HW2 Classification

Critical Thinking Group One

2021-03-19

Contents

Authorship

Critical Thinking Group 1:

- Angel Claudio,
- Bonnie Cooper,
- Manolis Manoli,
- Magnus Skonberg,
- Christian Thieme and
- Leo Yi

Background

In the following exercises we will be working with a version of a well known dataset known as the "Pima Indians Diabetes Database". The dataset was gathered by the National Institute of Diabetes and Digestive and Kidney Diseases with the purpose of determining which variables are predictive of diabetes. In our version of the dataset, we begin with the original columns describing various health measurements of an individual as well as two columns, `scored.class` and `scored.probability`, which are the predictions of a previously run classification model. We will use this dataset to explore different model diagnostic metrics associated with classification models. We will begin by calculating these metrics manually to solidify our intuition into how to use them; then, we'll explore R packages that automate these calculations and allow for quick diagnostics.

Task 1 Import

We download the classification output data set and then utilize the built-in `glimpse()` method to gain insight into the dimensions, variable characteristics, and value range:

Solution

```
## Rows: 181
## Columns: 11
## $ pregnant           <dbl> 7, 2, 3, 1, 4, 1, 9, 8, 1, 2, 5, 5, 13, 0, 7, 12, 0~
## $ glucose            <dbl> 124, 122, 107, 91, 83, 100, 89, 120, 79, 123, 88, 1~
## $ diastolic          <dbl> 70, 76, 62, 64, 86, 74, 62, 78, 60, 48, 78, 72, 60,~
## $ skinfold           <dbl> 33, 27, 13, 24, 19, 12, 0, 0, 42, 32, 30, 43, 0, 26~
## $ insulin            <dbl> 215, 200, 48, 0, 0, 46, 0, 0, 48, 165, 0, 75, 0, 50~
## $ bmi                <dbl> 25.5, 35.9, 22.9, 29.2, 29.3, 19.5, 22.5, 25.0, 43.~
## $ pedigree           <dbl> 0.161, 0.483, 0.678, 0.192, 0.317, 0.149, 0.142, 0.~
## $ age                <dbl> 37, 26, 23, 21, 34, 28, 33, 64, 23, 26, 37, 33, 41,~
## $ class              <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, ~
## $ scored.class       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, ~
## $ scored.probability <dbl> 0.32845226, 0.27319044, 0.10966039, 0.05599835, 0.1~
```

The classification dataset is hosted on a public GitHub repo. The variable of interest is the `class` column. `class` has the following two values:

- 0 : does not have diabetes
- 1 : has diabetes

Task 2 Raw Confusion Matrix

The data set has three key columns we will use:

- **class:** the actual class for the observation
- **scored.class:** the predicted class for the observation (based on a threshold of 0.5)
- **scored.probability:** the predicted probability of success for the observation

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?

Solution

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

The confusion matrix displays real observations as columns and predicted classifications as rows. The values in the left column represent the number of cases where the model predicted no diabetes and the case indeed did not have diabetes (top-left) and the number of cases where the model assigned a positive diabetes diagnosis when this was false (bottom-left).

Conversely, the values in the right column represent the cases where the model predicted no diabetes when in fact the case was positive (top-right) and the occurrences where the model predicted positive diabetes and the patient indeed had diabetes (bottom-right). Looking at the diagonal values (starting from top-left to bottom-right), show how the model performed in selecting the actual class.

Task 3 Accuracy Function

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

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

Solution

```
get_accuracy <- function(df){
  dt <- df %>% dplyr::select( scored.class, class ) %>% table()

  TP <- dt[2,2] # True Positive
  TN <- dt[1,1] # True Negative
  FP <- dt[2,1] # False Positive
  FN <- dt[1,2] # False Negative

  return(round((TP + TN)/(TP + FP + TN + FN),4))
}

paste('Accuracy:', get_accuracy(classification_df))
```

```
## [1] "Accuracy: 0.8066"
```

Accuracy is a high level spot check for classification models. It tells us overall, how well the model is predicting the outcomes. However, if you have imbalanced classes in your response variable, accuracy won't be a very helpful metric. For example, if you are trying to detect fraud, and only 1 out of 100 transactions are fraudulent, if you predict every case as having no fraud, you will still have a model that is 99% accurate. However, maybe that one fraudulent case causes millions of dollars in damages. We know that for a model like this, that predicting the 1 fraudulent transaction would be the focus of the model, and that our metrics should be tuned to help us understand how well the model is performing in that regard. As such, in most classification settings, accuracy needs to be broken down even further to analyze how accurately the model is predicting the outcomes we care about.

Task 4 Classification Error Rate

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.

$$Classification\ Error\ Rate = \frac{FP + FN}{TP + FP + TN + FN}$$

Solution

```
get_classification_error <-  function(df){
  dt <- df %>% dplyr::select( scored.class, class ) %>% table()

  TP <- dt[2,2] # True Positive
  TN <- dt[1,1] # True Negative
  FP <- dt[2,1] # False Positive
  FN <- dt[1,2] # False Negative

  return(round((FP + FN)/(TP + FP + TN + FN), 4))
}

paste('Classification Error:',get_classification_error(classification_df))
```

```
## [1] "Classification Error: 0.1934"
```

The classification error rate is the opposite of accuracy. In this metric we want to determine the percentage of time our outcomes are being misclassified. We can see that if we were to add the accuracy from Task 3 (.807) with the above classification error (.193), we would get 1 for verification purposes, as the error is filling in the missing accuracy.

Task 5 Precision

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

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

Solution

```
get_precision <-  function(df){
  dt <- df %>% dplyr::select( scored.class, class ) %>% table()

  TP <- dt[2,2] # True Positive
  FP <- dt[2,1] # False Positive

  return(round((TP)/(TP + FP), 4))
}

paste('Precision:', get_precision(classification_df))
```

```
## [1] "Precision: 0.8438"
```

Precision is looking at the ratio of true positives to the predicted positives. This metric becomes very important when having false positives is unwanted. In a model that is classifying email as SPAM or HAM, we would want an extremely high precision so that the model wasn't misclassifying good emails as bad (false positives). The cost of a false positive is high here.

Task 6 Sensitivity

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.

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

Solution

```
get_sensitivity <-  function(df){
  dt <- df %>% dplyr::select( scored.class, class ) %>% table()

  TP <- dt[2,2] # True Positive
  FN <- dt[1,2] # False Negative

  return(round((TP)/(TP + FN), 4))
}

paste('Sensitivity:', get_sensitivity(classification_df))
```

```
## [1] "Sensitivity: 0.4737"
```

Sensitivity or recall is important when you are concerned with identifying positive outcomes. This metric is key if it is imperative that you identify positive cases, even if you pick up some false positives along the way. For example, if you are predicting whether a patient has cancer or not, it is important that the sensitivity be incredibly high so that we can catch the most positive cases possible, even if it means we pull in a few patients who don't actually have cancer. The cost of a false positive is low here.

Task 7 Specificity

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

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

Solution

```r
get_specificity <-  function(df){
  dt <- df %>% dplyr::select( scored.class, class ) %>% table()

  TN <- dt[1,1] # True Negative
  FP <- dt[2,1] # False Positive

  return(round((TN)/(TN + FP), 4))
}

paste('Specificity:', get_specificity(classification_df))
```

```
## [1] "Specificity: 0.9597"
```

Specificity is the ratio of true negatives to all negative outcomes. You would be interested in this metric if you were concerned about the accuracy of your negative rate and there was a high cost to a positive outcome, so you don't want to blow this whistle if you don't have to. For example, if you're an auditor looking over financial transactions and a positive outcome would mean a 1 year investigation, but not finding one would only cost the company $100.

Task 8 F1 Score Function

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.

$$F1\ Score = \frac{2 \text{ x Precision x Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

Solution

```r
get_f1_score <- function (df) {

  precision <- get_precision(df)
  sensitivity <- get_sensitivity(df)

  return(round((2 * precision * sensitivity)/(precision + sensitivity), 4))
  }

paste('F1 Score:', get_f1_score(classification_df))
```

```
## [1] "F1 Score: 0.6068"
```

The F1 score is calculated with both precision and sensitivity (recall). It becomes important if you have an uneven class distribution, because it demonstrates the balance between precision and sensitivity. The first accuracy metric we described above had a significant weakness in that it was heavily influenced by the amount of true negatives. The F1 score tries to avoid that by focusing on false positives and false negatives. Because it combines both precision and sensitivity, the F1 score is often most useful when comparing different models running on the same dataset.

Task 9 F1 Score Bounds

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)

Solution

$$\text{given: } Precision = \frac{TP}{TP + FP}$$

$$\text{given: } Sensitivity = \frac{TP}{TP + FN}$$

$$\text{given: } F1\ Score = \frac{\text{2 x Precision x Sensitivity}}{\text{Precision + Sensitivity}}$$

$$\therefore F1\ Score = \frac{\frac{2\ TP^2}{(TP+FP)(TP+FN)}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}}$$

$$= \frac{2\ TP^2}{(TP + FP)(TP + FN)} \times \frac{1}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}}$$

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

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

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

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

$$F1\ Score = \frac{2\ TP}{TP + FN + TP + FP}$$

$$\text{given: } \{TP, FN, FP \in \mathbb{N}\}$$

$$2TP \leq TP + FN + TP + FP$$

$$\therefore 0 \leq F1\ Score \leq 1$$

Task 10 ROC Function

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.

Solution

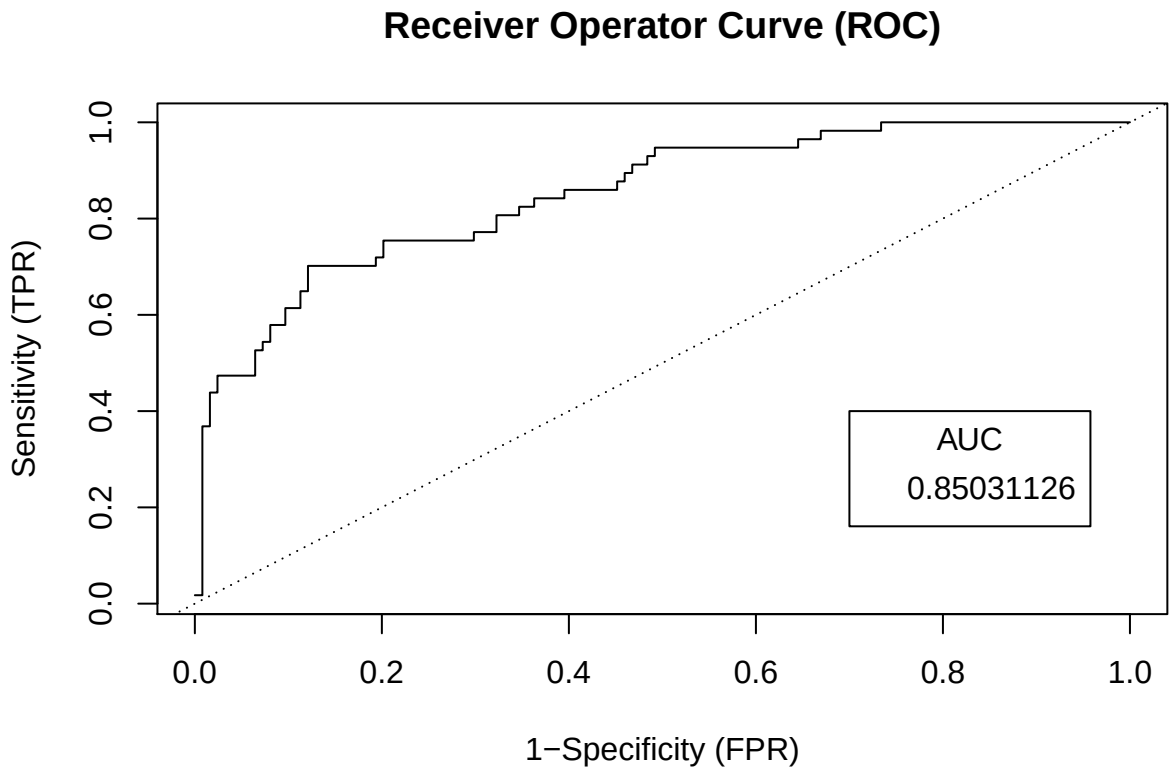
```
get_roc_curve <- function(x,p){
    #order x by descending p
    x <- x[order(p, decreasing=TRUE)]

    TP = cumsum(x)/sum(x)
    FP = cumsum(!x)/sum(!x)

    roc_df <- data.frame(TP, FP)
    auc <- sum(TP * c(diff(FP), 0)) + sum(c(diff(TP), 0) * c(diff(FP), 0))/2

    return(c(df=roc_df, auc = auc))
}

#apply to our dataset
roc_data <- get_roc_curve(classification_df$class, classification_df$scored.probability)
plot(roc_data[[2]],roc_data[[1]], type = 'l', main = "Receiver Operator Curve (ROC)",xlab="1-Specificity
abline(0,1, lty=3)
legend(0.7,0.4, round(roc_data$auc,8), title = 'AUC')
```



The ROC curve is a plot of the true positive rate against the false positive rate. It gives us a visual way to inspect what happens to our accuracy metrics as we adjust the probability thresholds that classify the observations into classes.

The AUC is a metric that gives us a way to measure the model's ability to separate positive from negative outcomes. The higher the AUC, the better the model is at predicting outcomes correctly.

Task 11 Use Predefined R Functions

Solution

```
##        metric  value
## 1    Accuracy 0.8066
## 2  Error Rate 0.1934
## 3   Precision 0.8438
## 4 Sensitivity 0.4737
## 5 Specificity 0.9597
## 6    F1 Score 0.6068
```

If the goal of this model was to identify as many Pima Indians as possible who had diabetes so we could educate them and get them on treatment plans, this model would be a failure. Our sensitivity rate is ~0.47, which means it is pretty bad at accurately predicting the positive class. We would want to try some other models on this dataset and compare the sensitivity and the F1 Scores.

Task 12 caret Package

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?

Solution

```
#Convert to factors to feed caret functions
classification_df$actual <- factor(classification_df$class, levels = c(1,0), labels = c('Diabetes', 'No
classification_df$predicted <- factor(classification_df$scored.class, levels = c(1,0), labels = c('Diabe

#compare caret vs table confusion matrix
(confusionMatrix(data = classification_df$predicted, reference = classification_df$actual, positive = '
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Diabetes No Diabetes
##   Diabetes          27           5
##   No Diabetes       30         119
##
##                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 : 0.00004976
##
```

```
##             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 : Diabetes
##
```

```
conf
```

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

In applying the `confusionMatrix` function to our dataset we can see the output of the confusion matrix which matches what we created manually above (although columns and rows are ordered differently). Additionally, in the output above, we see the sensitivity and specificity scores match the percentages we calculated manually as well.

In addition, we'll check our manually calculated sensitivity and specificity scores against those calculated by the `sensitivity` and `specificity` functions from the `caret` package.

```
#Utilitize caret sensitivity function
(caret_sensitivity <- caret::sensitivity(data = classification_df$predicted,
                                         reference = classification_df$actual,
                                         positive = 'Diabetes'))
```

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

```
#Compare caret vs created sensitivity function
round(caret_sensitivity,4) == round(cl_sensitivity,4)
```

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

Our sensitivity score matches perfectly.

```
#Utilitize caret specificity function
(caret_specificity <- caret::specificity(data = classification_df$predicted,
                                         reference = classification_df$actual,
                                         positive = 'Diabetes'))
```

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

```
#Compare caret vs created specificity function
round(caret_specificity,4) == round(cl_specificity,4)
```

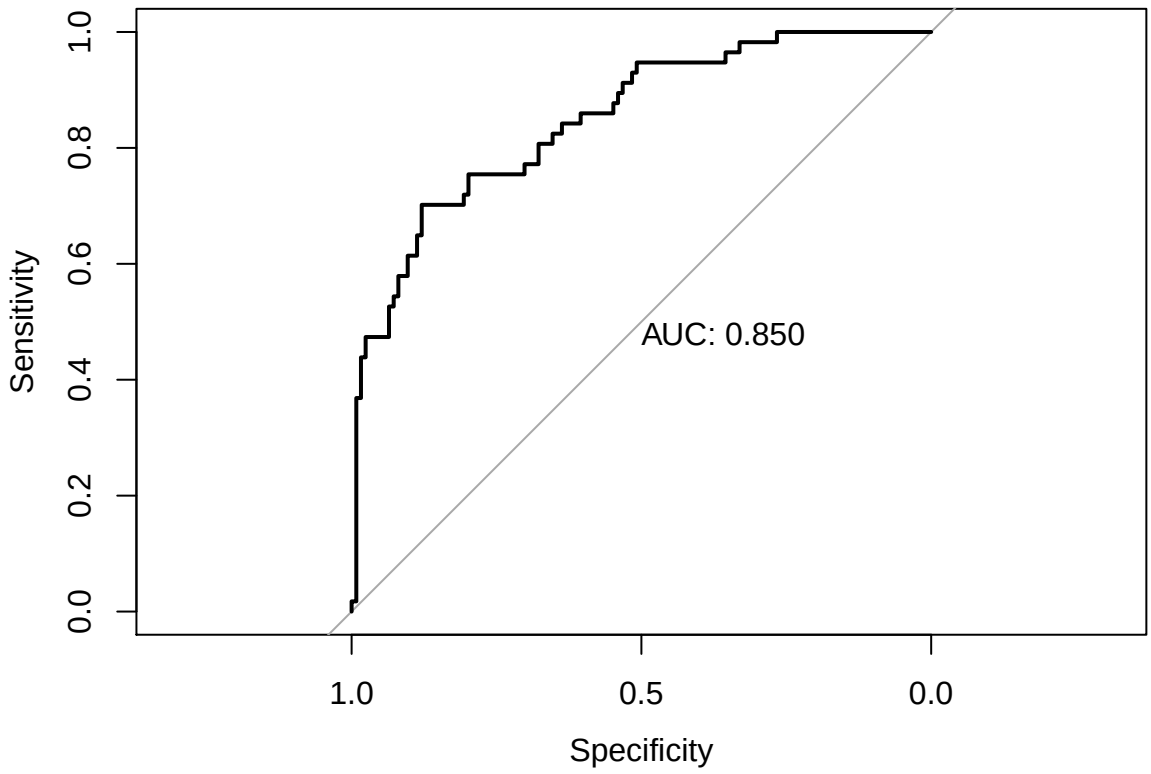
```
## [1] TRUE
```

As does our specificity score.

Task 13 pROC Package

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

Solution




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

The results we see above appear to be exactly the same as those we calculated manually.

Conclusion

We have presented here an incremental approach to exploring different model diagnostic metrics associated with classification models. We calculated the metrics manually, confirmed their consistency with built in pROC and caret functions, and analyzed our output each step of the way.

The type of metric we use to measure our model's performance depends on the purpose of the model. This is often determined by analyzing what the cost for misclassifications are (false negatives and false positives).

Using the metrics outlined above, we were able to quickly determine the model's predictive capabilities and suitability. Thus confirming their value and ease of interpretability.

Based on our relatively high error rate (~0.20) and sensitivity scores (~0.50), we'd recommend trying different models and optimizing our fit to more precisely predict positive cases.

. . . . . .

References

In solving HW2, we referred to the following:

1. Ruchi Toshniwal. (2020). **How to select Performance Metrics for Classification Models** [article]. Retrieved from https://medium.com/analytics-vidhya/how-to-select-performance-metrics-for-classification-models-c847fe6b1ea3
2. Yihui Xie, Christophe Dervieux, Emily Riederer. (2020) **R Markdown Cookbook** [e-book]. Retrieved from https://bookdown.org/yihui/rmarkdown-cookbook/code-appendix.html

. . . . . .

Appendix: `R` Statistical Code

In solving HW2, we utilized the following code chunks to solve questions where defining a function was not required:

Task 1 Import

```
classification_df <-
  readr::read_csv(paste0('https://raw.githubusercontent.com',
                         '/dataconsumer101/data621/main/hw2/classification-output-data.csv'))

dplyr::glimpse( classification_df )
```

Task 2 Raw Confusion Matrix

```
## Task 2 Raw Confusion Matrix
conf <- classification_df %>%
        dplyr::select( scored.class, class ) %>%
        table()
conf
```

Tasks 3 - 10

Corresponding code / equations noted above.

Task 11 Use Predefined R Functions

```r
## Task 11 Use Predefined R Functions
cl_accuracy <- get_accuracy(classification_df)
cl_error <- get_classification_error(classification_df)
cl_precision <- get_precision(classification_df)
cl_sensitivity <- get_sensitivity(classification_df)
cl_specificity <- get_specificity(classification_df)
cl_f1_score <- get_f1_score(classification_df)

(output <- data.frame(
    metric = c('Accuracy', 'Error Rate', 'Precision', 'Sensitivity', 'Specificity', 'F1 Score'),
    value = c(cl_accuracy, cl_error, cl_precision, cl_sensitivity, cl_specificity, cl_f1_score)))
```

Task 12 caret Package

Corresponding code noted above to provide context.

Task 13 pROC Package

```r
roc(classification_df$class,classification_df$scored.probability, plot = T, print.auc = T)
```