Critical Thinking Group 4 - HW2

Sreejaya, Suman, Vuthy

September 27, 2016

Overview

In this assignment, we will work through various classification metrics. We will implement the various functions to calculate the metrics as well as explore functions from various packages that perform equivalent calculations. We will also generate graphical outputs used to evaluate the classification models.

Dataset
Classification Output

Data Exploration

The *classification output* dataset contains 181 observations and 11 variables. All 11 variables are numeric including the 3 variables we are interested in:

- **class**: the actual class for the observations
- **score.class**: the predicted class for the observations (based on a threshold of 0.5)
- **scored.probability**: the predicted probability of success for the observation

```
## Observations: 181
## Variables: 11
## $ pregnant           <int> 7, 2, 3, 1, 4, 1, 9, 8, 1, 2, 5, 5, 13, 0, ...
## $ glucose            <int> 124, 122, 107, 91, 83, 100, 89, 120, 79, 12...
## $ diastolic          <int> 70, 76, 62, 64, 86, 74, 62, 78, 60, 48, 78,...
## $ skinfold           <int> 33, 27, 13, 24, 19, 12, 0, 0, 42, 32, 30, 4...
## $ insulin            <int> 215, 200, 48, 0, 0, 46, 0, 0, 48, 165, 0, 7...
## $ bmi                <dbl> 25.5, 35.9, 22.9, 29.2, 29.3, 19.5, 22.5, 2...
## $ pedigree           <dbl> 0.161, 0.483, 0.678, 0.192, 0.317, 0.149, 0...
## $ age                <int> 37, 26, 23, 21, 34, 28, 33, 64, 23, 26, 37,...
## $ class              <int> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1...
## $ scored.class       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1...
## $ scored.probability <dbl> 0.32845226, 0.27319044, 0.10966039, 0.05599...
```

Data Preparation

A confusion matrix is typically used to describe the performance of a classification model. The matrix contains 2 rows and 2 columns that reports the number of true negative, false negative, false positive and true positive. The values in the dataset needs to be factors in order to be classified using the table() function.

Since all the variables in the dataset were numeric, *class* and *score.class* variables were converted to factors.

```
## Observations: 181
## Variables: 11
## $ pregnant           <int> 7, 2, 3, 1, 4, 1, 9, 8, 1, 2, 5, 5, 13, 0, ...
## $ glucose            <int> 124, 122, 107, 91, 83, 100, 89, 120, 79, 12...
## $ diastolic          <int> 70, 76, 62, 64, 86, 74, 62, 78, 60, 48, 78,...
## $ skinfold           <int> 33, 27, 13, 24, 19, 12, 0, 0, 42, 32, 30, 4...
## $ insulin            <int> 215, 200, 48, 0, 0, 46, 0, 0, 48, 165, 0, 7...
## $ bmi                <dbl> 25.5, 35.9, 22.9, 29.2, 29.3, 19.5, 22.5, 2...
## $ pedigree           <dbl> 0.161, 0.483, 0.678, 0.192, 0.317, 0.149, 0...
## $ age                <int> 37, 26, 23, 21, 34, 28, 33, 64, 23, 26, 37,...
## $ class              <fctr> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ scored.class       <fctr> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ scored.probability <dbl> 0.32845226, 0.27319044, 0.10966039, 0.05599...
```

```
##  class   scored.class scored.probability
##  0:124   0:149        Min.   :0.02323
##  1: 57   1: 32        1st Qu.:0.11702
##                       Median :0.23999
##                       Mean   :0.30373
##                       3rd Qu.:0.43093
##                       Max.   :0.94633
```

The resulting confusion matrix where rows represents the Actual class and columns represent the predicted class.

Table 1: Confusion Matrix

	0	1
0	119	5
1	30	27

Classification Metrics

Please see Appendix (Appendix section) for custom implementation details.

- Accuracy = How often is the classifier correct?

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

- Classification Error Rate = How often is the classifier incorrect?

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

- Precision = When the classifier predicts yes, how often is it correct?

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

- Sensitivity = When the actual value is yes, how often does it predict yes?

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

- Specificity = When the actual value is no, how often does it predict no?

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

- F1 Score = Weighted average of the sensitivity and precision

$$\frac{2 X Precision X Sensitivity}{Precision + Sensitivity}$$

Metrics definition sourced from **Data School**

Table 2: Confusion Matrix Metrics using custom functions

Metric	Value
Accuracy	0.8066
Sn(True Positive Rate)	0.4737
Sp(True Negative Rate)	0.9597
ClassErrorRate	0.1934
F1Score	0.6067
Precision	0.8438

Confusion Matrix

```
confusionMatrix(class.df$scored.class, class.df$class, positive = "1")
```

```
## 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

```
round(caret::sensitivity(factor(class.df$scored.class), factor(class.df$class), positive = "1"),
    4)
```

```
## [1] 0.4737
```

Specificity

```
round(caret::specificity(factor(class.df$scored.class), factor(class.df$class), negative = "0"),
    4)
```

```
## [1] 0.9597
```

We compared the caret confusionmatrix with the one we created; all the measures are matched with the manually created one. Specificity and sensitivity are also same as we calculated.

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.

F1 score reaches its best value at 1 and worst at 0.since both precision and sensitivity are in between 0 and 1, we know

$$0 < p * s <= 1$$

and

$$2 * p * s <= p + s$$

(maximum value p and s can have is 1), so

$$2 * p * s / p + s$$

can have values in between 0 and 1.

Graphical Evaluation

ROC Curve

We will use the *ROC Curve* to characterize the sensitivity/specificity tradeoffs for our binary classifier (class in our example) . We will have a series of threshold levels (a.k.a Cut-Offs) between zero and one in inrements of 0.01, and then with the help of the probability column (scored.probability) we will find out the sensitivity and specificity at each of the threshold levels. Basically, the cases with 'scored.probability' equal to or above the 'threshold' are classified as positive,and the ones below the threshold are classified as negative.So, different thresholds give different levels of sensitivity and specificity.

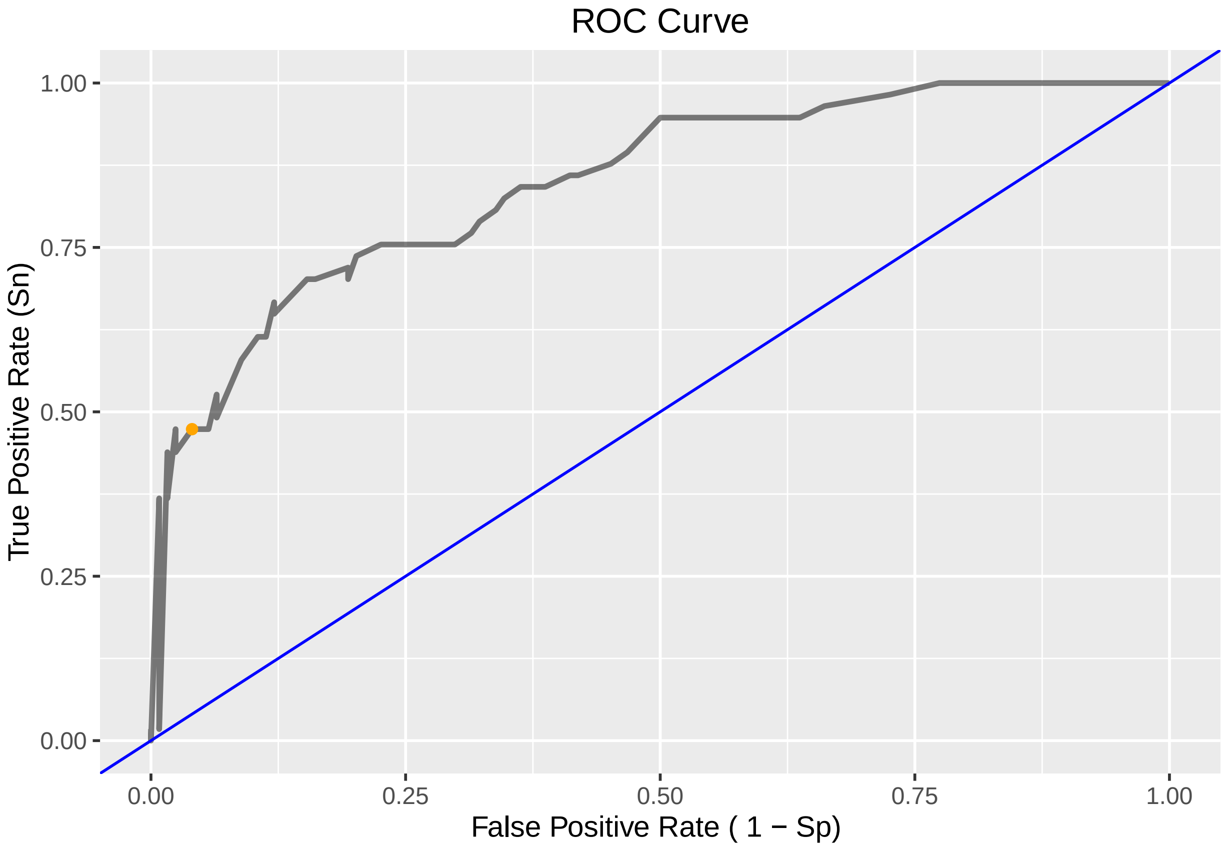
A high threshold is generally results in labelling more negative cases and low threshold produces more postive labels.

And the *ROC Curve* plots the **true positive rate against the false positive rate**.

Here is the RoC Curve along with few threshold rows from our custom R function **getRoCData**:

Threshold	Sn	Sp
0.00	1.0000000	0.0000000
0.01	1.0000000	0.0000000
0.02	1.0000000	0.0000000
0.03	1.0000000	0.0241935
0.04	1.0000000	0.0322581
0.05	1.0000000	0.0483871
0.06	1.0000000	0.0967742
0.07	1.0000000	0.1532258
0.08	1.0000000	0.1774194
0.09	1.0000000	0.2258065
0.10	0.9824561	0.2741935
0.11	0.9649123	0.3387097
0.12	0.9473684	0.3629032

Threshold	Sn	Sp
0.13	0.9473684	0.3951613
0.14	0.9473684	0.4516129
0.15	0.9473684	0.4758065
0.16	0.9473684	0.5000000
0.17	0.8947368	0.5322581
0.18	0.8771930	0.5483871
0.19	0.8596491	0.5806452



```
##   [1] 0.0000000000 0.0000000000 0.0000000000 0.0002419355 0.0005645161
##   [6] 0.0010483871 0.0020161290 0.0035483871 0.0053225806 0.0075806452
##  [11] 0.0101471420 0.0131833616 0.0162860781 0.0197113752 0.0237011885
##  [16] 0.0279329372 0.0324066214 0.0366765705 0.0409323713 0.0453353141
##  [21] 0.0498189021 0.0543689870 0.0590803622 0.0638723826 0.0686502547
##  [26] 0.0733333333 0.0780022637 0.0825764007 0.0871363894 0.0918576684
##  [31] 0.0969015280 0.1021873231 0.1075396152 0.1127971138 0.1178791737
##  [36] 0.1229612337 0.1283658744 0.1338511602 0.1393081494 0.1447651387
##  [41] 0.1500466893 0.1550580079 0.1601499717 0.1651471420 0.1700495190
##  [46] 0.1749518959 0.1795698925 0.1840124505 0.1882795699 0.1924518959
##  [51] 0.1967855122 0.2011191285 0.2054527448 0.2099476514 0.2142671194
##  [56] 0.2184111488 0.2225551783 0.2267798529 0.2310045274 0.2352292020
##  [61] 0.2392784380 0.2433276740 0.2468505942 0.2504541596 0.2537068478
##  [66] 0.2569595359 0.2602122241 0.2632894737 0.2661912847 0.2687422184
##  [71] 0.2711177136 0.2733177702 0.2753423882 0.2771915676 0.2790407470
##  [76] 0.2808899264 0.2827391058 0.2844128466 0.2860865874 0.2875848896
##  [81] 0.2890831919 0.2905814941 0.2919043577 0.2932272213 0.2943746463
##  [86] 0.2953466327 0.2961431805 0.2967642898 0.2973853990 0.2974801924
##  [91] 0.2976556310 0.2978310696 0.2980065082 0.2981819468 0.2983573854
##  [96] 0.2983573854 0.2983573854 0.2983573854 0.2983573854 0.2983573854
## [101] 0.2983573854
```

Our RoC curve follows the left-hand border and then the top border of the ROC space, so it indicates a good test.

Appendix

1. Download the classification output data set

```
classification.orig <- read.csv("https://raw.githubusercontent.com/Nguyver/DATA621-HW/master/HW2/classi
    header = TRUE, sep = ",", stringsAsFactors = FALSE)
```

2. The data set has three key columns we will use:

- **class**: the actual class for the observations
- **score.class**: the predicted class for the observations (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?

In our case, the columns are the predictions and the rows are the actuals.

```
class.df <- classification.orig

# Convert numbers to factors
class.df$class <- factor(class.df$class)
class.df$scored.class <- factor(class.df$scored.class)

glimpse(class.df)
summary(class.df)

cnf.mtx <- table(class.df$class, class.df$scored.class)
knitr::kable(cnf.mtx)

# Rows represent Actual class and the columns represent Predicted class.

# Verifying Table results = Predict-Actual True Positive = 1-1 False Positive =
# 1-0 True Negative = 0-0 False Negative = 0-1 sum(class.df$class ==
# class.df$scored.class & class.df$class ==1) #TP (2,2) sum(class.df$class !=
# class.df$scored.class & class.df$class ==0) #FN (1,2)

# sum(class.df$class == class.df$scored.class & class.df$class ==0) #TN (1,1)
# sum(class.df$class != class.df$scored.class & class.df$class ==1) #FP (2,1)
```

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.

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

Accuracy: Overall, how often is the classifier correct?

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.

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

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

Error Rate: Overall, how often is it wrong?

```r
classErrorRate <- function(data, actual.idx = 9, predict.idx = 10) {
    TP <- 4
    TN <- 1
    FP <- 3
    FN <- 2

    confusion <- table(data[, actual.idx], data[, predict.idx])
    value <- (confusion[FP] + confusion[FN])/(confusion[TP] + confusion[FP] + confusion[TN] +
        confusion[FN])

    return(value)
}
classErrorRate(class.df)
classErrorRate(class.df) + accuracy(class.df)
```

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
precision <- function(data, actual.idx = 9, predict.idx = 10) {
    TP <- 4
    TN <- 1
    FP <- 3
    FN <- 2

    confusion <- table(data[, actual.idx], data[, predict.idx])
    value <- confusion[TP]/(confusion[TP] + confusion[FP])

    return(value)
}

precision(class.df)
```

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 <- function(data, actual.idx = 9, predict.idx = 10) {
    TP <- 4
    TN <- 1
    FP <- 3
    FN <- 2

    confusion <- table(data[, actual.idx], data[, predict.idx])
    value <- confusion[TP]/(confusion[TP] + confusion[FN])

    return(value)
}

sensitivity(class.df)
```

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 <- function(data, actual.idx = 9, predict.idx = 10) {
    TP <- 4
    TN <- 1
    FP <- 3
    FN <- 2

    confusion <- table(data[, actual.idx], data[, predict.idx])
    value <- confusion[TN]/(confusion[TN] + confusion[FP])

    return(value)
}

specificity(class.df)
```

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.

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

```
f1Score <- function(data, actual.idx = 9, predict.idx = 10) {
    confusion <- table(data[, actual.idx], data[, predict.idx])

    value <- (2 * precision(data) * sensitivity(data))/(precision(data) + sensitivity(data))

    return(value)
}

f1Score(class.df)
```

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.)

F1 score reaches its best value at 1 and worst at 0. since both presition and sensitivity are in between 0 and 1 we know 0 < p_s <=1 and 2ps <=p +s (maximum value p and s can have is 1) so 2_p*s/(p +s) can have values in between 0 and 1

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.

```
getRoCData <- function(data) {
    rocResults <- data.frame(Threshold = c(), Sn = c(), Sp = c())

    thresholds <- seq(0, 1, by = 0.01)
    for (threshold in thresholds) {
        data <- data %>% mutate(predicted = as.numeric(scored.probability >= threshold))
        data$predicted <- factor(data$predicted, levels = c(0, 1))

        Sn <- sensitivity(data, actual.idx = 9, predict.idx = 12)
        Sp <- specificity(data, actual.idx = 9, predict.idx = 12)

        rocResults <- rbind(rocResults, data.frame(Threshold = c(threshold), Sn = c(Sn),
            Sp = c(Sp)))
    }

    auc <- cumsum(rocResults$Sn - (1 - rocResults$Sp)) * 0.01

    rocPlot <- ggplot(rocResults, aes(x = 1 - Sp, y = Sn)) + geom_line(size = 1,
        alpha = 0.6) + geom_point(data = rocResults[rocResults$Threshold == 0.5,
        ], aes(x = 1 - Sp, y = Sn), colour = "orange") + geom_abline(slope = 1, intercept = 0,
        colour = "blue", size = 0.5) + xlim(0, 1) + ylim(0, 1) + labs(title = "ROC Curve",
        x = "False Positive Rate ( 1 - Sp) ", y = "True Positive Rate (Sn)")

    results <- list(plot = rocPlot, auc = auc)

    return(results)
}

result <- getRoCData(class.df)
result$plot
result$auc
```

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

```
accuracy <- accuracy(class.df)
Sn <- sensitivity(class.df)
Sp <- specificity(class.df)
ClsErrRate <- classErrorRate(class.df)
f1 <- f1Score(class.df)
precsn <- precision(class.df)

metrics <- data.frame(Metric = c(), Value = c())
metrics <- rbind(metrics, data.frame(Metric = "Accuracy", Value = round(accuracy,
    4)))
metrics <- rbind(metrics, data.frame(Metric = "Sn(True Positive Rate)", Value = round(Sn,
    4)))
metrics <- rbind(metrics, data.frame(Metric = "Sp(True Negative Rate)", Value = round(Sp,
    4)))
metrics <- rbind(metrics, data.frame(Metric = "ClassErrorRate", Value = round(ClsErrRate,
    4)))
metrics <- rbind(metrics, data.frame(Metric = "F1Score", Value = round(f1, 4)))
metrics <- rbind(metrics, data.frame(Metric = "Precision", Value = round(precsn,
    4)))

kable(metrics)
```

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?

Confusion Matrix

```
confusionMatrix(class.df$scored.class, class.df$class, positive = "1")
```

Sensitivity

```
round(caret::sensitivity(factor(class.df$scored.class), factor(class.df$class), positive = "1"),
    4)
```

Specificity

```
round(caret::specificity(factor(class.df$scored.class), factor(class.df$class), negative = "0"),
    4)
```

We compared the caret confusionmatrix with the one we created; all the measures are matched with the manually created one. Specificity and sensitivity are also same as we calculated.

lift

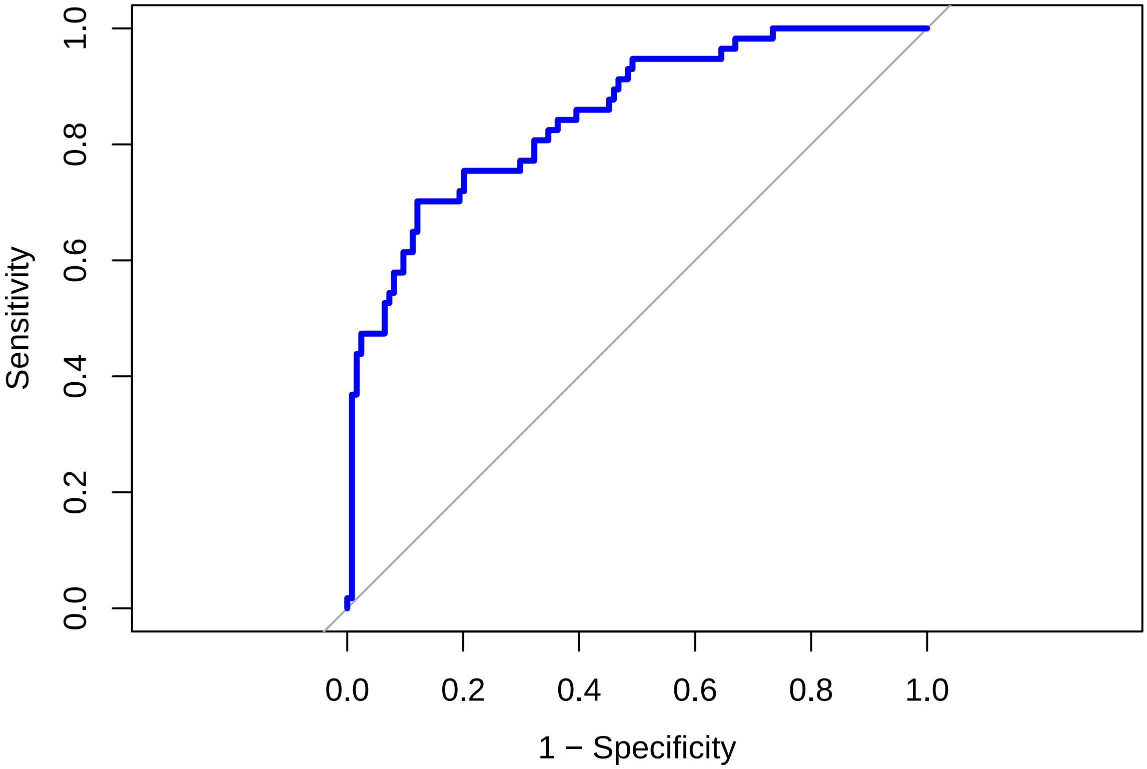
```
(liftResult <- lift(data = class.df, factor(class) ~ scored.probability))
graphics::plot(liftResult, plot = "gain", title(main = "Sensitivity Vs Support"))
graphics::plot(liftResult, plot = "lift")
```

13. Investate the pROC package. Use it to generate an ROC curve for the data set. How do the results compare with your own functions?

```
(roc <- roc(factor(class) ~ scored.probability, data = , class.df, plot = FALSE,
    ci = TRUE))
```

```
##
## Call:
## roc.formula(formula = factor(class) ~ scored.probability, data = class.df,     plot = FALSE, ci = TRU
##
## Data: scored.probability in 124 controls (factor(class) 0) < 57 cases (factor(class) 1).
## Area under the curve: 0.8503
## 95% CI: 0.7905-0.9101 (DeLong)
```

```
graphics::plot(roc, legacy.axes = TRUE, col = "blue", lwd = 3)
```



```
##
## Call:
## roc.formula(formula = factor(class) ~ scored.probability, data = class.df,     plot = FALSE, ci = TRU
##
## Data: scored.probability in 124 controls (factor(class) 0) < 57 cases (factor(class) 1).
## Area under the curve: 0.8503
## 95% CI: 0.7905-0.9101 (DeLong)
```

```
auc(factor(class) ~ scored.probability, class.df)
```

```
## Area under the curve: 0.8503
```

The RoC curve is very similar to our RoC plot drawn using the getRoCData() function above.