COVID-19 Healthy Diet - CART, Random Forest

Yunlong Pan 113061415

5/9/2020

Introduction

In the past couple months, we've witnessed doctors, nurses, paramedics and thousands of medical workers putting their lives on the frontline to save patients who are infected. And as the battle with COVID-19 continues, we should all ask ourselves – What should we do to help out? What can we do to protect our loved ones, those who sacrifice for us, and ourselves from this pandemic?

And a simple answer is : We need to protect our families and our own healths by adapting to a healthy diet.

The USDA Center for Nutrition Policy and Promotion recommends a very simple daily diet intake guideline: 30% grains, 40% vegetables, 10% fruits, and 20% protein, but are we really eating in the healthy eating style recommended by these food divisions and balances?

In this dataset, I have combined data of different types of food, world population obesity and undernourished rate, and global COVID-19 cases count from around the world in order to learn more about how a healthy eating style could help combat the Corona Virus. And from the dataset, we can gather information regarding diet patterns from countries with lower COVID infection rate, and adjust our own diet accordingly.

Libraries Required

```r
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.7-1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(maps)
library(caTools)
```

Load Data

```
data_kg<-read.csv('Food_Supply_Quantity_kg_Data.csv')
data_kg1<-data_kg[-c(32,31,30,29,28,27,26,1)]
data_kg2<-data.frame(data_kg1,'HighRecovery'<-as.numeric(data_kg$Recovered>0.01))
colnames(data_kg2)[ncol(data_kg2)]<-'HighRecovery'
# data_kg3<-na.omit(data_kg2)
```

Exploratory Data Analysis and Descriptive Statistics

```
# Find out Total Number of Rows and Columns
dim(data_kg2)
```

```
## [1] 170  25
```

```
# Find out Names of the Columns (Features)
names(data_kg2)
```

```
##  [1] "Alcoholic.Beverages"      "Animal.fats"
##  [3] "Animal.Products"          "Aquatic.Products..Other"
##  [5] "Cereals...Excluding.Beer" "Eggs"
##  [7] "Fish..Seafood"            "Fruits...Excluding.Wine"
##  [9] "Meat"                     "Milk...Excluding.Butter"
## [11] "Miscellaneous"            "Offals"
## [13] "Oilcrops"                 "Pulses"
## [15] "Spices"                   "Starchy.Roots"
## [17] "Stimulants"               "Sugar...Sweeteners"
## [19] "Sugar.Crops"              "Treenuts"
## [21] "Vegetable.Oils"           "Vegetables"
## [23] "Vegetal.Products"         "Obesity"
## [25] "HighRecovery"
```

```
# Find out Class of each Feature, along with internal structure
str(data_kg2)
```

```
## 'data.frame':    170 obs. of  25 variables:
##  $ Alcoholic.Beverages     : num  0.0014 1.6719 0.2711 5.8087 3.5764 ...
##  $ Animal.fats             : num  0.1973 0.1357 0.0282 0.056 0.0087 ...
##  $ Animal.Products         : num  9.43 18.77 9.63 4.93 16.66 ...
##  $ Aquatic.Products..Other : num  0 0 0 0 0 0 0 0.0033 0.0011 0 ...
##  $ Cereals...Excluding.Beer: num  24.81 5.78 13.68 9.11 6 ...
##  $ Eggs                    : num  0.2099 0.5815 0.5277 0.0587 0.2274 ...
##  $ Fish..Seafood           : num  0.035 0.213 0.242 1.771 4.149 ...
##  $ Fruits...Excluding.Wine : num  5.35 6.79 6.38 6 10.75 ...
##  $ Meat                    : num  1.2 1.88 1.13 2.06 5.69 ...
##  $ Milk...Excluding.Butter : num  7.583 15.721 7.619 0.831 6.366 ...
##  $ Miscellaneous           : num  0.0728 0.1123 0.1671 0.1165 0.7139 ...
##  $ Offals                  : num  0.206 0.232 0.087 0.155 0.222 ...
##  $ Oilcrops                : num  0.07 0.938 0.349 0.419 0.217 ...
##  $ Pulses                  : num  0.295 0.238 0.478 0.651 0.184 ...
##  $ Spices                  : num  0.0574 0.0008 0.0557 0.0009 0.1524 ...
##  $ Starchy.Roots           : num  0.88 1.81 4.13 18.11 1.45 ...
##  $ Stimulants              : num  0.3078 0.1055 0.2216 0.0508 0.1564 ...
##  $ Sugar...Sweeteners      : num  1.35 1.54 1.83 1.85 3.87 ...
```

```
##  $ Sugar.Crops            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Treenuts               : num  0.077 0.1515 0.1152 0.0061 0.0253 ...
##  $ Vegetable.Oils         : num  0.534 0.326 1.031 0.646 0.81 ...
##  $ Vegetables             : num  6.76 11.78 11.65 2.3 5.45 ...
##  $ Vegetal.Products       : num  40.6 31.2 40.4 45.1 33.3 ...
##  $ Obesity                : num  4.5 22.3 26.6 6.8 19.1 28.5 20.9 30.4 21.9 19.9 ...
##  $ HighRecovery           : num  0 1 0 0 1 0 1 1 1 1 ...
```

```r
# Check top 10 and bottom 10 Rows of the Dataset
head(data_kg2,5)
```

```
##   Alcoholic.Beverages Animal.fats Animal.Products Aquatic.Products..Other
## 1              0.0014      0.1973          9.4341                       0
## 2              1.6719      0.1357         18.7684                       0
## 3              0.2711      0.0282          9.6334                       0
## 4              5.8087      0.0560          4.9278                       0
## 5              3.5764      0.0087         16.6613                       0
##   Cereals...Excluding.Beer   Eggs Fish..Seafood Fruits...Excluding.Wine   Meat
## 1                  24.8097 0.2099        0.0350                  5.3495 1.2020
## 2                   5.7817 0.5815        0.2126                  6.7861 1.8845
## 3                  13.6816 0.5277        0.2416                  6.3801 1.1305
## 4                   9.1085 0.0587        1.7707                  6.0005 2.0571
## 5                   5.9960 0.2274        4.1489                 10.7451 5.6888
##   Milk...Excluding.Butter Miscellaneous Offals Oilcrops Pulses Spices
## 1                  7.5828        0.0728 0.2057   0.0700 0.2953 0.0574
## 2                 15.7213        0.1123 0.2324   0.9377 0.2380 0.0008
## 3                  7.6189        0.1671 0.0870   0.3493 0.4783 0.0557
## 4                  0.8311        0.1165 0.1550   0.4186 0.6507 0.0009
## 5                  6.3663        0.7139 0.2219   0.2172 0.1840 0.1524
##   Starchy.Roots Stimulants Sugar...Sweeteners Sugar.Crops Treenuts
## 1        0.8802     0.3078             1.3489           0   0.0770
## 2        1.8096     0.1055             1.5367           0   0.1515
## 3        4.1340     0.2216             1.8342           0   0.1152
## 4       18.1102     0.0508             1.8495           0   0.0061
## 5        1.4522     0.1564             3.8749           0   0.0253
##   Vegetable.Oils Vegetables Vegetal.Products Obesity HighRecovery
## 1         0.5345     6.7642          40.5645     4.5            0
## 2         0.3261    11.7753          31.2304    22.3            1
## 3         1.0310    11.6484          40.3651    26.6            0
## 4         0.6463     2.3041          45.0722     6.8            0
## 5         0.8102     5.4495          33.3233    19.1            1
```

```r
# Check for Missing Values
colSums(is.na(data_kg2))
```

```
##      Alcoholic.Beverages              Animal.fats          Animal.Products
##                        0                        0                        0
##  Aquatic.Products..Other Cereals...Excluding.Beer                     Eggs
##                        0                        0                        0
##            Fish..Seafood  Fruits...Excluding.Wine                     Meat
##                        0                        0                        0
##  Milk...Excluding.Butter            Miscellaneous                   Offals
##                        0                        0                        0
##                 Oilcrops                   Pulses                   Spices
##                        0                        0                        0
##            Starchy.Roots               Stimulants       Sugar...Sweeteners
```

```
##                   0                       0                       0
##          Sugar.Crops                Treenuts          Vegetable.Oils
##                   0                       0                       0
##           Vegetables        Vegetal.Products                 Obesity
##                   0                       0                       3
##         HighRecovery
##                  10
```

```r
table(HighRecovery)
```

```
## HighRecovery
##  0  1
## 96 64
```

```r
prop.table(table(HighRecovery))
```

```
## HighRecovery
##   0   1
## 0.6 0.4
```

```r
data_kg3<-na.omit(data_kg2)
```

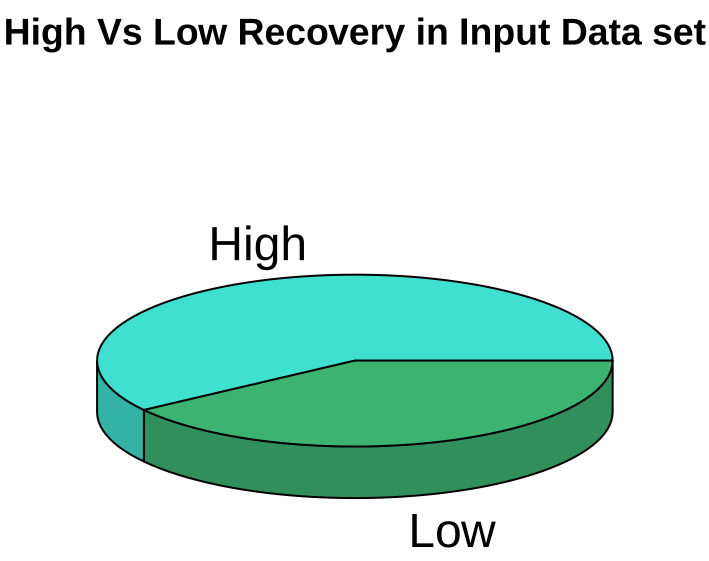
Key observations: Number of Rows and Columns:

The number of rows in the dataset is 170 The number of columns (Features) in the dataset is 25

Missing values check: > 10 Missing values present in the dataset.

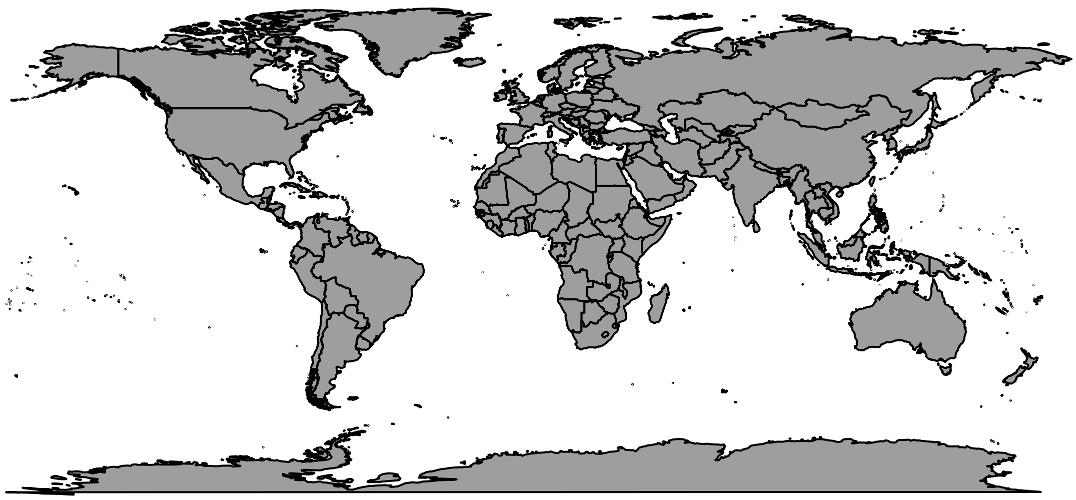
Proportion of Responders Vs Non Responders: > Total Low Recovery(less than 1%): 96 (60%) > Total High Recovery(more than 1%): 64 (40%)

```r
library(plotrix)

pie3D(prop.table((table(data_kg3$HighRecovery))),
      main='High Vs Low Recovery in Input Data set',
      #explode=0.1,
      labels=c("High", "Low"),
      col = c("Turquoise", "Medium Sea Green")
      )
```



Countries included

```r
map('world',data_kg1$Country, fill = T, col=8)
```



Summary statistics

```r
summary(data_kg3)
```

```
##  Alcoholic.Beverages  Animal.fats      Animal.Products  Aquatic.Products..Other
##  Min.   : 0.0000     Min.   :0.00180   Min.   : 1.739   Min.   :0.00000
##  1st Qu.: 0.9794     1st Qu.:0.04025   1st Qu.: 7.325   1st Qu.:0.00000
##  Median : 3.0805     Median :0.11850   Median :12.280   Median :0.00000
##  Mean   : 3.1042     Mean   :0.22840   Mean   :12.346   Mean   :0.01329
##  3rd Qu.: 4.7612     3rd Qu.:0.26535   3rd Qu.:16.704   3rd Qu.:0.00110
##  Max.   :15.3706     Max.   :1.35590   Max.   :26.887   Max.   :1.67940
##  Cereals...Excluding.Beer       Eggs         Fish..Seafood
##  Min.   : 3.401           Min.   :0.0239   Min.   :0.0342
##  1st Qu.: 7.137           1st Qu.:0.1954   1st Qu.:0.5442
##  Median :10.251           Median :0.4601   Median :1.0050
##  Mean   :11.820           Mean   :0.4748   Mean   :1.3145
##  3rd Qu.:15.140           3rd Qu.:0.6466   3rd Qu.:1.7335
##  Max.   :29.805           Max.   :1.6960   Max.   :8.7959
##  Fruits...Excluding.Wine       Meat          Milk...Excluding.Butter
##  Min.   : 0.6596          Min.   :0.356   Min.   : 0.0963
##  1st Qu.: 3.5233          1st Qu.:1.878   1st Qu.: 2.1970
##  Median : 4.9733          Median :3.399   Median : 5.8598
##  Mean   : 5.6316          Mean   :3.332   Mean   : 6.7875
##  3rd Qu.: 6.8392          3rd Qu.:4.378   3rd Qu.:10.9448
##  Max.   :19.3028          Max.   :8.170   Max.   :20.8378
##  Miscellaneous          Offals           Oilcrops           Pulses
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0098   Min.   :0.0010
##  1st Qu.:0.02865   1st Qu.:0.1054   1st Qu.:0.1330   1st Qu.:0.1434
##  Median :0.17880   Median :0.1672   Median :0.3113   Median :0.3109
##  Mean   :0.40906   Mean   :0.1957   Mean   :0.5872   Mean   :0.5437
##  3rd Qu.:0.56255   3rd Qu.:0.2280   3rd Qu.:0.6203   3rd Qu.:0.7185
##  Max.   :3.66340   Max.   :1.2256   Max.   :9.9865   Max.   :3.4838
##      Spices          Starchy.Roots      Stimulants      Sugar...Sweeteners
##  Min.   :0.00000   Min.   : 0.6796   Min.   :0.0042   Min.   :0.3666
```

```
##  1st Qu.:0.01700   1st Qu.: 2.0086   1st Qu.:0.0846   1st Qu.:1.7328
##  Median :0.04290   Median : 3.0975   Median :0.1677   Median :2.6110
##  Mean   :0.09331   Mean   : 5.2264   Mean   :0.2090   Mean   :2.8486
##  3rd Qu.:0.12585   3rd Qu.: 5.4066   3rd Qu.:0.2690   3rd Qu.:3.8468
##  Max.   :0.66260   Max.   :27.7128   Max.   :1.2823   Max.   :9.7259
##   Sugar.Crops         Treenuts       Vegetable.Oils     Vegetables
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0915   Min.   : 0.857
##  1st Qu.:0.00000   1st Qu.:0.0215   1st Qu.:0.5088   1st Qu.: 3.624
##  Median :0.00000   Median :0.0812   Median :0.7816   Median : 5.054
##  Mean   :0.09639   Mean   :0.1186   Mean   :0.8575   Mean   : 6.111
##  3rd Qu.:0.00000   3rd Qu.:0.1474   3rd Qu.:1.0772   3rd Qu.: 7.789
##  Max.   :3.06770   Max.   :0.7569   Max.   :2.2026   Max.   :19.299
##  Vegetal.Products    Obesity       HighRecovery
##  Min.   :23.11    Min.   : 2.10   Min.   :0.0000
##  1st Qu.:33.29    1st Qu.: 8.50   1st Qu.:0.0000
##  Median :37.72    Median :21.30   Median :0.0000
##  Mean   :37.65    Mean   :18.52   Mean   :0.4025
##  3rd Qu.:42.67    3rd Qu.:25.70   3rd Qu.:1.0000
##  Max.   :48.26    Max.   :37.30   Max.   :1.0000
```

Data Partition

Creating Training and Testing Dataset The given data set is divided into Training and Testing data set, with 80:20 proportion. The distribution of High Recovery and Low Recovery Class is verified in both the data sets, and ensured it's close to equal.

```
set.seed(123)
split = sample.split(data_kg3$HighRecovery,SplitRatio = 0.8)
dataTrain = subset(data_kg3, split == TRUE)
dataTest = subset(data_kg3, split == FALSE)
dim(dataTrain)
```

```
## [1] 127  25
```

```
dim(dataTest)
```

```
## [1] 32 25
```

Check if distribution of partition data is correct

```
table(dataTrain$HighRecovery)
```

```
##
##  0  1
## 76 51
```

```
table(dataTest$HighRecovery)
```

```
##
##  0  1
## 19 13
```

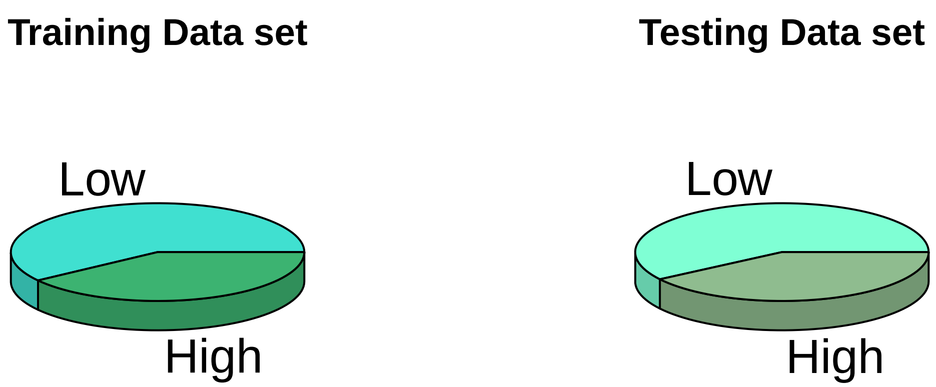
```
prop.table((table(dataTrain$HighRecovery)))
```

```
##
##         0         1
## 0.5984252 0.4015748
```

```
prop.table((table(dataTest$HighRecovery)))
```

```
##
##       0       1
## 0.59375 0.40625
```

```
par(mfrow=c(1,2))

pie3D(prop.table((table(dataTrain$HighRecovery))),
      main='Training Data set',
      #explode=0.1,
      labels=c("Low", "High"),
      col = c("Turquoise", "Medium Sea Green")
      )
pie3D(prop.table((table(dataTest$HighRecovery))),
      main='Testing Data set',
      #explode=0.1,
      labels=c("Low", "High"),
      col = c("Aquamarine", "Dark Sea Green")
      )
```



Model Building - CART

Decision Trees are commonly used in data mining with the objective of creating a model that predicts the value of a target (or dependent variable) based on the values of several input (or independent variables).

As an Umbrella term, the Classification and Regression Tree refers to the following decision trees:

> Classification Trees: where the target variable is categorical and the tree is used to identify the "class" within which a target variable would likely fall into.
>
> Regression Trees: where the target variable is continuous and tree is used to predict its value.

The CART algorithm is structured as a sequence of questions, the answers to which determine what the next question, if any should be. The result of these questions is a tree like structure where the ends are terminal nodes at which point there are no more questions.

Model Building - CART

Setting the control parameter inputs for rpart

```
r.ctrl <- rpart.control(minsplit = 5,
                        minbucket = 5,
```

```
                        cp = 0,
                        xval = 5
                        )
```

Build the model on Training Dataset

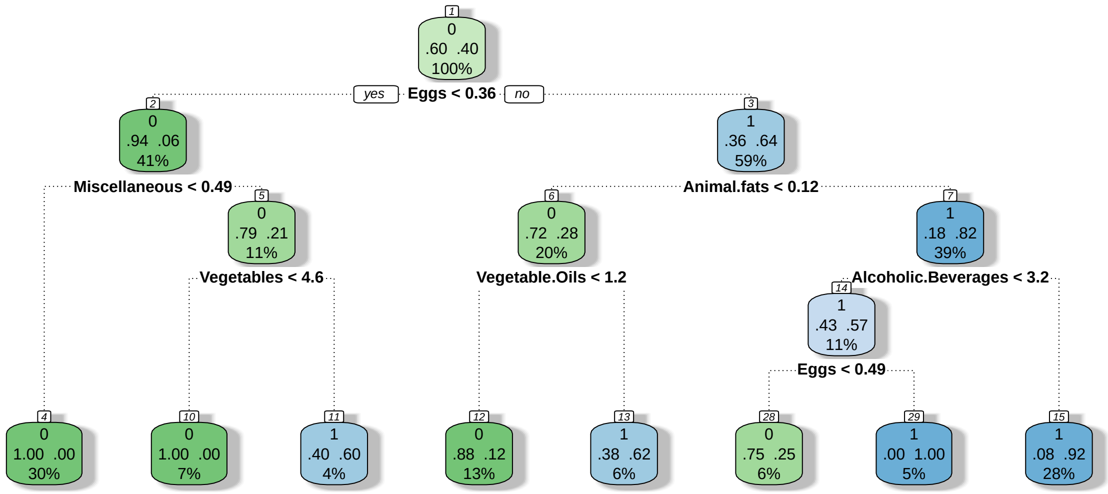
```
#Exclude columns - "Customer ID" and "Acct Opening Date"
cart.train <- dataTrain
names(cart.train)
```

```
##  [1] "Alcoholic.Beverages"      "Animal.fats"
##  [3] "Animal.Products"          "Aquatic.Products..Other"
##  [5] "Cereals...Excluding.Beer" "Eggs"
##  [7] "Fish..Seafood"            "Fruits...Excluding.Wine"
##  [9] "Meat"                     "Milk...Excluding.Butter"
## [11] "Miscellaneous"            "Offals"
## [13] "Oilcrops"                 "Pulses"
## [15] "Spices"                   "Starchy.Roots"
## [17] "Stimulants"               "Sugar...Sweeteners"
## [19] "Sugar.Crops"              "Treenuts"
## [21] "Vegetable.Oils"           "Vegetables"
## [23] "Vegetal.Products"         "Obesity"
## [25] "HighRecovery"
```

```
m1 <- rpart(formula = HighRecovery~.,
            data = cart.train,
            method = "class",
            control = r.ctrl
            )

library(rattle)
```

```
## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

##
## Attaching package: 'rattle'

## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(RColorBrewer)

fancyRpartPlot(m1)
```

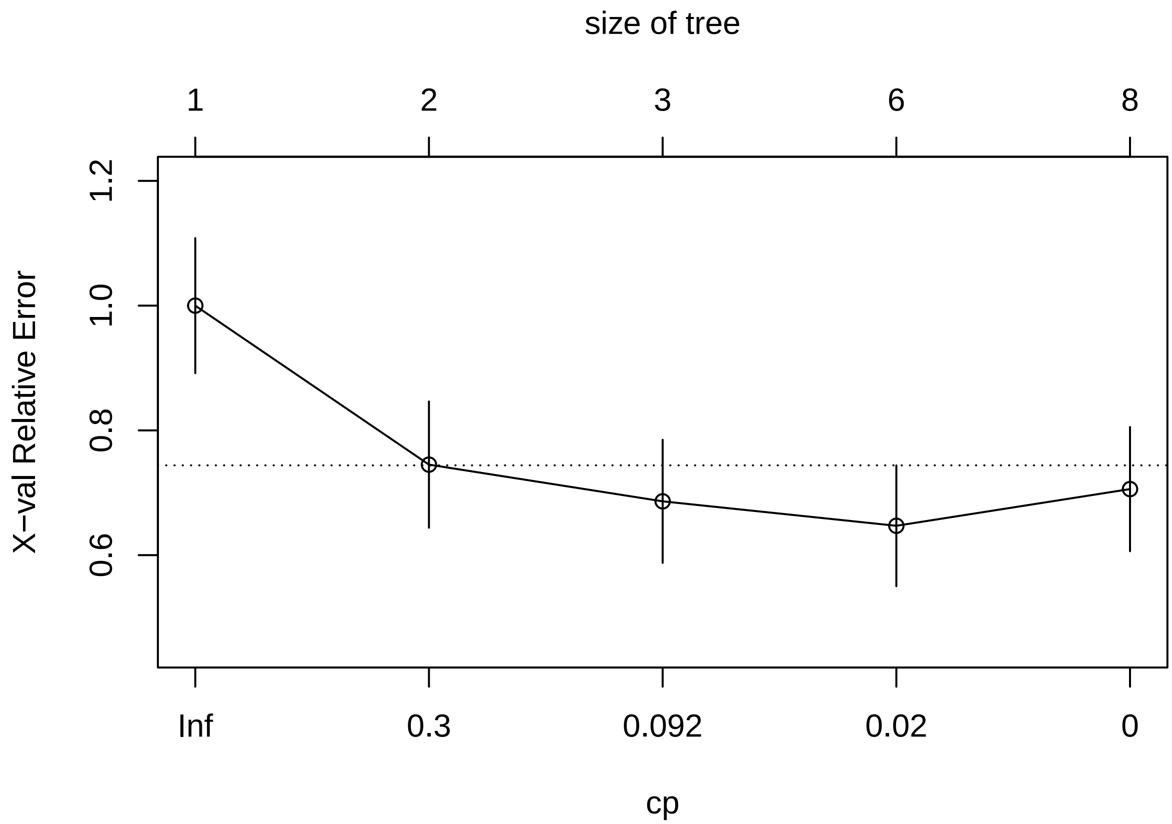


Rattle 2022−Apr−04 02:01:14 panyl

```
printcp(m1)
```

```
##
## Classification tree:
## rpart(formula = HighRecovery ~ ., data = cart.train, method = "class",
##     control = r.ctrl)
##
## Variables actually used in tree construction:
## [1] Alcoholic.Beverages Animal.fats         Eggs
## [4] Miscellaneous       Vegetable.Oils      Vegetables
##
## Root node error: 51/127 = 0.40157
##
## n= 127
##
##          CP nsplit rel error  xerror     xstd
## 1 0.4117647      0   1.00000 1.00000 0.108323
## 2 0.2156863      1   0.58824 0.74510 0.101185
## 3 0.0392157      2   0.37255 0.68627 0.098732
## 4 0.0098039      5   0.25490 0.64706 0.096906
## 5 0.0000000      7   0.23529 0.70588 0.099586
```

```
plotcp(m1)
```



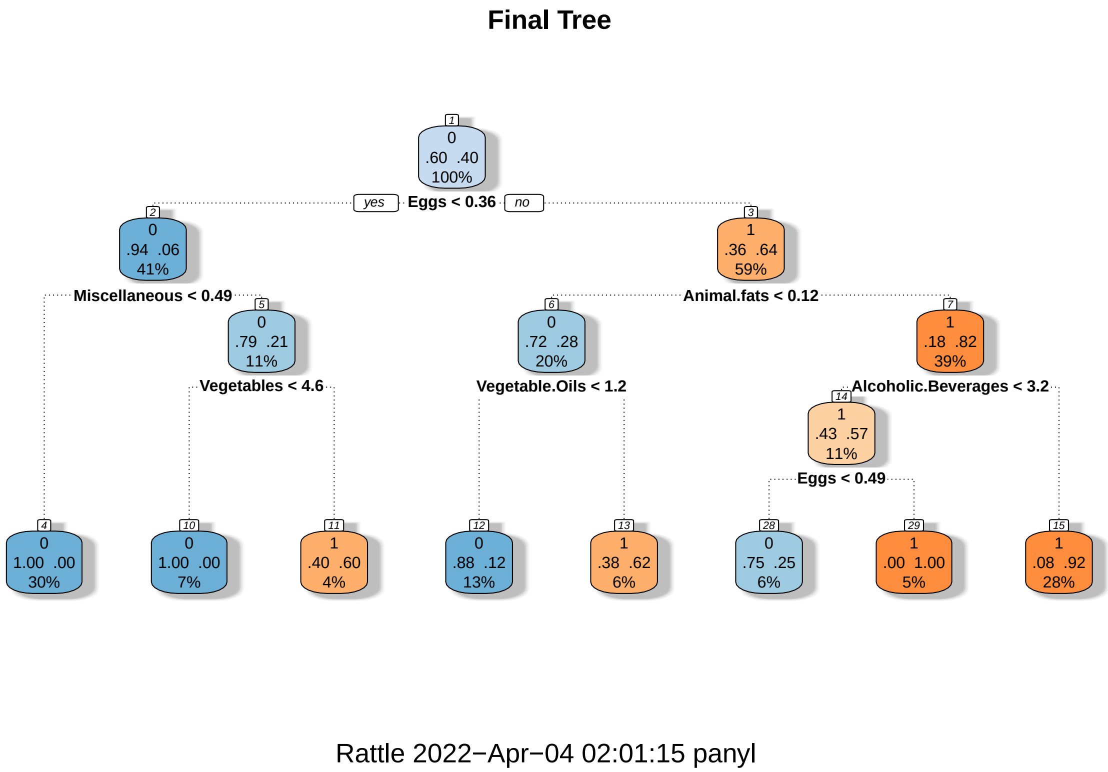

```
ptree<- prune(m1, cp= 0.0017 ,"CP")
printcp(ptree)
```

```
##
## Classification tree:
## rpart(formula = HighRecovery ~ ., data = cart.train, method = "class",
##     control = r.ctrl)
##
## Variables actually used in tree construction:
## [1] Alcoholic.Beverages Animal.fats         Eggs
## [4] Miscellaneous       Vegetable.Oils      Vegetables
##
## Root node error: 51/127 = 0.40157
##
## n= 127
##
##          CP nsplit rel error  xerror     xstd
## 1 0.4117647      0   1.00000 1.00000 0.108323
## 2 0.2156863      1   0.58824 0.74510 0.101185
## 3 0.0392157      2   0.37255 0.68627 0.098732
## 4 0.0098039      5   0.25490 0.64706 0.096906
## 5 0.0000000      7   0.23529 0.70588 0.099586
```

Ploting the final CART model

```
fancyRpartPlot(ptree,
               uniform = TRUE,
               main = "Final Tree",
```

```
                palettes = c("Blues", "Oranges")
                )
```



Performance Measures on Training Data Set The following model performance measures will be calculated on the training set to gauge the goodness of the model:

KS

Area Under Curve (AUC)

Gini Coefficient

Classification Error

Predict Training Data Set

```
cart.train$predict.class = predict(ptree, cart.train, type = "class")
cart.train$predict.score = predict(ptree, cart.train, type = "prob")
```

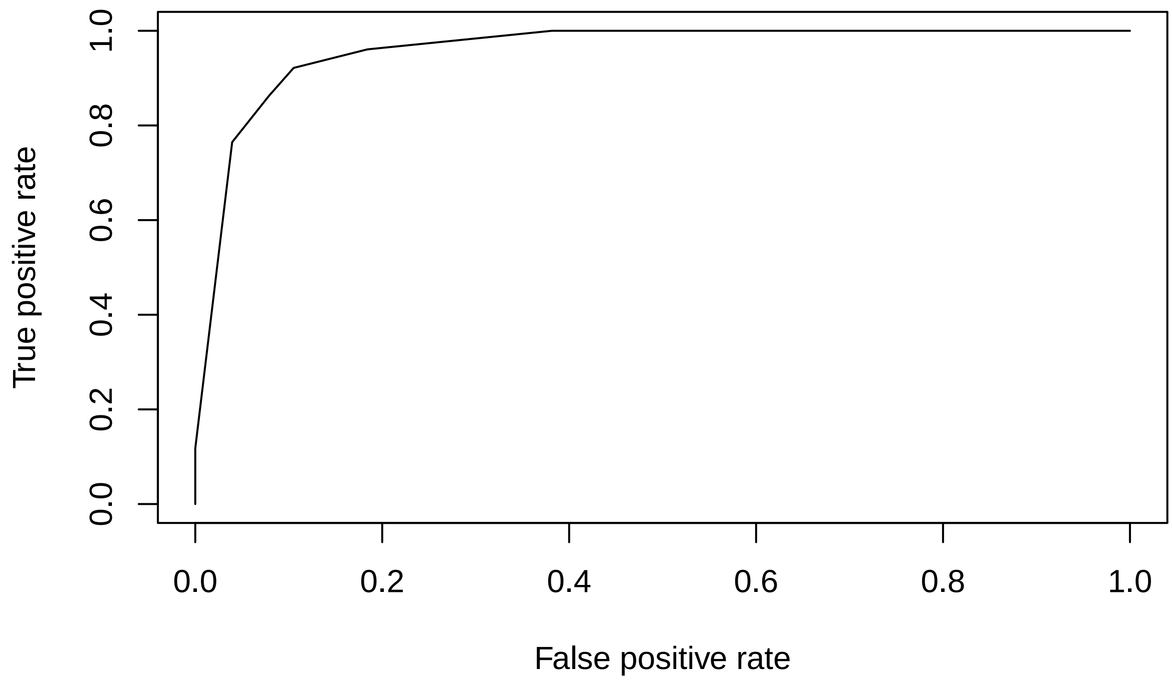
KS and Area under Curve

```
library(ROCR)
library(ineq)
pred <- prediction(cart.train$predict.score[,2], cart.train$HighRecovery)
perf <- performance(pred, "tpr", "fpr")

KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])

auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
```

```r
gini = ineq(cart.train$predict.score[,2], type="Gini")
with(cart.train, table(HighRecovery, predict.class))
```

```
##              predict.class
## HighRecovery  0  1
##             0 68  8
##             1  4 47
```

```r
plot(perf)
```




```r
KS
```

```
## [1] 0.8163055
```

```r
auc
```

```
## [1] 0.9592363
```

```r
gini
```

```
## [1] 0.5496372
```

Summary:CART - Model Performance(Training Dataset):

The KS = 81.6% and the AUC = 95.9% which indicates that the model is very good.

The Gini Coefficient = 54.9% also indicates that the model is good.

Confusion matrix:

1. Accuracy = (68 + 47)/(68+47+8+4) = 90.56%
2. Classification Error Rate = 1 - Accuracy = 9.44%

Predict Test Data Set

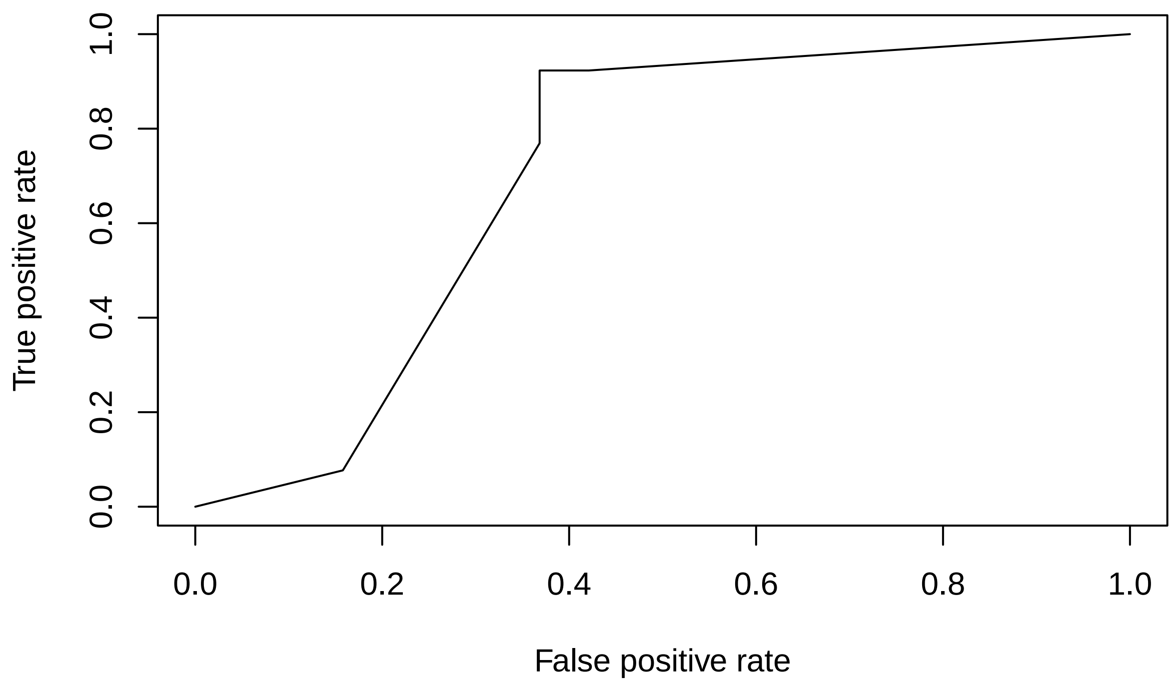
```r
## Scoring Holdout sample
cart.test <- dataTest
cart.test$predict.class = predict(ptree, cart.test, type = "class")
cart.test$predict.score = predict(ptree, cart.test, type = "prob")
#head(cart.test)
```

KS and Area under Curve

```r
pred <- prediction(cart.test$predict.score[,2], cart.test$HighRecovery)
perf <- performance(pred, "tpr", "fpr")
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
gini = ineq(cart.test$predict.score[,2], type="Gini")
with(cart.test, table(HighRecovery, predict.class))
```

```
##              predict.class
## HighRecovery  0  1
##             0 12  7
##             1  1 12
```

```r
plot(perf)
```




```r
KS
```

```
## [1] 0.5546559
```

```r
auc
```

```
## [1] 0.7004049
```

```r
gini
```

```
## [1] 0.4298129
```

CART - Model Performance(Test Dataset):

The KS = 55.5% and the AUC = 70.0% which indicates that the model is good.

The Gini Coefficient = 42.98% also indicates that the model in good.

Confusion matrix:

1. Accuracy = (12 + 12)/(12+7+1+12) = 75%
2. Classification Error Rate = 1 - Accuracy = 25%

CART - Conclusion

Comparative Summary of the CART Model on Training and Testing Dataset is as follows:

Measures	Train	Test	%Deviation
KS	81.60	55.50	32.0
AUC	95.90	70.00	27.0
Gini	54.90	42.98	21.7
Accuracy	90.56	75.00	17.2
CeR	9.44	25.00	62.0

CeR= Clasification error rate

The good performance on the model performance measures indicates good prediction making capabilities of the developed CART model.

Model Building - Random Forest

A Supervised Classification Algorithm, as the name suggests, this algorithm creates the forest with a number of trees in random order. In general, the more trees in the forest the more robust the forest looks like. In the same way in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results.

Some advantages of using Random Forest are as follows: > The same random forest algorithm or the random forest classifier can use for both classification and the regression task.

> Random forest classifier will handle the missing values.
>
> When we have more trees in the forest, random forest classifier won't over fit the model.
>
> Can model the random forest classifier for categorical values also.

Creating Training and Testing Dataset for RF Model

```
rf.train <- dataTrain
rf.test  <- dataTest

dim(rf.train)
```

```
## [1] 127  25
```

```
dim(rf.test)
```

```
## [1] 32 25
```

```r
library(randomForest)
```

Random Forest Model - Train Dataset

The model is built with dependant variable as HighRecovery, and considering all independent variables.

```r
RF=randomForest(as.factor(HighRecovery)~.,
                data = rf.train,
                ntree = 501, mtry = 3, nodesize = 10,
                importance=TRUE)
print(RF)
```

```
##
## Call:
##  randomForest(formula = as.factor(HighRecovery) ~ ., data = rf.train,      ntree = 501, mtry = 3, no
##               Type of random forest: classification
##                     Number of trees: 501
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 19.69%
## Confusion matrix:
##    0  1 class.error
## 0 67  9   0.1184211
## 1 16 35   0.3137255
```

Out of Bag Error Rate:

Random Forests algorithm is a classifier based on primarily two methods - bagging and random subspace method.

Suppose we decide to have S number of trees in our forest then we first create S datasets of "same size as original" created from random resampling of data with-replacement. Each of these datasets is called a bootstrap dataset.

Due to "with-replacement" option, every dataset can have duplicate data records and at the same time, can be missing several data records from original datasets. This is called Bagging.

The algorithm uses m (=sqrt(M)) random sub features out of M possible features to create any tree. This is called random subspace method.

After creating the classifiers (S trees), there is a subset of records which does not include any of the records part of the classifier tree. This subset, is a set of boostrap datasets which does not contain a particular record from the original dataset. This set is called out-of-bag examples. There are n such subsets (one for each data record in original dataset T). OOB classifier is the aggregation of all such records.

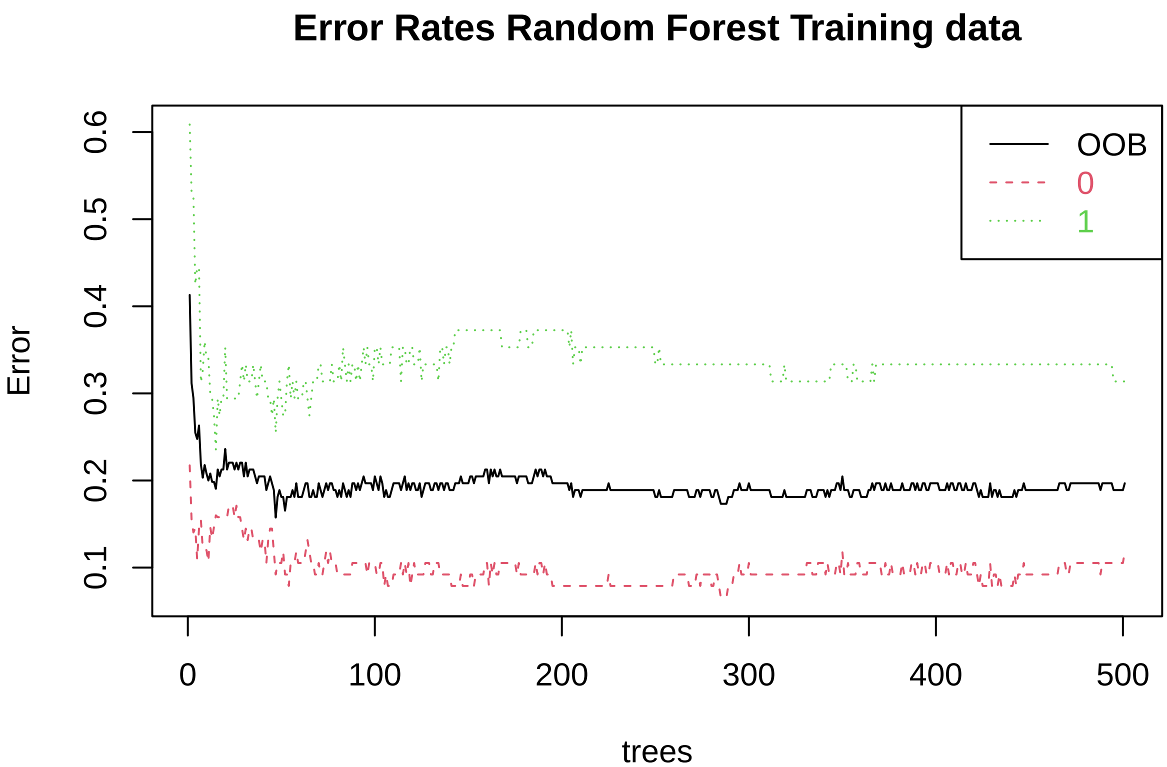
Out-of-bag estimate for the generalization error is the error rate of the outof-bag classifier on the training set (compare it with known yi's).

Out-of-bag (OOB) error, also called out-of-bag estimate, is a method of measuring the prediction error of random forests, boosted decision trees, and other machine learning models utilizing bootstrap aggregating to subsample data samples used for training.

Out-of-bag estimates help in avoiding the need for an independent validation dataset.

The graphical output for the OOB estimate of error rate.

```r
#dev.off()
plot(RF, main="")
legend("topright", c("OOB", "0", "1"), text.col=1:6, lty=1:3, col=1:3)
title(main="Error Rates Random Forest Training data")
```



The output in tabular form for the OOB estimate of error rate.

It is observed that as the number of tress increases, the OOB error rate starts decreasing till it reaches around 80th tree with OOB = 0.1757 (the minimum value). After this, the OOB doesn't decrease further and remains around steady. Hence, the optimal number of trees would be around 80.

Variable Importance

To understand the important variables in Random Forest, the following measures are generally used: > Mean Decrease in Accuracy is based on permutation >> Randomly permute values of a variable for which importance is to be computed in the OOB sample >> Compute the Error Rate with permuted values >> Compute decrease in OOB Error rate (Permuted - Not permuted) >> Average the decrease over all the trees > Mean Decrease in Gini is computed as "total decrease in node impurities from splitting on the variable, averaged over all trees"

```r
#List the iimportance of the variable
impVar <- round(randomForest::importance(RF), 2)
impVar[order(impVar[,3], decreasing=TRUE),]
```

```
##                              0     1 MeanDecreaseAccuracy MeanDecreaseGini
## Eggs                      6.46 10.93                11.22             4.65
## Obesity                   2.72 10.85                10.24             4.02
## Vegetal.Products          6.45  8.05                 9.46             4.21
## Cereals...Excluding.Beer  6.72  6.94                 9.28             3.74
## Animal.Products           5.29  7.83                 9.05             4.01
```

```
## Animal.fats               4.65  7.45                 8.45              3.46
## Meat                      3.29  7.82                 7.81              2.94
## Pulses                    2.53  6.91                 6.81              1.52
## Alcoholic.Beverages       3.16  6.91                 6.76              2.42
## Vegetables                0.97  7.41                 6.22              2.27
## Milk...Excluding.Butter   2.08  3.77                 4.60              1.80
## Treenuts                  1.54  4.93                 4.23              1.27
## Oilcrops                  2.33  3.42                 4.20              1.54
## Stimulants                1.27  4.35                 4.16              1.17
## Sugar.Crops               0.79  3.23                 2.93              0.38
## Sugar...Sweeteners       -0.30  3.82                 2.84              0.94
## Fish..Seafood            -0.21  3.81                 2.73              1.07
## Starchy.Roots             0.26  3.23                 2.06              0.90
## Aquatic.Products..Other   0.67  1.24                 1.51              0.97
## Spices                   -1.38  2.76                 1.26              0.76
## Offals                    1.65 -1.05                 0.45              0.84
## Miscellaneous             1.93 -2.41                -0.24              0.92
## Fruits...Excluding.Wine  -2.68  2.30                -0.50              0.93
## Vegetable.Oils           -1.06 -0.13                -0.56              0.42
```

Optimal mtry value

In the random forests the number of variables available for splitting at each tree node is referred to as the mtry parameter. The optimum number of variables is obtained using tuneRF function. x = Predictor variables y = Target variable mtryStart = starting value of mtry ntree = No of tree used for tuning stepFactor = steps to increase (deflate) mtry improve = the relative oob by atleast this much trace = print the trace or not plot = plot OOB vs mtry graph or not doBest = Finally build the RF using optimal mtry nodesize = min terminal node size importance = compute variable importance or not

```r
#Tuning Random Forest
# tRF<- tuneRF(x = rf.train,
#              y=as.factor(rf.train$HighRecovery),
#              mtryStart = 5, #Aprox, Sqrt of Total no. of variables
#              ntreeTry = 100,
#              stepFactor = 1,
#              improve = 0.0001,
#              trace = TRUE,
#              plot = TRUE,
#              doBest = TRUE,
#              nodesize = 10,
#              importance = TRUE
# )
tRF <- train(
  as.factor(HighRecovery) ~., data =rf.train, method = "rpart",
  trControl = trainControl("cv", number = 10),
  tuneLength = 100)
tRF$finalModel
```

```
## n= 127
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 127 51 0 (0.59842520 0.40157480)
##   2) Eggs< 0.3634 52  3 0 (0.94230769 0.05769231) *
```

```
##   3) Eggs>=0.3634 75 27 1 (0.36000000 0.64000000)
##     6) Animal.fats< 0.1239 25  7 0 (0.72000000 0.28000000) *
##     7) Animal.fats>=0.1239 50  9 1 (0.18000000 0.82000000) *
```

KS and Area under Curve

```
rf.train$predict.class <- predict(tRF$finalModel, rf.train, type = "class")
rf.train$predict.score <- predict(tRF$finalModel, rf.train, type = "prob")
#head(rf.train)
#class(rf.train$predict.score)
library(ROCR)
pred <- prediction(rf.train$predict.score[,2], rf.train$HighRecovery)
perf <- performance(pred, "tpr", "fpr")
#plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
KS
```

```
## [1] 0.6855005
```

```
# Area Under Curve
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
auc
```

```
## [1] 0.880031
```

```
# Gini Coefficient
library(ineq)
gini = ineq(rf.train$predict.score[,2], type="Gini")
gini
```

```
## [1] 0.4548402
```

```
# Classification Error
with(rf.train, table(HighRecovery, predict.class))
```

```
##             predict.class
## HighRecovery  0  1
##            0 67  9
##            1 10 41
```

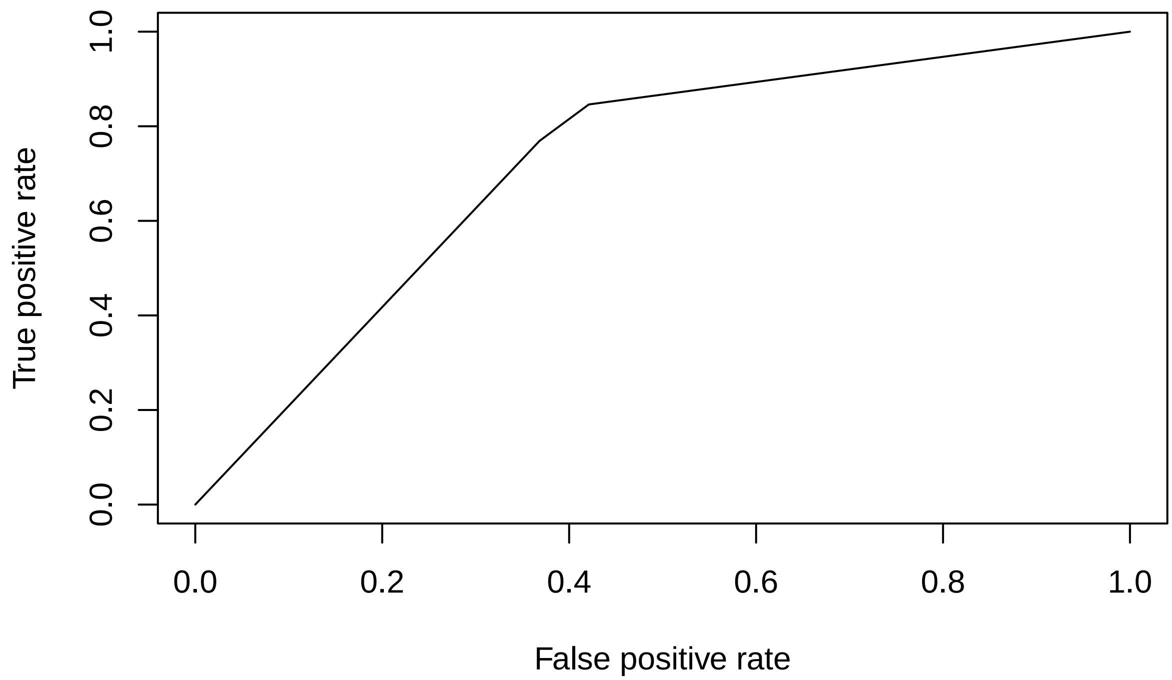
KS=68.6% AUC=88.0% Gini=45.5% Accuracy=85.03% CeR=14.96%

Model Performance on Testing Data Set

```
rf.test$predict.class <- predict(tRF$finalModel, rf.test, type="class")
rf.test$predict.score <- predict(tRF$finalModel, rf.test, type="prob")
```

KS and AUC

```
pred1 <- prediction(rf.test$predict.score[,2], rf.test$HighRecovery)
perf1 <- performance(pred1, "tpr", "fpr")
plot(perf1)
```




```
KS1 <- max(attr(perf1, 'y.values')[[1]]-attr(perf1, 'x.values')[[1]])
KS1
```

```
## [1] 0.4251012
```

```
# Area Under Curve
auc1 <- performance(pred1,"auc");
auc1 <- as.numeric(auc1@y.values)
auc1
```

```
## [1] 0.7186235
```

```
# Gini Coefficient
library(ineq)
gini1 = ineq(rf.test$predict.score[,2], type="Gini")
gini1
```

```
## [1] 0.3946926
```

```
# Classification Error
with(rf.test, table(HighRecovery, predict.class))
```

```
##               predict.class
## HighRecovery  0  1
##             0 12  7
##             1  3 10
```

KS = 42.5% AUC = 71.9% Gini = 39.5% Accuracy = 68.8% CeR = 31.2%

Random Forest Conclusion

```
library('knitr')
conclusion<-data.frame("Measures"<-c('KS','AUC','Gini','Accuracy','CeR'),
           'Train'<-c(68.6,88.0,45.5,85.03,14.96),
           'Test'<-c(42.5,71.9,39.5,68.8,31.2),
```

```
           '%Deviation'<-c(38.0,18.3,13.2,19.1,52.1)
           )
names(conclusion)[1]='Measures'
names(conclusion)[2]='Train'
names(conclusion)[3]='Test'
names(conclusion)[4]='%Deviation'
kable(conclusion)
```

Measures	Train	Test	%Deviation
KS	68.60	42.5	38.0
AUC	88.00	71.9	18.3
Gini	45.50	39.5	13.2
Accuracy	85.03	68.8	19.1
CeR	14.96	31.2	52.1

Model Comparision

The main objective of the project was to develop a predictive model to predict if a country suffering COVID-19 will have a high recovery using tools of Machine Learning. In this context, the key parameter for model evaluation was 'Accuracy', i.e., the proportion of the total number of predictions that were correct.

The predictive models was be developed using the following Machine Learning techniques: > Classification Tree - CART > Random Forest

The snap shot of the performance of all the models on accuracy, over-fitting and other model performance measures is provided below:

CART

```
library('knitr')
conclusion<-data.frame("Measures"<-c('KS','AUC','Gini','Accuracy','CeR'),
           "Train"<-c(81.6,95.9,54.9,90.56,9.44),
           Test<-c(55.5,70.0,42.98,75,25),
           '%Deviation'<-c(32.0,27.0,21.7,17.2,62))
names(conclusion)[1]='Measures'
names(conclusion)[2]='Train'
names(conclusion)[3]='Test'
names(conclusion)[4]='%Deviation'
kable(conclusion)
```

Measures	Train	Test	%Deviation
KS	81.60	55.50	32.0
AUC	95.90	70.00	27.0
Gini	54.90	42.98	21.7
Accuracy	90.56	75.00	17.2
CeR	9.44	25.00	62.0

Random Forest

```
library('knitr')
RF_conclusion<-data.frame("Measures"<-c('KS','AUC','Gini','Accuracy','CeR'),
```

```r
            'Train'<-c(68.6,88.0,45.5,85.03,14.96),
            'Test'<-c(42.5,71.9,39.5,68.8,31.2),
            '%Deviation'<-c(38.0,18.3,13.2,19.1,52.1)
            )
names(RF_conclusion)[1]='Measures'
names(RF_conclusion)[2]='Train'
names(RF_conclusion)[3]='Test'
names(RF_conclusion)[4]='%Deviation'
kable(RF_conclusion)
```

Measures	Train	Test	%Deviation
KS	68.60	42.5	38.0
AUC	88.00	71.9	18.3
Gini	45.50	39.5	13.2
Accuracy	85.03	68.8	19.1
CeR	14.96	31.2	52.1

Interpretation:

The Random Forest method has given poor performance compared to CART.

The CART method has the best performance (best accuracy) among all the models. The percentage deviation between Training and Testing Dataset also is reasonably under control, suggesting a robust model.

CART seems to be the overall winner because of the best accuracy % and reasonable deviations.

Conclusion

During model building and prediction using COVID-19 Healthy Diet Dataset, I find it's difficult to get a good CART or Random Forest model (Best accuracy is only about 50%) to predict confirmed rate or death rate by food supply(in kg).

However, the result model between food consuming and recovered rate is good(75% accuracy in CART model) which tells us high percentage of eggs and fishes in daily food consumed may be help us have a high recovered rate among all confirmed cases.

Appendix