0.661290323
## 19  0.0025466893 0.637096774
## 21  0.0029711375 0.588709677
## 23  0.0016977929 0.532258065
## 25  0.0009903792 0.500000000
## 27  0.0023344652 0.483870968
## 29  0.0031833616 0.459677419
## 31  0.0022637238 0.435483871
## 33  0.0022637238 0.419354839
## 35  0.0036078098 0.403225806
## 37  0.0025466893 0.379032258
## 39  0.0000000000 0.362903226
## 41  0.0028296548 0.362903226
## 43  0.0014855688 0.346774194
## 45  0.0032541030 0.338709677
## 47  0.0055178268 0.322580645
## 49  0.0019807583 0.298387097
## 51  0.0039615167 0.290322581
## 53  0.0118845501 0.274193548
## 55  0.0019807583 0.225806452
## 57  0.0041029994 0.217741935
## 59  0.0021929825 0.201612903
## 61  0.0000000000 0.193548387
## 63  0.0048104131 0.193548387
## 65  0.0072156197 0.177419355
## 67  0.0096208263 0.153225806
## 69  0.0000000000 0.120967742
## 71  0.0027589134 0.120967742
## 73  0.0029711375 0.112903226
## 75  0.0031833616 0.104838710
## 77  0.0000000000 0.096774194
## 79  0.0033955857 0.096774194
## 81  0.0034663271 0.088709677
## 83  0.0073571024 0.080645161
## 85  0.0000000000 0.064516129
## 87  0.0000000000 0.064516129
## 89  0.0083474816 0.064516129
## 91  0.0042444822 0.048387097
## 93  0.0084889643 0.040322581
## 95  0.0000000000 0.024193548
## 97  0.0000000000 0.024193548
## 99  0.0045274477 0.024193548
## 101 0.0000000000 0.016129032
## 103 0.0000000000 0.016129032
## 105 0.0050933786 0.016129032
```

```
## 107 0.0000000000 0.008064516
## 109 0.0000000000 0.008064516
## 111 0.0000000000 0.008064516
## 113 0.0000000000 0.008064516
## 115 0.0000000000 0.008064516
## 117 0.0000000000 0.008064516
## 119 0.0000000000 0.008064516
## 121 0.0000000000 0.008064516
## 123 0.0000000000 0.008064516
## 125 0.0000000000 0.008064516
## 127 0.0000000000 0.008064516
## 129 0.0000000000 0.008064516
## 131 0.0000000000 0.008064516
## 133 0.0000000000 0.008064516
## 135 0.0078522920 0.008064516
## 137 0.0000000000 0.000000000
## 139 0.0000000000 0.000000000
##
## [[2]]
## NULL
##
## [[3]]
## [1] 0.1494765
```

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?

```
confusionMatrix(data = as.factor(df$scored.class),
                reference = as.factor(df$class))
```

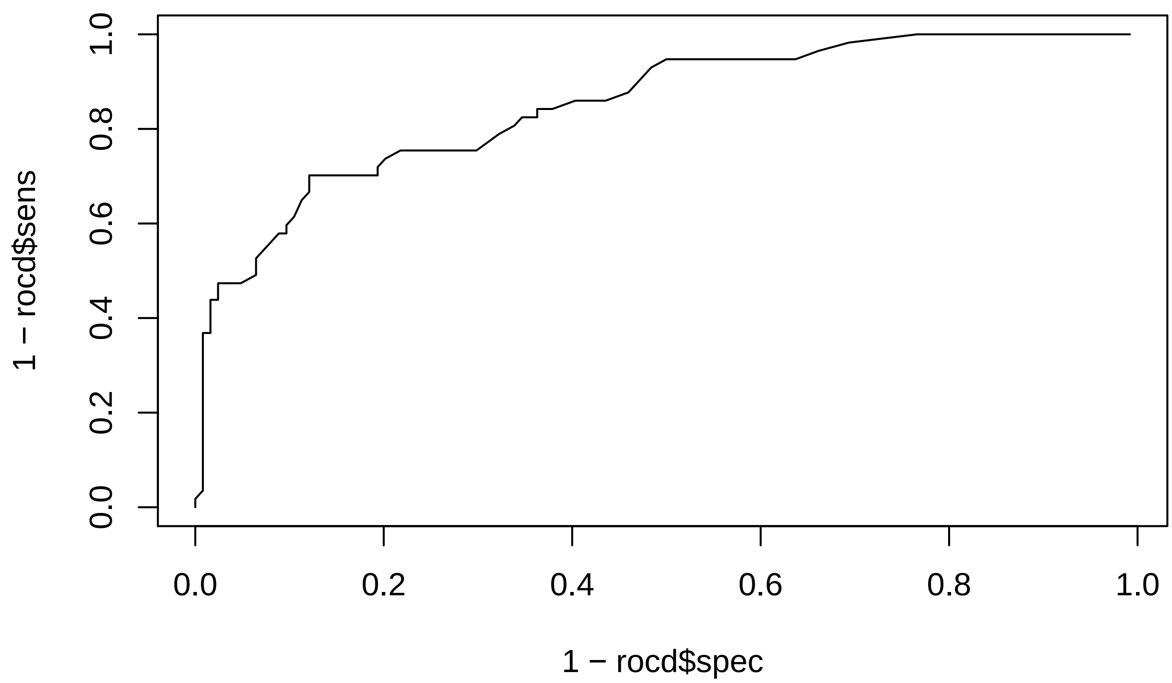
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 119  30
##          1   5  27
##
##                Accuracy : 0.8066
##                  95% CI : (0.7415, 0.8615)
##     No Information Rate : 0.6851
##     P-Value [Acc > NIR] : 0.0001712
##
##                   Kappa : 0.4916
##
##  Mcnemar's Test P-Value : 4.976e-05
##
##             Sensitivity : 0.9597
##             Specificity : 0.4737
##          Pos Pred Value : 0.7987
##          Neg Pred Value : 0.8438
##              Prevalence : 0.6851
```

```
##          Detection Rate : 0.6575
##    Detection Prevalence : 0.8232
##       Balanced Accuracy : 0.7167
##
##        'Positive' Class : 0
##
```

Holy cow! They're the same!

13. Investigate the procd package. Use it to generate an rocd curve for the data set. How do the results compare with your own functions?

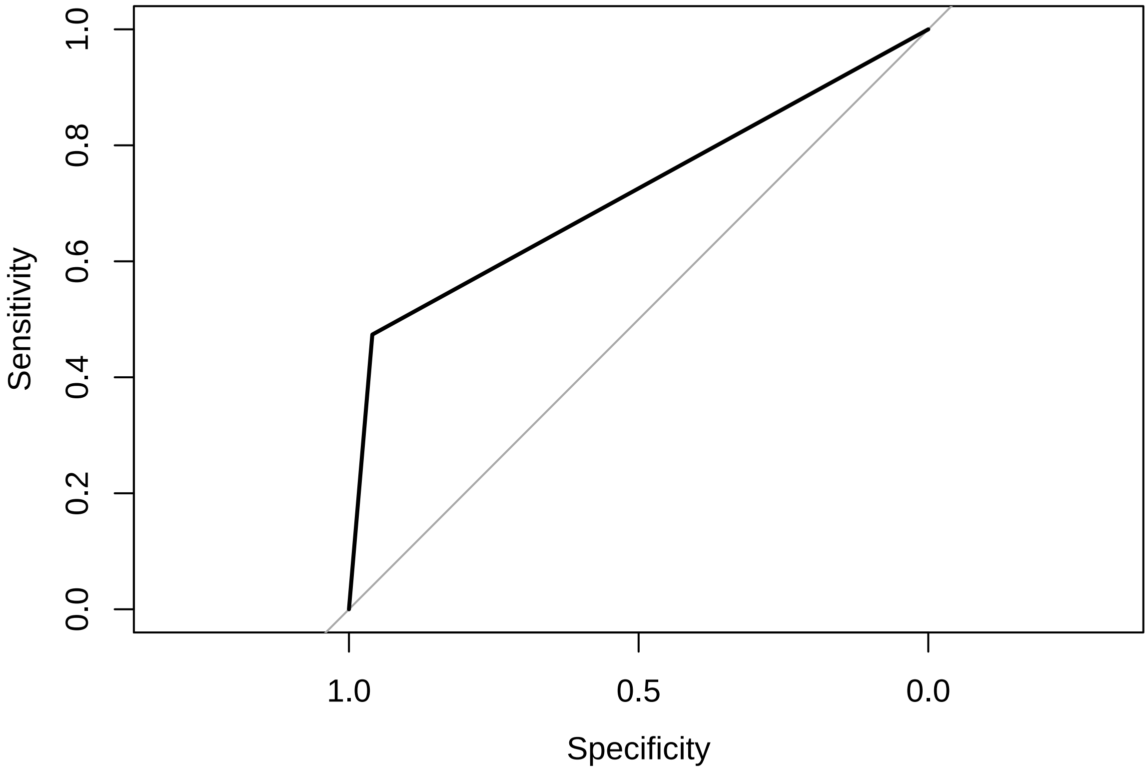
```
d <- return.rocd(df)[[1]]
```




```
plot(roc(response = df$class,
    predictor = df$scored.class))
```

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

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



I'm confused about this error. I think my function is correct, though.