DATA 621 Homework 2

Critical Thinking Group 2

October 10, 2019

Introduction

This assignment assumes actual/predicted value of `0` to be negative and 1 to be positive, but this could be reversed with valid outcomes.

1-2. Data Set

The `classification-output-data.csv` contains 181 observations with 11 variables. We will be considering three observations for this work - `class` (actual class for the observation), `scored.class` (predicted class for the observation), and `scored.probability` (predicted probability of success for the observation).

Raw confusion matrix using `table()`:

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

From the observation we can understand that 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.

Based on the assumption of 0 as a negative class and 1 as 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

R function to calculate TP, FP, TN, and FP:

```
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
```

From this we can understand that the function only works for observations with two classes.

Accuracy

Accuracy is the number of correct predictions from all predictions made.

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

R function:

```
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))
}
```

Accuracy of the dataset:

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

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

Classification Error Rate

Classification error rate is the number of incorrect predictions from all predictions made.

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

R function:

```
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))
}
```

Classification error rate of the dataset:

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

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

When we add accuracy and classification error rate we should get one:

```
acc + cer
```

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

Precision

Precision measures how reliable a model's positive predictions are, i.e., $P(Positive \mid model = Positive)$. A poor (low) precision score indicates many of the positive predictions are likely to be *false* positives. For instance, if you have 100 positive (spam) predictions from a spam e-mail detection model, how many of them are actually positive (spam)? Thus the formula is:

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

R function:

```
precision <- function(actual, predicted, pos=1) {
    if (length(unique(actual)) != 2 | length(unique(predicted)) != 2) {
        stop('Actual and predicted vectors may only have two distinct values')
    }

    # set pos/neg to TRUE/FALSE as conveniance
    actual <- ifelse(actual == pos, TRUE, FALSE)
    predicted <- ifelse(predicted == pos, TRUE, FALSE)

    conf_matrix <- as.data.frame(table(actual, predicted)) %>%
        arrange(actual, predicted)

    TP <- conf_matrix$Freq[4]
    FP <- conf_matrix$Freq[2]

    return( TP / (TP + FP) )
}
```

Precision of the dataset:

```
(pre <- precision(actual=dt$class, predicted=dt$predicted, pos=1))
```

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

Sensitivity

Also called recall, this measures how often a model will return a positive result given a positive example. In a way, it is the reverse of precision, i.e., sensitivity is $P(model = Positive \mid Positive)$. For instance, if a spam e-mail detection model was given 100 spam e-mails, how many will the model label as spam (positive)? The resulting number is sensitivity. It is calculated by:

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

where a score of 1 indicates perfect sensitivity, the event where there are no false negatives.

Function to calculate sensitivity:

```
sensitivity <- function(actual, predicted, pos=1) {
    #if (length(unique(actual)) != 2 | length(unique(predicted)) != 2) {
    #    stop('Actual and predicted vectors may only have two distinct values')
    #}

    # set pos/neg to TRUE/FALSE as conveniance
    actual <- ifelse(actual == pos, TRUE, FALSE)
    predicted <- ifelse(predicted == pos, TRUE, FALSE)

    conf_matrix <- as.data.frame(table(actual, predicted)) %>%
        arrange(actual, predicted)

    TP <- conf_matrix$Freq[4]
    FN <- conf_matrix$Freq[3]

    return( TP / (TP + FN) )
}
```

Sensitivity of the dataset:

```
(sens <- sensitivity(actual=dt$class, predicted=dt$predicted, pos=1))
```

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

Specificity

Specificity is defined as the rate that nonevent samples (negatives) are predicted as nonevents (negatives). The false-positive rate is defined as one minus the specificity.

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

R function:

```
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))
}
```

Specificity of the dataset:

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

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

8. F1 Score

The F1 score is an attempt to balance sensitivity and precision, perhaps in cases where costs of false negative and false positive predictions are comparable. The formula is:

$$F1 = 2 \cdot \frac{Precision \cdot Sensitivity}{Precision + Sensitivity}$$

R function:

```
calc_f1 <- function(actual, predicted, pos=1) {
    if (length(unique(actual)) != 2 | length(unique(predicted)) != 2) {
        stop('Actual and predicted vectors may only have two distinct values')
    }

    # set pos/neg to TRUE/FALSE as conveniance
    actual <- ifelse(actual == pos, TRUE, FALSE)
    predicted <- ifelse(predicted == pos, TRUE, FALSE)

    # truncating in this annoying manner to avoid package conflicts with caret
    precis <- precision(actual, predicted, pos=pos)
    sensit <- sensitivity(actual, predicted, pos=pos)

    return( 2 * ( (precis * sensit) / (precis + sensit) ) )
}
```

The F1 score for the scored dataset is:

```
calc_f1(dt$class, dt$predicted)
```

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

9. F1 Bounds

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

In this section we will be calculating specificity and sensitivity on the basis of provided probability by iterating from 0 to 1 at 0.01 interval.

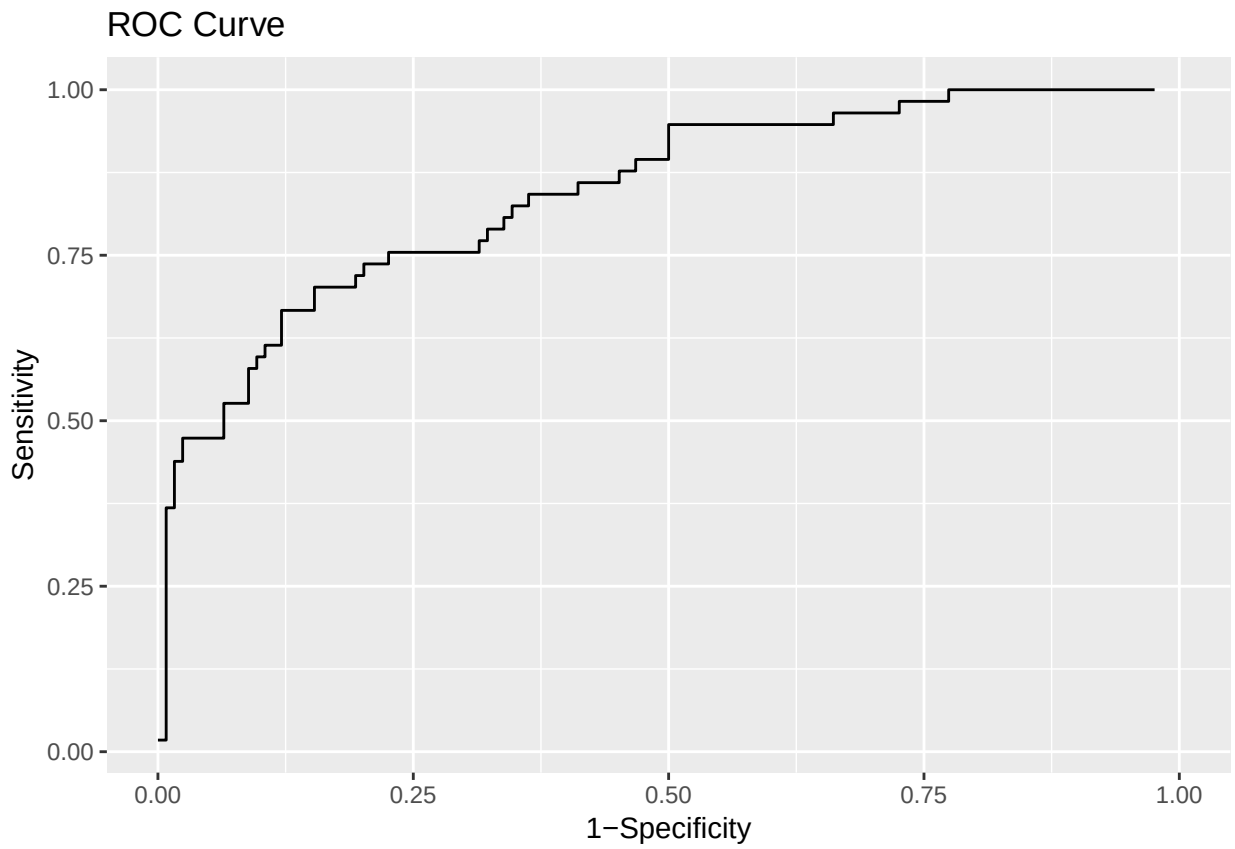
R function:

```
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(actual=tbl3$class, predicted=tbl3$predicted, pos=1)
    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
```

```
## Warning: Removed 9 rows containing missing values (geom_path).
```



Area Under Curve (AUC)

The higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.

```
mROC$AUC
```

```
##    sum(auc)
## 1 0.8297963
```

11. Classification Metrics

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

Accurary

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

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

Classification Error Rate

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

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

Precision

```
precision(actual=dt$class, predicted=dt$predicted, pos=1)
```

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

Sensitivity

```
sensitivity(actual=dt$class, predicted=dt$predicted, pos=1)
```

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

Specificity

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

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

F1 score

```
calc_f1(dt$class, dt$predicted)
```

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

12. caret Package

caret package helps us to calculate sensitivity and specificity.

Sensitivity from caret package:

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

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

Specificity from caret package:

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

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

confusionMatrixfunction can also be used to calculate sensitivity and specificity:

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

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.

```
cm$byClass
```

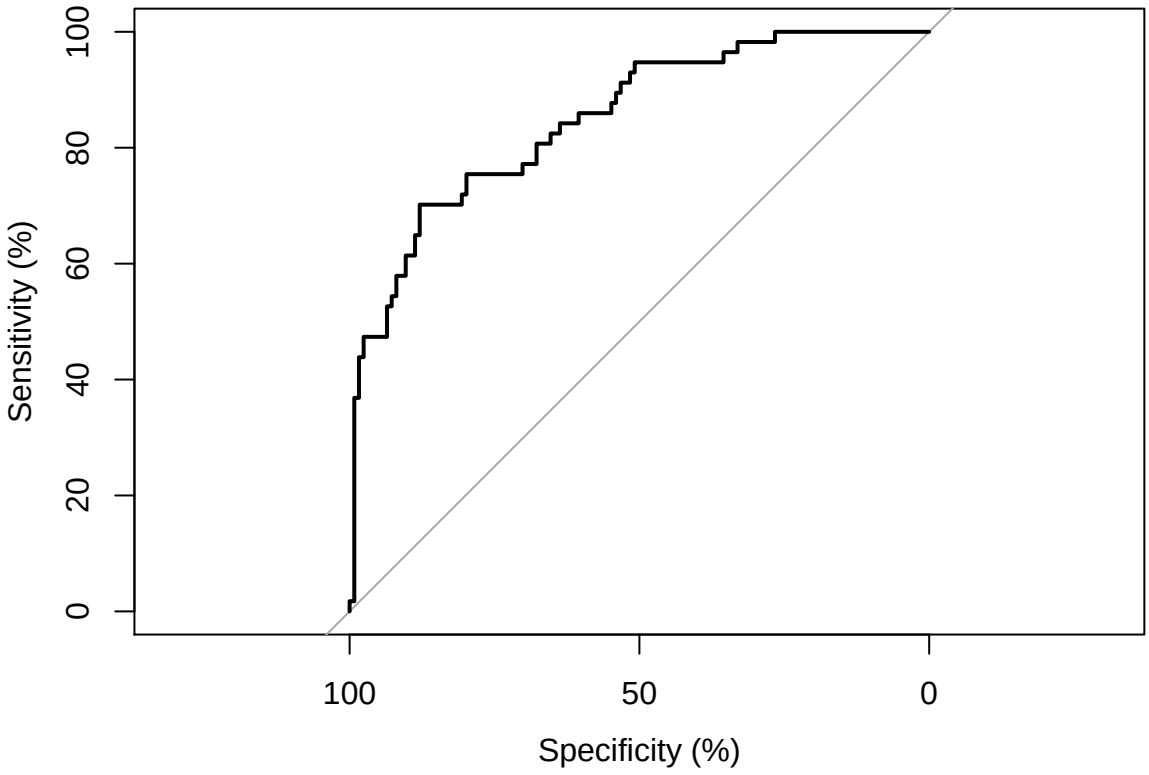
```
##          Sensitivity          Specificity       Pos Pred Value
##            0.4736842            0.9596774            0.8437500
##       Neg Pred Value            Precision               Recall
##            0.7986577            0.8437500            0.4736842
##                   F1         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

13. Investigate the pROC package. Use it to generate an ROC curve for the data set.

Let us try the `pROC` package.

The `pROC` package calculates area under the curve to be 85.03%, which is higher compared to 0.8297963 that was calculated by the manual roc.

```
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)
```