DATA 621 Homework 1

Ilya Kats

March 14, 2018

Introduction

Report below covers the second homework. I made an attemp to make the code more generic by requiring the dataframe, field names and positive/negative values to be passed into all functions (rather than relying simply on always having zeros and ones). At the end I also realized that I only needed to pass positive or negative value (not both). That would be a task for the first revision of the code. I have assumed that 0 is negative and 1 is positive, but this could be reversed with valid outcomes. In fact some of the `caret` code may treat 0 as positive if default values are used because it is the first value that is encountered.

1-2. Data Set

Provided file `classification-output-data.csv` contains 181 observations with 11 variables. Three variables will be considered for this report - `class` (actual class for the observation), `scored.class` (predicted class for the observation), and `scored.probability` (predicted probability of success for the observation).

Here is raw confusion matrix using `table()`:

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

Looking at the matrix above, rows represent actual class values of 0 or 1. Columns represent predicted class values of 0 or 1. So in the top left corner 119 is the number of observations where the class was correctly predicted to be 0. The top right corner shows 5 observations where the class of 0 was incorrectly predicted as 1. Similarly, we have 30 observations of class 1 incorrectedly predicted as class 0 and 27 observations of class 1 correctly predicted.

Assuming that 0 is a negative class and 1 is a positive class we have:

- 119 true negative observations (TN)
- 5 false positive observations (FP)
- 30 false negative observations (FN)
- 27 true positive observations (TP)

3-7. Calculating Accuracy, Classification Error Rate, Precision, Sensitivity and Specificity

First, let us define a generic function to return TP, TN, FP, and FN values based on provided dataframe, specified columns containing actual and predicted values and specified values to be considered True or False.

```
TF.values <- function(tbl, actual, predicted, pos_value, neg_value){
   conf.matrix <- as.data.frame(table(tbl[,c(actual)], tbl[,c(predicted)]))
   FP <- filter(conf.matrix, Var1==neg_value & Var2==pos_value)$Freq
   FN <- filter(conf.matrix, Var1==pos_value & Var2==neg_value)$Freq
   TP <- filter(conf.matrix, Var1==pos_value & Var2==pos_value)$Freq
   TN <- filter(conf.matrix, Var1==neg_value & Var2==neg_value)$Freq

   # Reset to 0 if no category was found
```

```
   if (length(FN)==0) FN<-0;
   if (length(FP)==0) FP<-0;
   if (length(TP)==0) TP<-0;
   if (length(TN)==0) TN<-0;
   return (list(TP=TP, FP=FP, TN=TN, FN=FN))
}

# Define values to treat as positive or negative
pos_value <- 1
neg_value <- 0
```

The limitation of this function is that it only works for observations containing only two classes.

Accuracy

```
accuracy <- function(tbl, actual, predicted, pos_value, neg_value){
  tf <- TF.values(tbl, actual, predicted, pos_value, neg_value)
  return ((tf$TP+tf$TN)/(tf$TP+tf$TN+tf$FP+tf$FN))
}

(acc <- accuracy(dt, actual='class', predicted='predicted',
                 pos_value, neg_value))
```

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

Classification Error Rate

```
error.rate <- function(tbl, actual, predicted, pos_value, neg_value){
  tf <- TF.values(tbl, actual, predicted, pos_value, neg_value)
  return ((tf$FP+tf$FN)/(tf$TP+tf$TN+tf$FP+tf$FN))
}

(cer <- error.rate(dt, actual='class', predicted='predicted',
                   pos_value, neg_value))
```

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

Accuracy and classification error rate should add up to one.

```
acc + cer
```

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

Precision

```
precision <- function(tbl, actual, predicted, pos_value, neg_value){
  tf <- TF.values(tbl, actual, predicted, pos_value, neg_value)
  return (tf$TP/(tf$TP+tf$FP))
}

(pre <- precision(dt, actual='class', predicted='predicted',
                  pos_value, neg_value))
```

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

Sensitivity

```
sensitivity <- function(tbl, actual, predicted, pos_value, neg_value){
  tf <- TF.values(tbl, actual, predicted, pos_value, neg_value)
  return (tf$TP/(tf$TP+tf$FN))
}

(sens <- sensitivity(dt, actual='class', predicted='predicted',
                     pos_value, neg_value))
```

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

Specificity

```
specificity <- function(tbl, actual, predicted, pos_value, neg_value){
  tf <- TF.values(tbl, actual, predicted, pos_value, neg_value)
  return (tf$TN/(tf$TN+tf$FP))
}

(spec <- specificity(dt, actual='class', predicted='predicted',
                     pos_value, neg_value))
```

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

8-9. F1 Score

```
f1.score <- function(tbl, actual, predicted, pos_value, neg_value){
  p <- precision(dt, actual='class', predicted='predicted',
                 pos_value, neg_value)
  s <- sensitivity(dt, actual='class', predicted='predicted',
                   pos_value, neg_value)
  return ((2*p*s)/(p+s))
}

(f1 <- f1.score(dt, actual='class', predicted='predicted',
                pos_value, neg_value))
```

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

Both *Precision* and *Sensitivity* have a range from 0 to 1. Consider that if $a > 0$ and $0 < b < 1$, then $ab < a$ (a fraction of any positive number will be smaller than the original number).

Then $Precision \times Sensitivity < Precision$ and $Precision \times Sensitivity < Sensitivity$.

Then $Precision \times Sensitivity + Precision \times Sensitivity < Precision + Sensitivity$, or

$2 \times Precision \times Sensitivity < Precision + Sensitivity$.

The fraction of these two values will be lower than 1. Also, since both values are positive, $F1$ *score* will be positive. If *Precision* is zero, then *Sensitivity* is zero and $F1$ *Score* is not defined. If *Precision* is one and *Sensitivity* is one, then $F1$ *Score* is one.

So we have $0 < F\ Score \leq 1$.

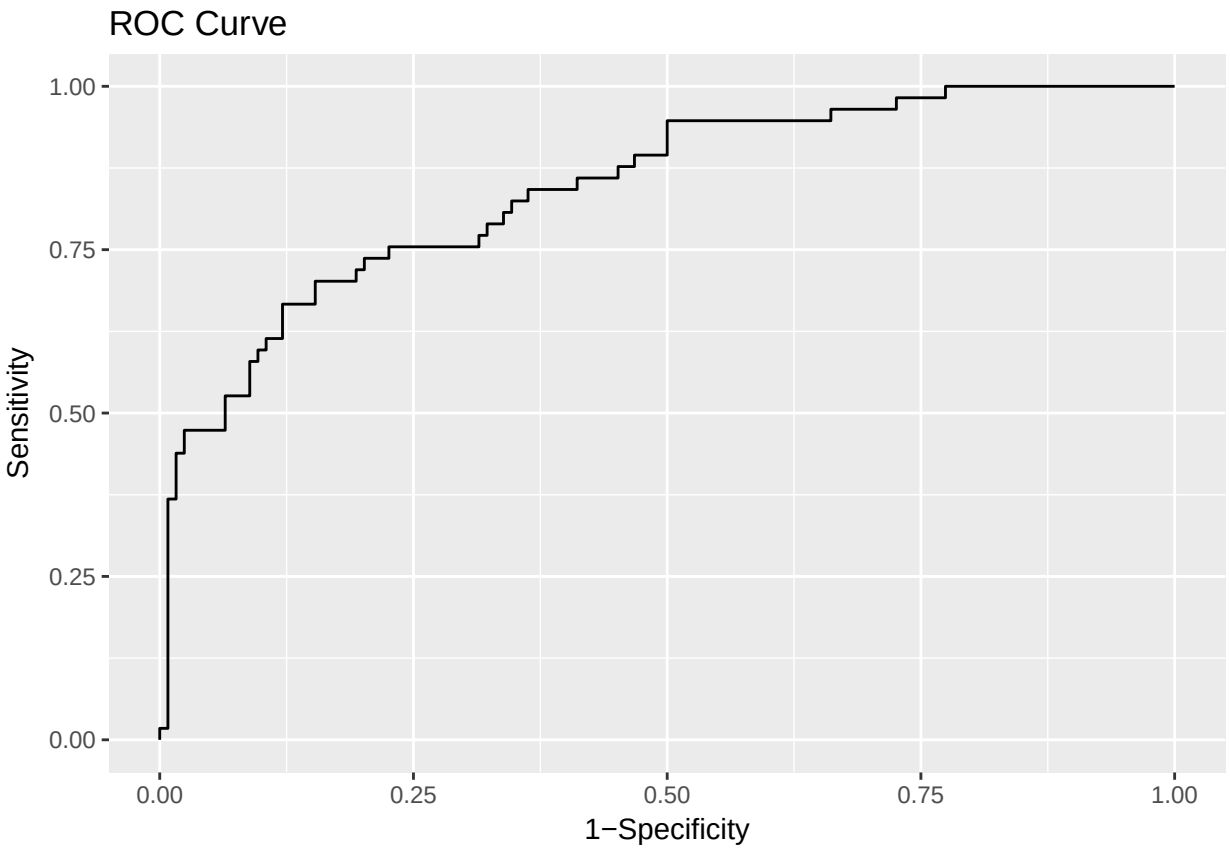
10. ROC Function

The following function calculates the proper specificity and sensivity based on provided probability by iterating from 0 to 1 at 0.01 interval. It then computes the area under the curve by summing up all relevant rectangles. Plot and AUC are returned.

```
manual.roc <- function(tbl, actual, prob, pos_value, neg_value) {
  tbl2 <- tbl[, c(actual, prob)]
  tbl2 <- cbind(tbl2, predicted = rep(0, nrow(tbl)))

  cutoff <- seq(0,1,0.01)
  sens <- rep(0,length(cutoff))
  spec <- rep(0,length(cutoff))
  for (i in 1:length(cutoff)) {
    tbl3 <- within(tbl2, predicted[prob>cutoff[i]] <- 1)
    sens[i] <- sensitivity(tbl3, actual='class',
                           predicted='predicted', pos_value, neg_value)
    spec[i] <- specificity(tbl3, actual='class',
                           predicted='predicted', pos_value, neg_value)
  }
  roc.values <- as.data.frame(cbind(cutoff, sens, spec))
  roc.values[,'spec'] <- 1 - roc.values[,'spec']

  # Prepare plot
  pl <- ggplot(arrange(roc.values, desc(cutoff)), aes(x=spec, y=sens)) +
    geom_step() +
    xlab("1-Specificity") + ylab("Sensitivity") +
    ggtitle("ROC Curve")

  # Find AUC
  auc <- roc.values %>%
    arrange(desc(cutoff)) %>%
    mutate(auc =  (spec-lag(spec))*sens) %>%
    replace(is.na(.),0) %>%
    summarize(sum(auc))

  return (list(plot=pl, AUC=auc))
}
```

Let us run the function and review plot and AUC value.

```
mROC <- manual.roc(dt, 'class', 'prob', pos_value, neg_value)
mROC$plot
```




```
mROC$AUC
```

```
##    sum(auc)
## 1 0.8539898
```

11. Classification Metrics

All metrics were provided as they were calculated. As we will see below using built-in functions makes life easier.

12. `caret` Package

`caret` package provides functions to calculate all of the values above and many more. For example, we can calculate sensitivity and specificity.

```
caret::sensitivity(as.factor(dt$predicted), as.factor(dt$class), positive='1')
```

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

```
caret::specificity(as.factor(dt$predicted), as.factor(dt$class), negative='0')
```

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

Even more encompassing function is `confusionMatrix`. It not only returns the actual confusion matrix, but other values including sensitivity and specificity.

```
(cm <- confusionMatrix(dt$predicted, dt$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
##
```

Even these results do not display all possible values. Let us look specifically at `byClass` values returned by `confusionMatrix`.

```
cm$byClass
```

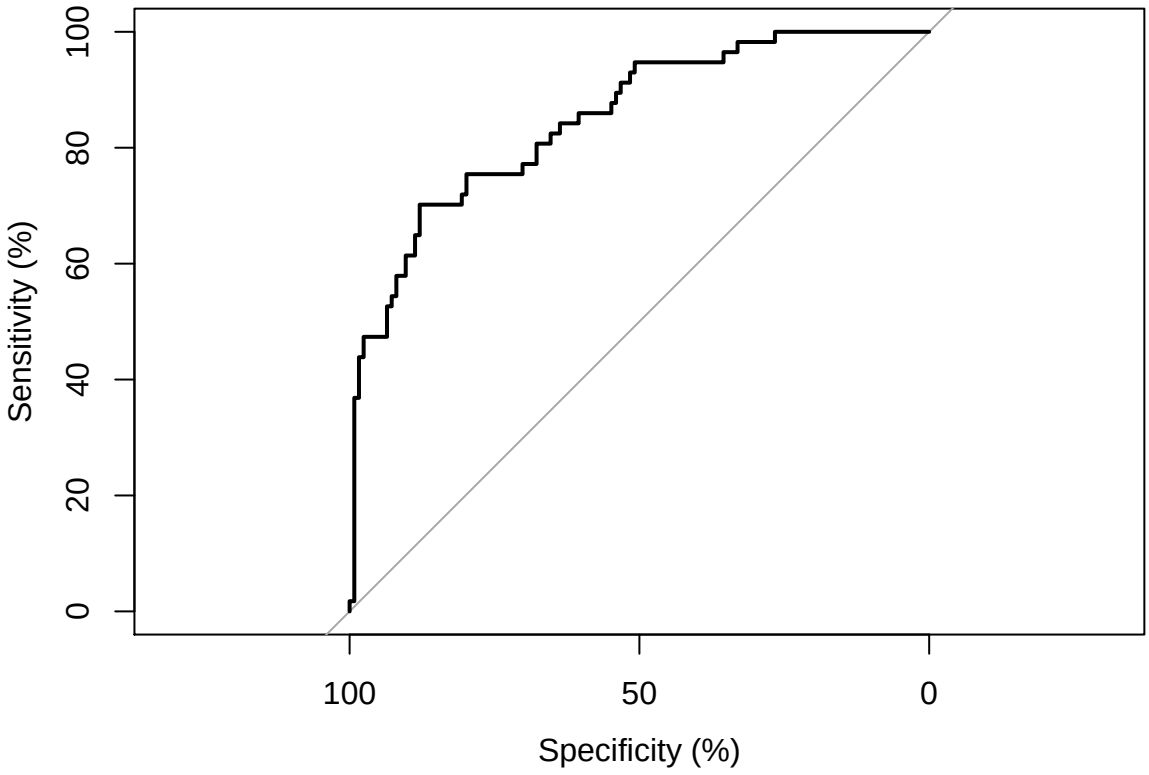
```
##          Sensitivity          Specificity       Pos Pred Value
##            0.4736842            0.9596774            0.8437500
##       Neg Pred Value            Precision               Recall
##            0.7986577            0.8437500            0.4736842
##                   F1           Prevalence       Detection Rate
##            0.6067416            0.3149171            0.1491713
## Detection Prevalence    Balanced Accuracy
##            0.1767956            0.7166808
```

We can see F1 score, Precision and other values and with higher precision than the summary output above. All of the values match our calculations.

13. `pROC` Package

Let us try the `pROC` package.

```
roc(dt$class, dt$prob, levels=c(0,1), percent=TRUE, plot=TRUE, ci=TRUE)
```




```
##
## Call:
## roc.default(response = dt$class, predictor = dt$prob, levels = c(0,     1), percent = TRUE, ci = TRU
##
## Data: dt$prob in 124 controls (dt$class 0) < 57 cases (dt$class 1).
## Area under the curve: 85.03%
## 95% CI: 79.05%-91.01% (DeLong)
```

With one line of code we can get the ROC curve and several metrics. This plot looks very similar to our manual plot (some difference is likely due to different aspects). We calculated AUC at 85.4% while `pROC` calculated it at 85.03%. The small difference is likely due to slight discrepancy in how steps were calculated (it is also possible that the code leaves out a sliver somewhere).