HW2_Andrew_Goldberg

Andrew Goldberg

3/11/2017

```
classData <- read.csv("https://raw.githubusercontent.com/aagoldberg/Data-621-Regression/master/classifi
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
```

```
length(classData$pregnant)
```

```
## [1] 181
```

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(pROC)
```

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

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
```

```
##
##     cov, smooth, var
```

Use table() function to get the raw confusion matrix

```
useData <- classData[9:11]

table(useData[2:1])
```

```
##              class
## scored.class   0   1
##            0 119  30
##            1   5  27
```

```
#rows = actual class, columns = predicted class
```

Write a function that takes the data set as a dataframe, with actual and predicted classifications identified, and returns the accuracy of the predictions.

```
is.data.frame(useData)
```

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

```
accFunc <- function(classDF){
  truePos <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 1])
  trueNeg <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 0])
  falsePos <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 1])
  falseNeg <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 0])
  accRes <- (truePos+trueNeg) / (truePos + trueNeg + falsePos + falseNeg)
  return(accRes)
}
```

Write a function that returns the classification error rate of the predictions. Verify that accuracy plus error rate sum to 1.

```
classErrFunc <- function(classDF){
  truePos <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 1])
  trueNeg <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 0])
  falsePos <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 1])
  falseNeg <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 0])
  classErrRes <- (falsePos + falseNeg) / (truePos + trueNeg + falsePos + falseNeg)
  return(classErrRes)
}

sum(accFunc(useData), classErrFunc(useData)) == 1
```

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

Write a function that returns the precision of the predictions.

```
precFunc <- function(classDF){
  truePos <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 1])
  trueNeg <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 0])
  falsePos <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 1])
  falseNeg <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 0])
```

```r
  precRes <- (truePos) / (truePos + falsePos)
  return(precRes)
}
```

Write a function that returns the sensitivity of the predictions.

```r
sensFunc <- function(classDF){
  truePos <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 1])
  trueNeg <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 0])
  falsePos <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 1])
  falseNeg <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 0])
  sensRes <- (truePos) / (truePos + falseNeg)
  return(sensRes)
}
```

Write a function that returns the sensitivity of the predictions.

```r
specFunc <- function(classDF){
  truePos <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 1])
  trueNeg <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 0])
  falsePos <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 1])
  falseNeg <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 0])
  specRes <- (trueNeg) / (trueNeg + falsePos)
  return(specRes)
}
```

Write a function that returns the f1 score the predictions.

```r
f1Func <- function(classDF){
  truePos <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 1])
  trueNeg <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 0])
  falsePos <- length(classDF$class[classDF$class == 0 & classDF$scored.class == 1])
  falseNeg <- length(classDF$class[classDF$class == 1 & classDF$scored.class == 0])
  precRes <- (truePos) / (truePos + falsePos)
  sensRes <- (truePos) / (truePos + falseNeg)
  f1Res <- (2*precRes*sensRes) / (precRes + sensRes)
  return(f1Res)
}
```

Show that the F1 score wil always be between 0 and 1.

If I'm understanding it correct, that assumption seems true in practice, but not theoretically, since its possible to have 100% true positives. If that were the case, then both precision and sensitivity would equal 1 and the f1 score would be: ((2 x 1 x 1) / 2) = 1. So 1 appears to be the upper bound. In practice, its highly unlikely to have 100% true positives, so the f1 score would rarely, if ever, reach 1.

Since there are no negative values in the calculations, the lower bound cannot go below 0, and will very likely be higher. However, if there's 100% false positives or 100% false negatives, you could actually get an undefined answer.

Write a function that generates an ROC curve.

It should return a list that includes the plot of the ROC curve and a vector that contains the calculated area under the curve (AUC). Not that I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.

```
rocFunc <- function(classDF){
  truePosRoc <- vector()
  trueNegRoc <- vector()
  falsePosRoc <- vector()
  falseNegRoc <- vector()
  threshProb <- vector()
  thresh <- seq(0, 1, .01)

  for (i in 1:100){
    threshProb[i] <- thresh[i]
    truePosRoc[i] <- sum(classDF$class == 1 & classDF$scored.probability >= thresh[i])
    trueNegRoc[i] <- sum(classDF$class == 0 & classDF$scored.probability < thresh[i])
    falsePosRoc[i] <- sum(classDF$class == 0 & classDF$scored.probability >= thresh[i])
    falseNegRoc[i] <- sum(classDF$class == 1 & classDF$scored.probability < thresh[i])

  }
  sensRoc <- truePosRoc / (truePosRoc + falseNegRoc)
  specRoc <- trueNegRoc / (trueNegRoc + falsePosRoc)
  plotData <- as.data.frame(sensRoc, specRoc)
  g <- ggplot(plotData, aes((1 - specRoc), sensRoc)) + geom_step()
  uac <- sensRoc * specRoc
  return(list(g, sum(uac), uac) )
}
```

Produce all of the previous classification metrics

```
accFunc(useData)
```

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

```
classErrFunc(useData)
```

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

```
precFunc(useData)
```

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

```
sensFunc(useData)
```

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

```
specFunc(useData)
```

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

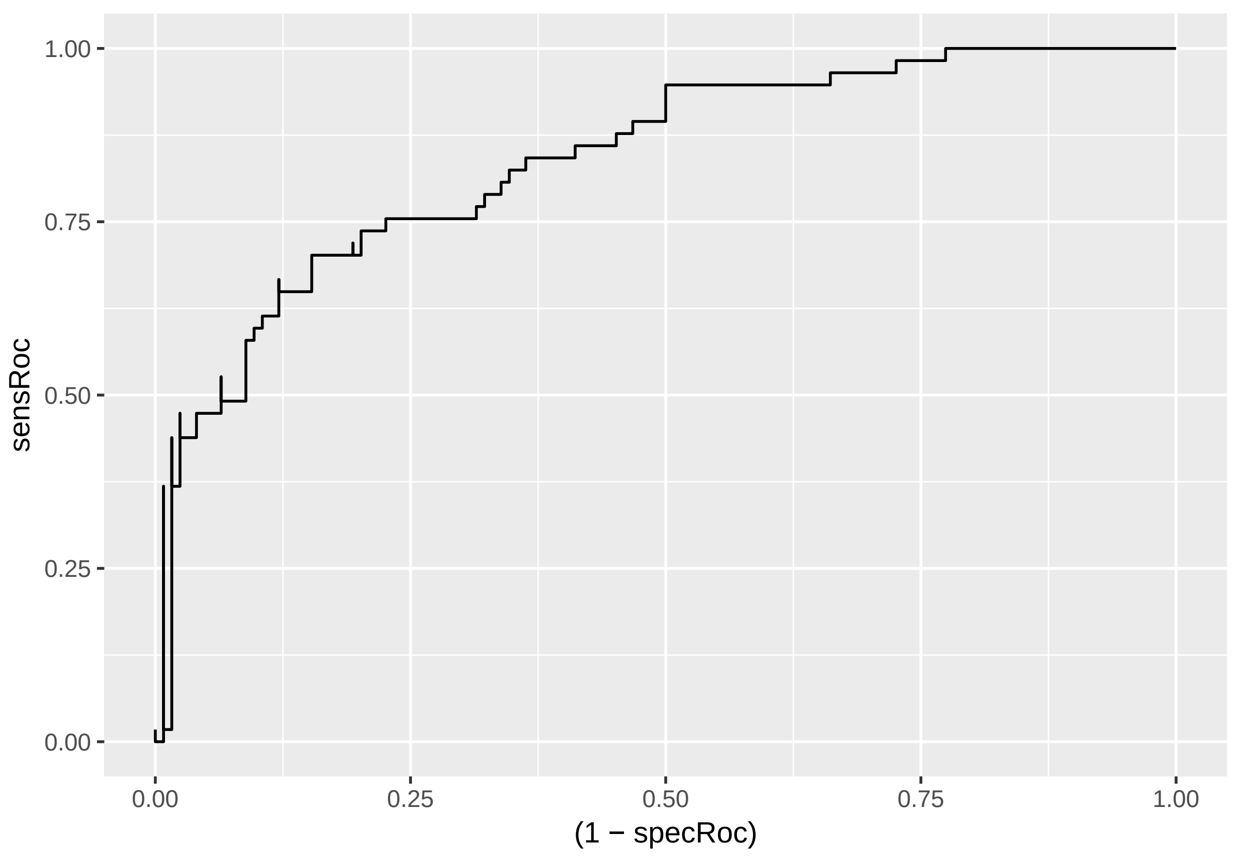
```
f1Func(useData)
```

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

```
rocFunc(useData)
```

```
## Warning in as.data.frame.numeric(sensRoc, specRoc): 'row.names' is not a
## character vector of length 100 -- omitting it. Will be an error!
```

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



```
##
## [[2]]
## [1] 32.05362
##
## [[3]]
##   [1] 0.00000000 0.00000000 0.00000000 0.02419355 0.03225806 0.04838710
##   [7] 0.09677419 0.15322581 0.17741935 0.22580645 0.26938314 0.32682513
##  [13] 0.34380306 0.37436333 0.42784380 0.45076401 0.47368421 0.47623090
##  [19] 0.48104131 0.49915110 0.50608376 0.51612903 0.52971138 0.53650255
##  [25] 0.53862479 0.53367289 0.53480475 0.52914544 0.52928693 0.54145444
##  [31] 0.56578947 0.58404075 0.58828523 0.58007923 0.56593096 0.56593096
##  [37] 0.58856819 0.59422750 0.58602151 0.58602151 0.57059989 0.54470855
##  [43] 0.54966044 0.53876627 0.52758913 0.52758913 0.49235993 0.47594793
##  [49] 0.45953594 0.44694397 0.45458404 0.45458404 0.45458404 0.46222411
##  [55] 0.44510470 0.42798529 0.42798529 0.43152235 0.43152235 0.43152235
##  [61] 0.41426146 0.41426146 0.36247878 0.36544992 0.33064516 0.33064516
##  [67] 0.33064516 0.31324278 0.29584041 0.26103565 0.24363328 0.22623090
##  [73] 0.20882852 0.19142615 0.19142615 0.19142615 0.19142615 0.17402377
##  [79] 0.17402377 0.15662139 0.15662139 0.15662139 0.13921902 0.13921902
##  [85] 0.12181664 0.10441426 0.08701188 0.06960951 0.06960951 0.01740238
##  [91] 0.01754386 0.01754386 0.01754386 0.01754386 0.01754386 0.00000000
##  [97] 0.00000000 0.00000000 0.00000000 0.00000000
```

Investigate the caret package. In particular, consider the functions confusionMatrix, sensitivity, and specificity. Compare the results.

```r
library(caret)
```

```
## Loading required package: lattice
```

```r
table(useData$class, useData$scored.class)
```

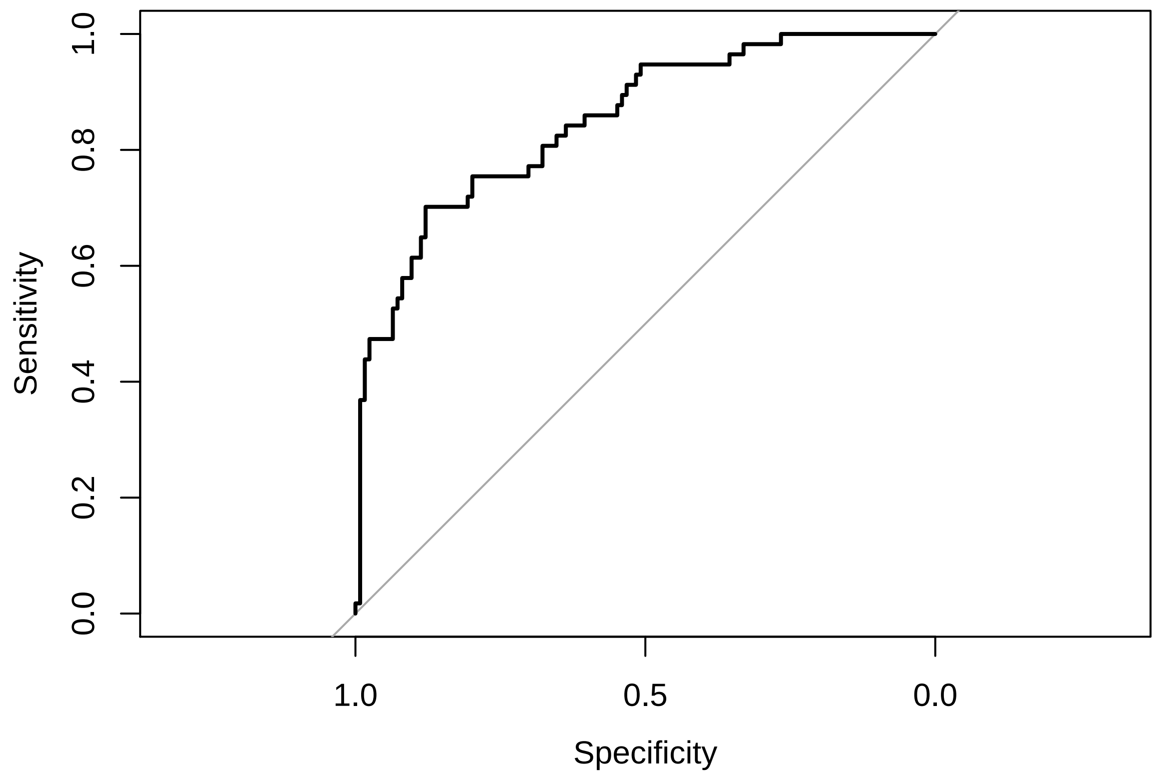
```
##
##       0   1
##   0 119   5
##   1  30  27
```

```r
xtable <- table(useData$scored.class, useData$class)
confusionMatrix(xtable, positive = "1") #same results!
```

```
## Confusion Matrix and Statistics
##
##
##       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
##
```

Investigate the pROC package. Use it to generate a ROC curve and compare.

```r
plot.roc(useData$class, useData$scored.probability)
```



```
#pretty similar !
```