621 HW1

Justin Hsi

October 9, 2020

Setup of environment and data

```
# load required packages and csv
#library(ggplot2)
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------

## v ggplot2 3.2.1     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------------ tid
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(pROC)
```

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

##
## Attaching package: 'pROC'

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

```
library(RCurl)
```

```
##
## Attaching package: 'RCurl'
```

```
## The following object is masked from 'package:tidyr':
##
##     complete
```

```
git_dir = 'https://raw.githubusercontent.com/jmhsi/data621-HW1/master/HW2/classification-output-data.csv
data = read.csv(git_dir, header=T)
# cut data to relevant columns
data = data[,c(9:11)]
```

2) 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?

```
con_mat = table(data$scored.class, data$class)
knitr::kable(con_mat)
```

	0	1
0	119	30
1	5	27

The rows/x-axis are the predicted class and the columns/y-axis are the actual class. Our positive label is 1 and negative label is 0.

3-8 & 11) Write a function that takes the dataset as a dataframe, with actual and predicted classifications identified, and returns the accuracy, error-rate, precision, sensitivity, specificity, and f1 score of the predictions. Verify that you get an accuracy and an error rate that sums to one. Use your created R functions and the provided classifciation output data set to produce all of the classification metrics discussed above.

```
SummarizeBinary = function(t) {
   # calculates TP, FP, FN, TN for use in producing accuracy, error_rate, precision, sensitivity
   # specificity, and f1 score
```

```
   tp = t[2,2]; fp = t[2,1]; fn = t[1,2]; tn = t[1,1]

   accuracy = (tp + tn)/(tp + fp + tn + fn)
   error_rate = (fp + fn)/(tp + fp + tn + fn)
   precision = (tp)/(tp + fp)
   sensitivity = (tp)/(tp + fn)
   specificity = (tn)/(tn + fp)
   f1 = (2 * precision * sensitivity)/(precision + sensitivity)

   df = data.frame(accuracy = accuracy, error_rate = error_rate, precision = precision, sensitivity = se
   return(df)
}

results = SummarizeBinary(con_mat)
print(results$accuracy + results$error_rate)
```

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

The accuracy and error rate do sum to 1.

```
knitr::kable(results)
```

accuracy	error_rate	precision	sensitivity	specificity	f1
0.8066298	0.1933702	0.84375	0.4736842	0.9596774	0.6067416

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 $ab < a$).

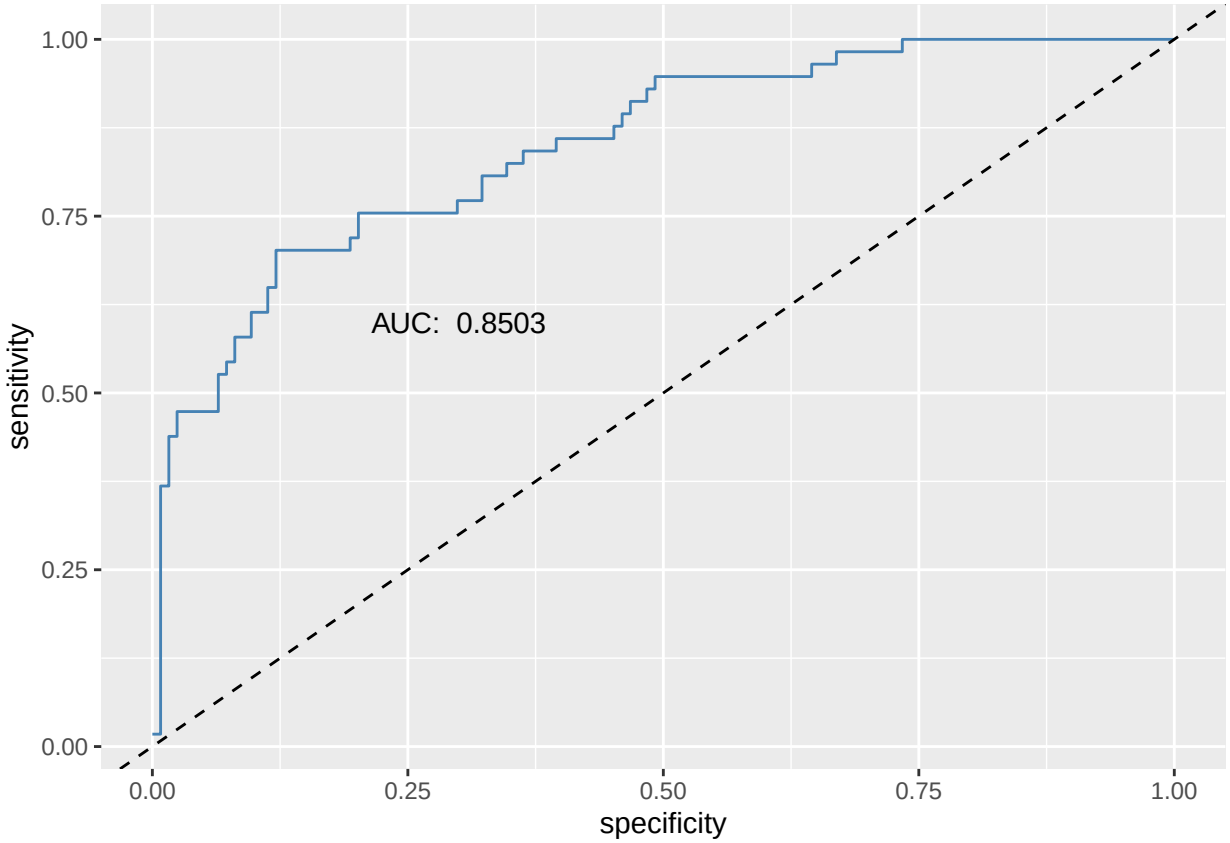
First, tp/fp/tn/fn are in [0,inf). F1-score, only takes precision and sensitivity as inputs, multiplying them in the numerator and adding them in the denominator. Precision and sensitivity are both calculated with denominator larger or equal than numerator using tp/fp/tn/fn, which limits them to [0,1]. Thus we evaluate the F1-score formula at the bounds of precision and sensitivity. If either precision or sensitivity is 0, we get an F1-score of 0. If both happen to be 0, the F1-score will be undefined because the denominator will be 0 for the F1-score. If both precision and sensitivity are 1, the F1-score fill evaluate to 1. So, the F1-score will be in [0,1] and can potentially be undefined.

10) Write a function that generates an ROC curve from a dataset 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). Notethat I recommend using a sequence of thresholds ranging from 0 to 1 at 0.01 intervals.

```
ROCAUC = function(class, scores) {
   class = class[order(scores, decreasing=TRUE)]
   sensitivity = cumsum(class)/sum(class)
   specificity = cumsum(!class)/sum(!class)
   df = data.frame(sensitivity, specificity, class)
   dspec = c(diff(specificity), 0)
   dsens = c(diff(sensitivity), 0)
   AUC = round(sum(sensitivity*dspec) + sum(dsens*dspec)/2, 4)
   results = list(df,AUC)
   return(results)
}

ROCAUC_res = ROCAUC(data$class, data$scored.probability)
ROC_res = ROCAUC_res[[1]]
AUC = ROCAUC_res[[2]]

ggplot(ROC_res, aes(specificity, sensitivity)) +
   geom_line(color='steelblue') + geom_abline(linetype=2) +
   annotate("text", x=.3, y=.6, label=paste("AUC: ", AUC))
```



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?

```
cm = confusionMatrix(as_factor(data$scored.class), as_factor(data$class), positive = "1")
cm
```

```
## 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
##
```

```
print(paste('Sensitivity: ', sensitivity(as_factor(data$scored.class), as_factor(data$class), positive =
```

```
## [1] "Sensitivity:  0.473684210526316"
```

```
print(paste('Specificity: ', specificity(as_factor(data$scored.class), as_factor(data$class), positive =
```

```
## [1] "Specificity:  0.959677419354839"
```

The results are the same, making sure that we correctly tell caret that "1" is the positive label and "0" is the negative label.

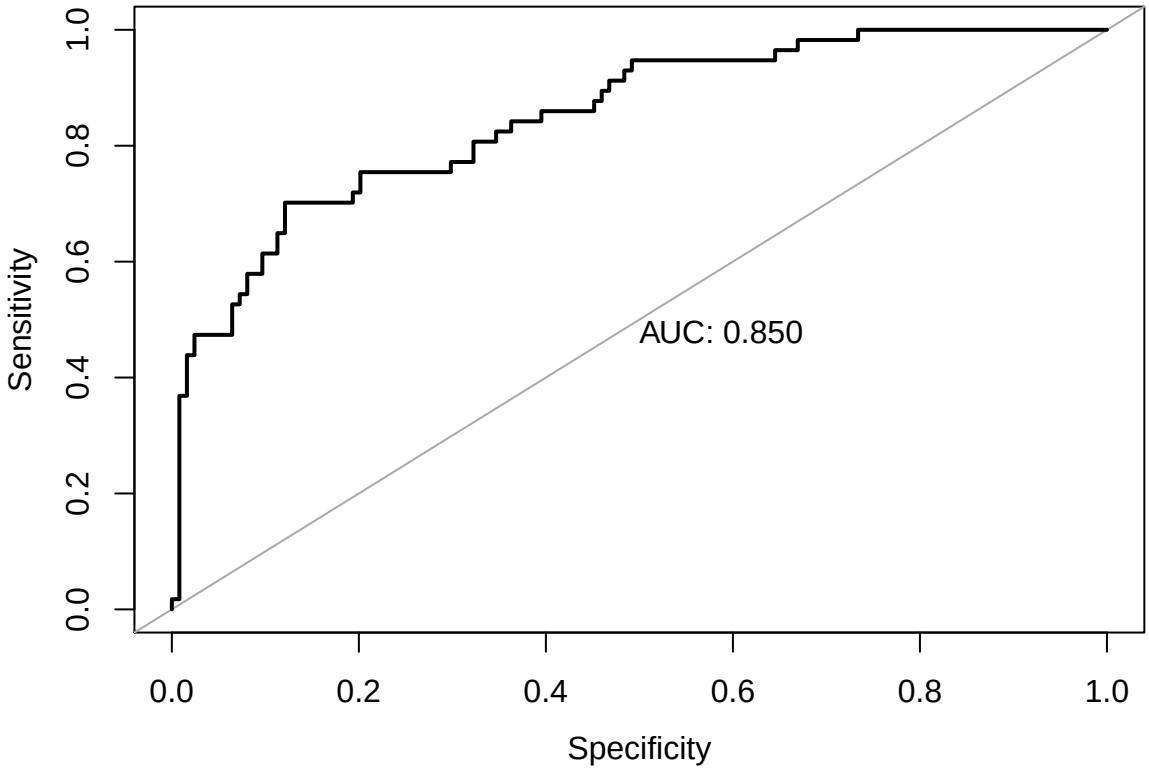
13) Investigate the pROCpackage. Use it to generate an ROC curve for the data set. How do the results compare with your own functions?

```
proc = roc(data$class, data$scored.probability)
```

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

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

```
plot(proc, asp=NA, legacy.axes=TRUE, print.auc=TRUE, xlab="Specificity")
```



The pROC ROC curve looks very similar to the curve generated by my function.

Appendix

Github repo at https://github.com/jmhsi/data621-HW1/tree/master/HW2