Package **caret**

Package caret

- The caret package (short form for Classification And REgression Training) is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for:
  - **data splitting**
  - pre-processing
  - feature selection
  - model tuning using resampling
  - variable importance estimation
  - **model performance**

- For more information: http://topepo.github.io/caret/index.html

Data Splitting Based on Outcome

- The function createDataPartition( ) can be used to create balanced splits of the data.
- Syntax : createDataPartition(y, p=0.5, list=TRUE,...)

Where

- y : Vector of outcomes. In case if y is factor, then sampling is done within the levels of y. In case if y is numerical, then sample is split into groups based on percentiles and sampling is done within these groups
- p : Percentage of Data that goes to training
- list : Logical value indicating whether the results should be in a form of list.TRUE by default

Output of createDataPartition()

- The function createDataPartition() outputs the vector / list of the indices of the observations we want in the sample.
- For random number generation we need to set the seed for randomization with set.seed(), otherwise the seed will be taken as system time and we won’t be able to have a uniformity across the different executions of the same program

Creating Partitions

```r
set.seed(333)
intrain <- createDataPartition(y=brupt$Class,p=0.7,list = FALSE)
```

Resample1
2
3
4
5
6
7
8
9
11
12
13
14
15
16
17
18
20

Setting of Random Seed

A vector of the indices of the observations to be included in the sample

```r
training <- brupt[intrain,   ]
validation <- brupt[-intrain,]
```

Data Partitioned

Example of createDataPartition()

```
> prop.table(table(training$Class))

        B        NB 
0.4261364 0.5738636 
> prop.table(table(validation$Class))

        B        NB 
0.4324324 0.5675676 
```

- In the above program, observe that the proportions of the levels for the outcome variable brupt$Class have come out to be nearly equal.
- This is the benefit we get from createDataPartition function which we can't get with sample() function.

Model Performance

```
> PredY # Predicted Vector
 [1] NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB
[32] NB NB NB NB NB NB NB NB NB NB NB B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B
[63] NB B  B  B  B  B  B  B  B  B  B  B
Levels: B NB
> validation[,7] # Existing Values Vector in Validation Set
 [1] NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB
[32] NB NB NB NB NB NB NB NB NB NB NB B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B
[63] B  B  B  B  B  B  B  B  B  B  B  B
Levels: B NB
```

- When we are done with predicting the values using any algorithm, we find the accuracy of the results with functions confusionMatrix() and postResample()

Function confusionMatrix()

Syntax : confusionMatrix(data, reference, dnn = c("Prediction", "Actual"),...)

Where

data : Predicted Values Vector in factor form or an object of class table

reference : Actual Values Vector from validation set in factor form

dnn : labels for the table

Examples of Confusion Matrix creation

```
> confusionMatrix(PredY, validation[,7],dnn=list('predicted','actual'))
Confusion Matrix and Statistics

         actual
predicted  B NB
       B  31  0
       NB  1 42

               Accuracy : 0.9865
                 95% CI : (0.927, 0.9997)
    No Information Rate : 0.5676
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9724
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9688
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9767
             Prevalence : 0.4324
         Detection Rate : 0.4189
   Detection Prevalence : 0.4189
      Balanced Accuracy : 0.9844

       'Positive' Class : B
```

Examples of Confusion Matrix creation

```
> tbl <- table(PredY, validation[,7],dnn=list('predicted','actual'))
> confusionMatrix(tbl)
Confusion Matrix and Statistics

         actual
predicted  B NB
       B  31  0
       NB  1 42

               Accuracy : 0.9865
                 95% CI : (0.927, 0.9997)
    No Information Rate : 0.5676
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9724
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9688
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9767
             Prevalence : 0.4324
         Detection Rate : 0.4189
   Detection Prevalence : 0.4189
      Balanced Accuracy : 0.9844

       'Positive' Class : B
```

Output of confusionMatrix()

	Reference	
Predicted	Event	No Event
Event	A	B
No Event	C	D

The formulas used here are:

$$Sensitivity = \frac{A}{A+C}$$

$$Specificity = \frac{D}{B+D}$$

$$Prevalence = \frac{A+C}{A+B+C+D}$$

$$PPV = \frac{sensitivity \times prevalence}{((sensitivity \times prevalence) + ((1 - specificity) \times (1 - prevalence))}$$

$$NPV = \frac{specificity \times (1 - prevalence)}{((1 - sensitivity) \times prevalence) + ((specificity) \times (1 - prevalence))}$$

$$Detection\ Rate = \frac{A}{A+B+C+D}$$

$$Detection\ Prevalence = \frac{A+B}{A+B+C+D}$$

Function postResample()

Syntax : postResample(pred, obs)

Where

pred : Vector with predicted values

obs : Vector with existing values(from validation/test sets)

- This function calculates the performance across resamples
- Given two numeric vectors of data, the mean squared error and R-squared are calculated.
- For two factors, the overall agreement rate and Kappa are determined.

Examples of postResample()

```
> pred.demand
        7         8        11        13        19        20        24        31        33 
 88.34375  43.70175  43.70175 114.81818  88.34375 114.81818  43.70175  43.70175  88.34375 
       34        40        44        51        52        53        54        60        61 
114.81818  88.34375  43.70175  43.70175  43.70175  43.70175  88.34375  43.70175  88.34375 
       63        78        84        86        93        95        98       101       102 
 88.34375  43.70175  88.34375  43.70175  43.70175  43.70175  88.34375  43.70175  43.70175 
      103       104       109       114       119       120       121       122       123 
 88.34375  88.34375  43.70175  43.70175  88.34375  43.70175  43.70175  43.70175  43.70175 
      125       128       135       140 
114.81818  43.70175  43.70175  88.34375 
> testing$Demand
 [1]  64  47  36 107  81 102  43  43 104  97  97  38  53  53  48  88  46  58  69  24  74  46  35
[24]  40  69  42  49 104 127  42  31  97  51  46  52  48 133  42  47  92
```

```
> postResample(pred.demand, testing$Demand) # For numeric vectors
     RMSE  Rsquared 
12.923683  0.789429 
```

Examples of postResample()

```
> PredY
 [1] NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB
[32] NB NB NB NB NB NB NB NB NB NB NB B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B
[63] NB B  B  B  B  B  B  B  B  B  B  B
Levels: B NB
> validation[,7]
 [1] NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB NB
[32] NB NB NB NB NB NB NB NB NB NB NB B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B
[63] B  B  B  B  B  B  B  B  B  B  B  B
Levels: B NB
```

```
> postResample(PredY, validation[,7]) # For factor variables
 Accuracy     Kappa
0.9864865 0.9723674
```