Data621: Homework#2

Group # 2

3/5/2018

Introduction

This assignment examines classification matrix using functions created with R to calculate Accuracy, Classification Error Rate, Precision, Sensitivity (also known as recall), Specificity and F1 score. It also introduces the R libraries caret and pROC that have similar functions and calls for a comparison between our own functions and the built-in functions from the packages.

Contributors

Sharon Morris
Brian Kreis
Keith Folsom
Michael D'acampora
Valerie Briot

Load the data

A data-set that contains 181 observations about patients. We are assuming this data set was part of the "pima Indians diabetes" data-set hosted at UCI Machine Learning repository. The objective of the data-set is probably to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the data-set.

The National Institute of Diabetes and Digestive and Kidney Diseases conducted a study on 768 adult female Pima Indians living near Phoenix. The following variables were recorded:

– number of times pregnant,
– plasma glucose concentration at 2 hours in an oral glucose tolerance test,
– diastolic blood pressure (mmHg),
– triceps skin fold thickness (mm),
– 2-hour serum insulin (mu U/ml),
– body mass index (weight in kg/(height in m2)),
– diabetes pedigree function, age (years) and
– a test whether the patient showed signs of diabetes (coded zero if negative, one if positive), (assumed to be class in our data set).

However, based on the goals of this project the meaning of the data-set is not relevant. This project examines the measures for the binary classifier,
1 = positive outcome, and the value of 0 = negative outcome.

```r
classData <- read.csv("https://raw.githubusercontent.com/indianspice/DATA621/master/HW2/classification-

summary(classData)
```

```
##     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
```

We are interested in building a confusion matrix from columns; class and scored.class and determining the Statistical Performance Measures for the classifier.

Confusion Matrix and Measures

1. Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand. it includes:

-True positive (TP): correct positive prediction
-False positive (FP): incorrect positive prediction
-True negative (TN): correct negative prediction
-False negative (FN): incorrect negative prediction

```r
#2
f_cm <- function (df, actual, predicted){

  v_actual <- df[[actual]]
  v_predicted <- df[[predicted]]

  cm <- as.matrix(table(Actual = v_actual, Predicted = v_predicted))
  cm <- t(cm)

  return(cm)
}

#f_cm(classData, "class", "scored.class")
```

---

1. Statistical Performance Measures

a. Accuracy

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.

Formula for Accuracy is given as follows:

$$Accuracy \ / \ \frac{TP \rightharpoondown TN}{TP \rightharpoondown FP \rightharpoondown TN \rightharpoondown FN}$$

```r
#3

f_accuracy <- function(df, actual, predicted){
 tab <- table(df[[actual]], df[[predicted]])
 tp <- tab[2,2]
 tn <- tab[1,1]
 fn <- tab[2,1]
 fp <- tab[1,2]

 accuracy = (tp + tn) / (tp + fp + tn + fn)

 return(as.numeric(accuracy))
}

#f_accuracy(classData, "class", "scored.class")
```

b. Classification Error Rate

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.

Formula for Classification Error Rate is given as follows:

$$ClassificationErrorRate \ / \ \frac{FP \rightharpoondown FN}{TP \rightharpoondown FP \rightharpoondown TN \rightharpoondown FN}$$

```r
#4
f_CER <- function(df, actual, predicted){
 tab <- table(df[[actual]], df[[predicted]])
 tp <- tab[2,2]
 tn <- tab[1,1]
 fn <- tab[2,1]
 fp <- tab[1,2]

 error = (fp + fn) / (tp + fp + tn + fn)

 return(as.numeric(error))
}

#f_CER(classData, "class", "scored.class")
```

We will now show that:

$$Accuracy \quad \rightharpoondown \quad Classification \quad Error \quad Rate \quad / \quad 1$$

$$Accuracy \rightarrow ClassificationErrorRate / \frac{TP \rightarrow TN}{TP \rightarrow FP \rightarrow TN \rightarrow FN} \rightarrow \frac{FP \rightarrow FN}{TP \rightarrow FP \rightarrow TN \rightarrow FN}$$

$$Accuracy \rightarrow ClassificationErrorRate / \frac{TP \rightarrow TN \rightarrow FP \rightarrow FN}{TP \rightarrow FP \rightarrow TN \rightarrow FN}$$

$$/ \frac{TP \rightarrow FP \rightarrow TN \rightarrow FN}{TP \rightarrow FP \rightarrow TN \rightarrow FN} / 1$$

c. Precision

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.

Formula for Precision is given as follows:

$$Precision / \frac{TP}{TP \rightarrow FP}$$

```r
#5
f_precision <- function(df, actual, predicted) {
  #Enter dataframe, followed by actual and predicted column names in quotes
  tab <- table(df[[actual]], df[[predicted]])
  tp <- tab[2,2]
  fp <- tab[1,2]

  precision <- tp/(tp+fp)

  return(precision)
}

#f_precision(classData, 'class', 'scored.class')
```

d. Sensitivity

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.

Formula for Sensitivity is given as follows:

$$Sensitivity / \frac{TP}{TP \rightarrow FN}$$

```r
#6
f_sensitivity <- function(df, actual, predicted) {
  #Enter dataframe, followed by actual and predicted column names in quotes
  tab <- table(df[[actual]], df[[predicted]])
  tp <- tab[2,2]
  fn <- tab[2,1]

  sensitivity <- tp/(tp+fn)

  return(sensitivity)
}
```

```
#f_sensitivity(classData, 'class', 'scored.class')
```

e. Specificity

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.

Formula for Specificity is given as follows:

$$Specificity \ / \ \frac{TN}{TN \rightharpoondown FP}$$

```
#7

f_specificity = function(df, actual, predicted) {
  tab <- table(df[[actual]], df[[predicted]])
  tn <- tab[1,1]
  fp <- tab[1,2]

  specificity <- tn / (tn + fp)

  return(specificity)
}
#f_specificity(classData, "class", "scored.class")
```

f. F1_Score

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 is the harmonic mean of Precision and Sensitivity. Hence it can be written as:

$$F1\_Score \quad / \quad \left(\frac{Precision^{-1} \rightharpoondown Sensitivity^{-1}}{2}\right)^{-1}$$

This can be further written as:

$$F1\_Score \quad / \quad \frac{2}{\frac{1}{Precision} \rightharpoondown \frac{1}{Sensitivity}}$$

Finally, we have:

$$F1\_Score \quad / \quad \frac{2}{\frac{1}{Precision} \rightharpoondown \frac{1}{Sensitivity}} \ / \ \frac{2}{\frac{Sensitivity \quad \rightharpoondown \quad Precision}{Precision \times Sensitivity}} \ / \ 2 \cdot \frac{Precision \times Sensitivity}{Sensitivity \quad \rightharpoondown \quad Precision}$$

```
#8
f_F1_score = function(df, actual, predicted) {

  precision <- f_precision(df, actual, predicted)
  sensitivity <- f_sensitivity(df, actual, predicted)

  f1_score <- (2 * precision * sensitivity) / (precision + sensitivity)
```

```r
  return(f1_score)
}

#f_F1_score(classData, "class", "scored.class")
```

---

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).

Taking the equation above for F1_Score, we will substitute the formula for Precision and Sensitivity respectively.

$$F1\_Score = 2 \cdot \frac{\frac{TP}{TP+FP} \cdot \frac{TP}{TP+FN}}{\frac{TP}{TP+FN} + \frac{TP}{TP+FP}} = 2 \cdot \frac{\frac{(TP)^2}{(TP+FP)\cdot(TP+FN)}}{\frac{TP\cdot(TP+FP)+TP\cdot(TP+FN)}{(TP+FN)\cdot(TP+FP)}}$$

$$F1\_Score = 2 \cdot \frac{(TP)^2}{(TP+FP)\cdot(TP+FN)} \cdot \frac{(TP+FN)\cdot(TP+FP)}{TP\cdot(TP+FP+TP+FN)}$$

This can be reduced to:

$$F1\_Score = 2 \cdot \frac{(TP)^2}{TP(TP+FP+TP+FN)} = \frac{2\cdot TP}{2\cdot TP+FP+FN}$$

The values of TP, FP, and FN are >=0.

Let us assume the extreme case where TP = 0 (we did not identify any true positive), hence F1_score = 0

If TP > 0, then 0 < 2TP <= 2TP + (FP+FN),

if FP+FN=0, meaning that we did not commit any errors and False Positive and False Negative are both zero, then we will have F1_score = 1.

Barring these 2 extreme cases, it is clear that F1_score will be between 0 and 1 since we will have:

0 < 2TP < 2TP + (FP+FN) (multiplying on both side by reciprocal of (2TP+FP+FN)), which is a positive number will not change sense of inequality).

0 < F1_Score < 1

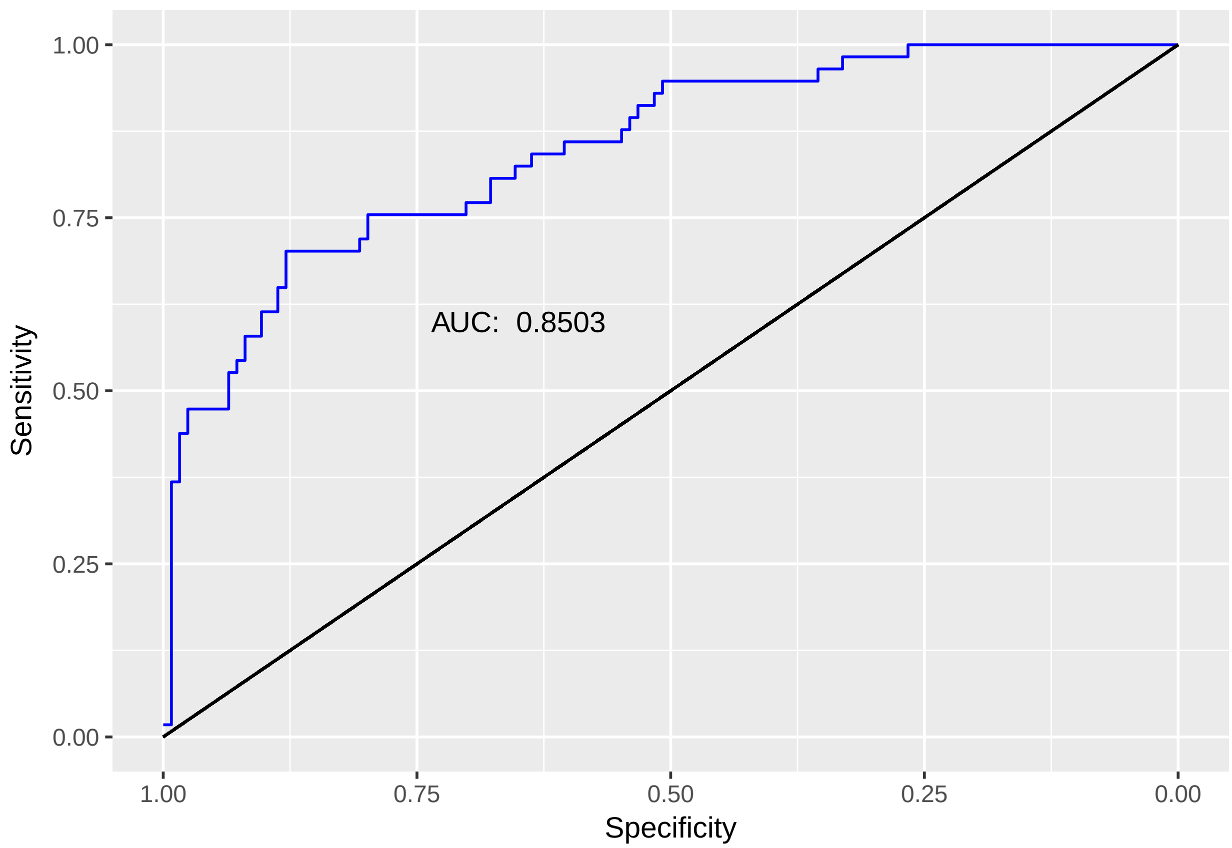
---

g. ROC Curve

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.

```r
#10

f_roc <- function(labels, scores) {

  require(ggplot2)

  # References:
  # http://blog.revolutionanalytics.com/2016/08/roc-curves-in-two-lines-of-code.html
```

```r
  # http://blog.revolutionanalytics.com/2016/11/calculating-auc.html

  # sort the classification data with the highest probability scores desc
  labels <- labels[order(scores, decreasing=TRUE)]

  # TPR: cumulative True Positive Rate divided by the total number of actual positives
  # FPR: False Positive Rate
  #      cumulative umber of false positives divided by total number of true negative
  df <- data.frame(TPR=cumsum(labels)/sum(labels),
                   FPR=cumsum(!labels)/sum(!labels),
                   label=labels)

  # add specificity
  df$specificity <- 1 - df$FPR
  # calculate distance of FPR and TPR from 0
  df$dFPR <- c(diff(df$FPR), 0)
  df$dTPR <- c(diff(df$TPR), 0)

  # calculate AUC
  AUC <- round(sum(df$TPR * df$dFPR) + sum(df$dTPR * df$dFPR)/2, 4)

  roc_plot <- ggplot(df, aes(x=specificity, y=TPR)) +
              xlim(1, 0) + ylim(0,1) +
              xlab("Specificity") + ylab("Sensitivity") +
              geom_line(color='blue') +
              geom_segment(aes(x = 1, y = 0, xend = 0, yend = 1) ) +
              annotate("text", x=.65, y = .60, label=paste("AUC: ", AUC))


  return (list(roc_plot, AUC))

}

roc <- f_roc(classData$class, classData$scored.probability)

roc[[1]]  # show the ROC curve plot
```




```
roc[[2]]  # show the AUC value
```

```
## [1] 0.8503
```

---

3. Caret Package vs Our own Calculations

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

We will first create a function to call and format the results.

```
#11

f_perform_measure <- function(df, actual, predicted, n=4, pr='Y'){

  # Build Confusion Matrix #
  pm_cm <-f_cm(df, actual, predicted)

  # Calculate Accuracy #
  pm_accuracy <- f_accuracy(df, actual, predicted)

  # Calcualte Classification Error Rate
  pm_error <- f_CER(df, actual, predicted)

  # Calculate Precision
```

```r
  pm_precision <- f_precision(df, actual, predicted)

  # Calculate Sensitivity
  pm_sensitivity <- f_sensitivity(df, actual, predicted)

  # Calculate Specificity
  pm_specificity <- f_specificity(df, actual, predicted)

  # Calculate F1_Score
  pm_F1_Score <- f_F1_score(df, actual, predicted)

  # If rounding required
  if (n>=0){

    pm_accuracy <- round(pm_accuracy,n)
    pm_error <- round(pm_error, n)
    pm_precision <- round(pm_precision, n)
    pm_sensitivity <- round(pm_sensitivity, n)
    pm_specificity <- round(pm_specificity, n)
    pm_F1_Score <- round(pm_F1_Score, n)
  }

  # If Output of information required
  if (pr == 'Y'){

  cat("The confusion matrix and statistics :\n\n")

  print(pm_cm)

  cat("\nAccuracy                  :   ", pm_accuracy)
  cat("\n\nClassification Error Rate :   ", pm_error)
  cat("\n\nPrecision                 :   ", pm_precision)
  cat("\n\nSensitifity               :   ", pm_sensitivity)
  cat("\n\nSpecificity               :   ", pm_specificity)
  cat("\n\nF1_Score                  :   ", pm_F1_Score)
  cat("\n\nSanity Check")
  cat("\nAccuracy + Classification Error Rate is ", pm_accuracy+pm_error)

  }

  return(list(pm_cm, pm_accuracy, pm_error, pm_precision, pm_sensitivity, pm_specificity, pm_F1_Score))
}

our_results <- f_perform_measure(classData, "class", "scored.class", n=4)
```

```
## The confusion matrix and statistics :
##
##          Actual
## Predicted   0   1
##         0 119  30
##         1   5  27
##
## Accuracy                  :    0.8066
##
```

```
## Classification Error Rate :    0.1934
##
## Precision                 :    0.8438
##
## Sensitifity               :    0.4737
##
## Specificity               :    0.9597
##
## F1_Score                  :    0.6067
##
## Sanity Check
## Accuracy + Classification Error Rate is  1
```

---

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?

The caret package (short for __C__lassification __A__nd __RE__gression __T__raining) is a set of functions that attempt to streamline the process for creating predictive models.

The package contains tools for:
– data splitting
– pre-processing
– feature selection
– model tuning using resampling
– variable importance estimation

as well as other functionality. One such functionality is Measures Performance.

The caret package has functionality for:
– Measures for Regression
– Measures for Predicted Classes
– Measures for Class Probabilities
– Lift Curves
– Calibration Curves

We will now examine the Measures for Predicted Classess.

we can use the function confusionMatrix, which shows a cross-tabulation of the observed and predicted classes and the key statistics.

From the documentation of the confusion matrix, the following statistics are computed:

`Reference`

Predicted	Event	No Event
Event	A	B
No Event	C	D

$Sensitivity\ /\ \frac{A}{A \rightharpoondown C}$
$Specificity\ /\ \frac{D}{B \rightharpoondown D}$
$Prevalence\ /\ \frac{\leftharpoonup A \rightharpoondown C \leftharpoondown}{\leftharpoonup A \rightharpoondown B \rightharpoondown C \rightharpoondown D \leftharpoondown}$
$PPV\ /\ \frac{\leftharpoonup sensitivity \times prevalence \leftharpoondown}{\leftharpoonup \leftharpoonup sensitivity \times prevalence \leftharpoondown \rightharpoondown \leftharpoonup \leftharpoonup 1 - specificity \leftharpoondown \times \leftharpoonup 1 - prevalence \leftharpoondown \leftharpoondown \leftharpoondown}$
$NPV\ /\ \frac{\leftharpoonup specificity \times \leftharpoonup 1 - prevalence \leftharpoondown \leftharpoondown}{\leftharpoonup \leftharpoonup \leftharpoonup 1 - sensitivity \leftharpoondown \times prevalence \leftharpoondown \rightharpoondown \leftharpoonup \leftharpoonup specificity \leftharpoondown \times \leftharpoonup 1 - prevalence \leftharpoondown \leftharpoondown \leftharpoondown}$
$DetectionRate\ /\ \frac{A}{\leftharpoonup A \rightharpoondown B \rightharpoondown C \rightharpoondown D \leftharpoondown}$
$DetectionPrevalence\ /\ \frac{\leftharpoonup A \rightharpoondown B \leftharpoondown}{\leftharpoonup A \rightharpoondown B \rightharpoondown C \rightharpoondown D \leftharpoondown}$
$BalancedAccuracy\ /\ \frac{\leftharpoonup sensitivity \rightharpoondown specificity \leftharpoondown}{2}$ $Precision\ /\ \frac{A}{\leftharpoonup A \rightharpoondown B \leftharpoondown}$ $Recall\ /\ \frac{A}{\leftharpoonup A \rightharpoondown C \leftharpoondown}$
$F1\_Score\quad /\quad \frac{\leftharpoonup 1 \rightharpoondown \beta^2 \leftharpoondown \times precision \times recall}{\leftharpoonup \beta^2 \times precision \leftharpoondown \rightharpoondown recall}$

We can map the value A, B, C, and D to our own value TP, FP, FN, TN as follows:

- A map to TP,
- B map to FP,
- C map to FN,
- D map to TN

We will compare the following statistics, beyond the actual confusion matrix;
* Accuracy
* Precision
* Sensitivity
* Specificity
* F1_Score

```r
#12

cm_caret <- confusionMatrix(data = classData$scored.class, reference = classData$class, positive = '1')

cm_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
##
```

```
caret_results <- list(cm_caret$table, cm_caret$overall[[1]], cm_caret$byClass[[5]], cm_caret$byClass[[1
```

We will now compare our results with the results from the caret package. To do so, we will build a comparison dataframe.

```
# Compare Confusion Matrices
our_results[[1]] == caret_results[[1]]
```

```
##          Actual
## Predicted    0    1
##         0 TRUE TRUE
##         1 TRUE TRUE
```

```
# Build Comparaison Data Frame
v_our_results <- as.vector(unlist(our_results[-c(1,3)]))
v_caret_results <- as.vector(unlist(caret_results[-1]))
v_measures <- c("Accuracy", "Precision", "Sensitivity", "Specificity", "F1_Score")

df_compare <- as.data.frame(cbind(v_measures, v_our_results, v_caret_results))

# Update Column Names
colnames(df_compare) <- c("Measures", "Our_Results", "Caret_Results")

# Force Numeric, convert from factor
df_compare$Our_Results <- as.numeric(as.character(df_compare$Our_Results))
df_compare$Caret_Results <- as.numeric(as.character(df_compare$Caret_Results))

df_compare$Difference <- df_compare$Our_Results - df_compare$Caret_Results

knitr::kable(df_compare)
```

Measures	Our_Results	Caret_Results	Difference
Accuracy	0.8066	0.8066298	-2.98e-05
Precision	0.8438	0.8437500	5.00e-05
Sensitivity	0.4737	0.4736842	1.58e-05
Specificity	0.9597	0.9596774	2.26e-05
F1_Score	0.6067	0.6067416	-4.16e-05

We obtain the same confusion matrix as is indicated by the comparison. The differences are due to the rounding that we selected when calculating our values and they are negligible. We will recalculate without rounding by setting rounding parameter to "none" by passing a (-1) and rebuild the comparison matrix (we will also select not to display our results by setting print parameter to 'N').

```r
our_results <- f_perform_measure(classData, "class", "scored.class", n=-1, pr='N')

# Compare Confusion Matrices
our_results[[1]] == caret_results[[1]]
```

```
##          Actual
## Predicted    0    1
##         0 TRUE TRUE
##         1 TRUE TRUE
```

```r
# Build Comparaison Data Frame
v_our_results <- as.vector(unlist(our_results[-c(1,3)]))
v_caret_results <- as.vector(unlist(caret_results[-1]))
v_measures <- c("Accuracy", "Precision", "Sensitivity", "Specificity", "F1_Score")

df_compare <- as.data.frame(cbind(v_measures, v_our_results, v_caret_results))

# Update Column Names
colnames(df_compare) <- c("Measures", "Our_Results", "Caret_Results")

# Force Numeric, convert from factor
df_compare$Our_Results <- as.numeric(as.character(df_compare$Our_Results))
df_compare$Caret_Results <- as.numeric(as.character(df_compare$Caret_Results))

df_compare$Difference <- df_compare$Our_Results - df_compare$Caret_Results

knitr::kable(df_compare)
```

Measures	Our_Results	Caret_Results	Difference
Accuracy	0.8066298	0.8066298	0
Precision	0.8437500	0.8437500	0
Sensitivity	0.4736842	0.4736842	0
Specificity	0.9596774	0.9596774	0
F1_Score	0.6067416	0.6067416	0

We obtain the same results for the statistical measures we compared: Accuracy, Precision, Sensitivity, Specificity, and F1_score as well as the confusion matrix.

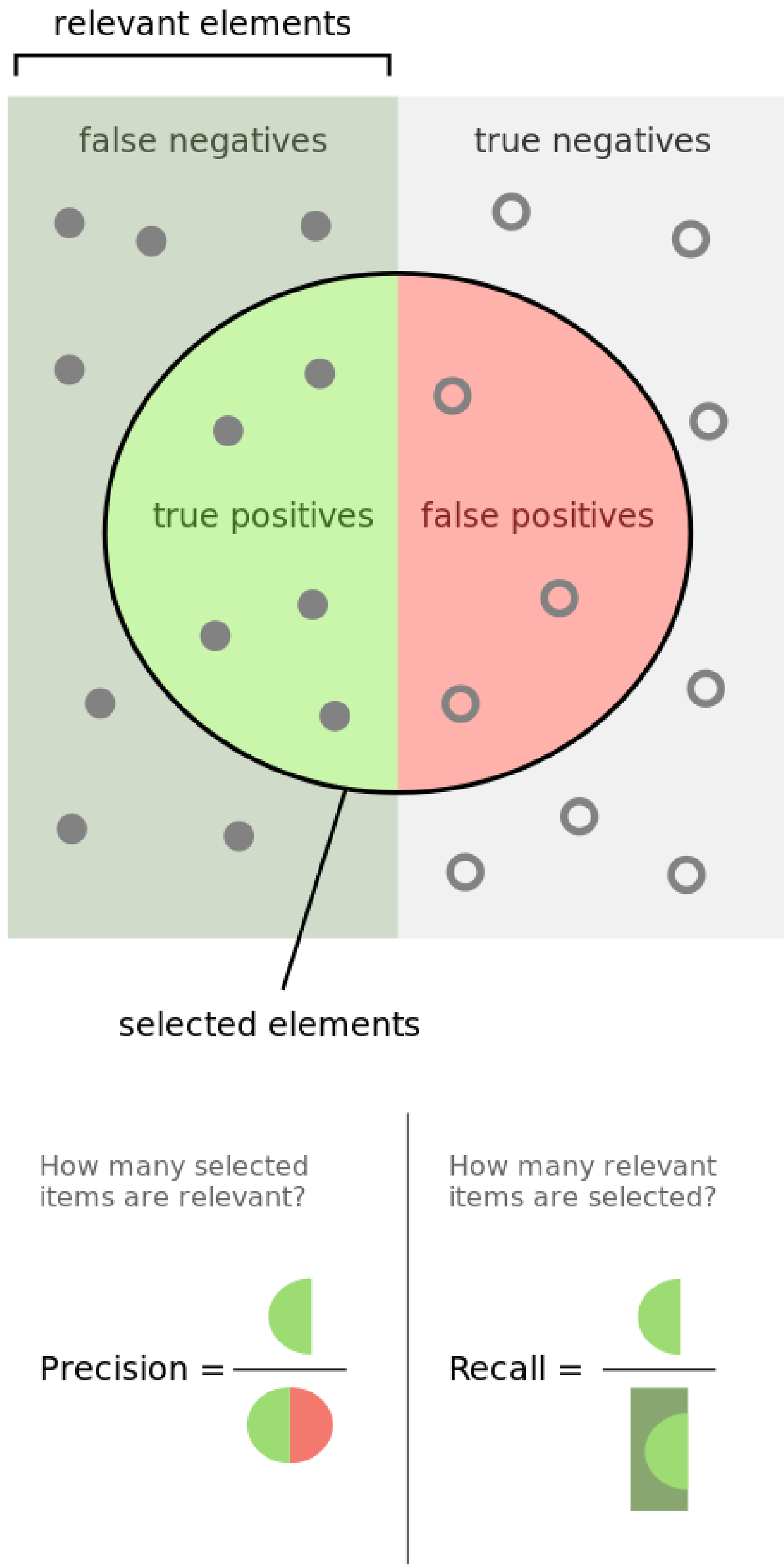


Figure 1: Classifyier Statitistical Measures

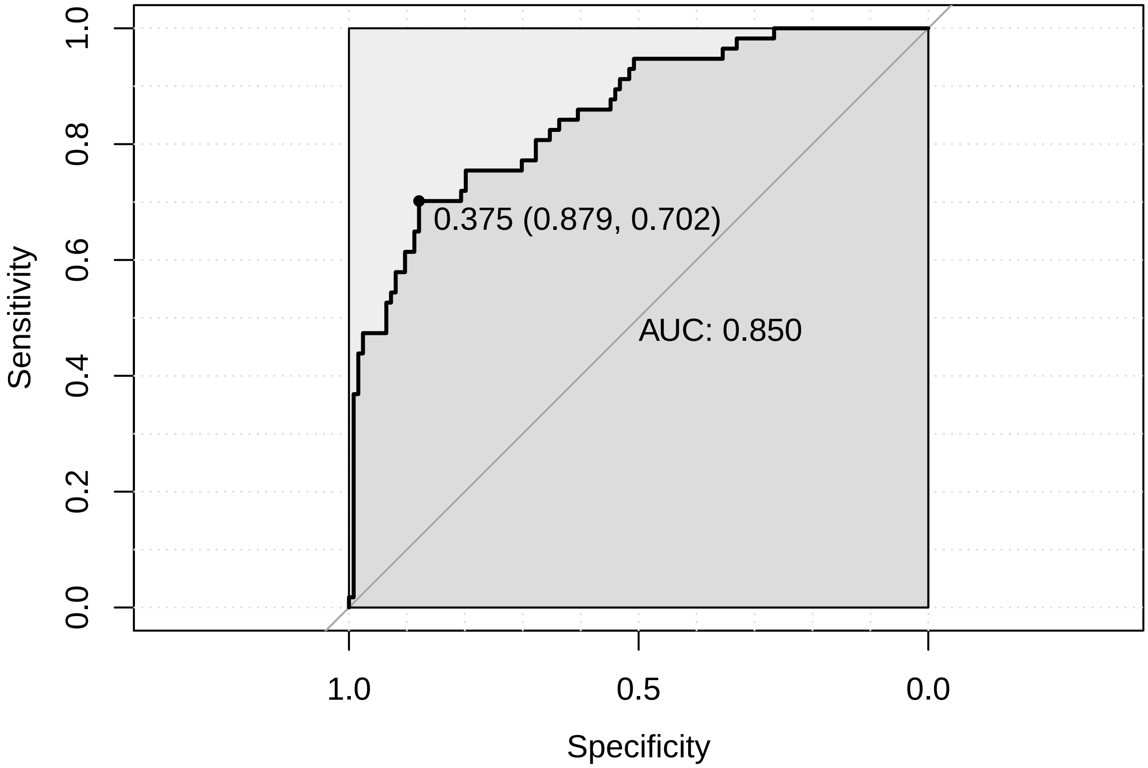
4. pROC Package vs Our own ROC Curve

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?

```
#13

require(pROC)
```

```
## Loading required package: pROC

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

##
## Attaching package: 'pROC'

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

```
# calculate and print the ROC curve with AUC

roc2 <- pROC::roc(classData$class,
                  classData$scored.probability,
                  print.thres=TRUE,
                  percent=F, plot=TRUE,
                  auc.polygon=TRUE,
                  max.auc.polygon=TRUE,
                  grid=TRUE,ci=F,
                  print.auc=TRUE)
```




```r
print(roc2)
```

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

```r
# calculate AUC directly using pROC

auc(classData$class, classData$scored.probability)
```

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

The ROC curve resulting from the function used in problem 10 looks similar to the one produced by the pROC function. The AUC calculations appear to be very similar with both showing 0.8503. However, after exploring the pROC package, this package offers far more built-in features around calculating, printing, testing, and comparing ROC curves.

References

http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/
https://en.wikipedia.org/wiki/F1_score
http://topepo.github.io/caret/index.html
https://www.rdocumentation.org/packages/caret/versions/6.0-78/topics/confusionMatrix
http://blog.revolutionanalytics.com/2016/08/roc-curves-in-two-lines-of-code.html
http://blog.revolutionanalytics.com/2016/11/calculating-auc.html