ASDM Workshop Week 7 : Decision trees with R

- Charith Silva, Dr.Mo Saraee

Part 1: Decision Trees

In data mining a decision tree is a predictive model which can be used to model both classification and regression problems, in operations research decision trees refer to a hierarchical model of decisions and their consequences. The decision maker employs decision trees to identify the strategy which will most likely reach its goal. When we use a decision tree for classification tasks, it is most commonly referred as a classification tree. When it is used for regression task, it is called regression tree. In this workshop, we will practice how to implement decision trees using R.

The dataset, "**Cardiotocographic.csv**", that we will use in the workshop can be downloaded from Blackboard Week 3 folder. We will be using cardio graphic data to classify Fetal case using 14 decision variables and 1 class variable.

Data Explanation

BPM	Beat per minutes
APC	Accelerations per second
FMPS	Fetal movement per second
UCPS	Uterine contractions per second
DLPS	Light declaration per second
SDPS	Severe declaration per second
PDPS	Prolonged declaration per second
ASTV	% of abnormal short term Variability
MSTV	Mean of short term Variability
ALTV	% of abnormal long term Variability
MLTV	Mean of long term Variability
Width	Width of FHR Histogram
Min	Min Width of FHR Histogram
Max	Max Width of FHR Histogram
NSP	Fetal State Class code N=normal (1); S=Suspect (2); P=Pathologic (3)

Implementation:

We can use different packages available in R to classify our data using decision trees. In this workshop, we will be using PARTY package which use CART (classification and regression tree) algorithm to classify our data, based on the prediction and class variables.

1. Download "**Cardiotocographic.csv**" dataset from Blackboard Week 7 folder and save it to a folder on your F: drive. Open it using excel to get a rough idea about the dataset, e.g, attributes and their values.

2. Open a new R script window:
   `File → New File → R script`

3. Import the "Cardiotocographic.csv"data file and create a data frame

```
cardio_data <- read.csv("Cardiotocographic.csv", header= TRUE)
```

4. Inspect the dataset in R

Once the file has been imported to R we often want to do few things to explore the dataset:

```
names(cardio_data)
head(cardio_data)
tail(cardio_data)
summary(cardio_data)
```

```
> summary(cardio_data)
      BPM             APC                 FMPS                UCPS                DLPS                SDPS                 PDPS
 Min.   :106.0   Min.   :0.000000   Min.   :0.000000   Min.   :0.000000   Min.   :0.000000   Min.   :0.000e+00   Min.   :0.0000000
 1st Qu.:126.0   1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.001876   1st Qu.:0.000000   1st Qu.:0.000e+00   1st Qu.:0.0000000
 Median :133.0   Median :0.001630   Median :0.000000   Median :0.004482   Median :0.000000   Median :0.000e+00   Median :0.0000000
 Mean   :133.3   Mean   :0.003170   Mean   :0.009474   Mean   :0.004357   Mean   :0.001885   Mean   :3.585e-06   Mean   :0.0001566
 3rd Qu.:140.0   3rd Qu.:0.005631   3rd Qu.:0.002512   3rd Qu.:0.006525   3rd Qu.:0.003264   3rd Qu.:0.000e+00   3rd Qu.:0.0000000
 Max.   :160.0   Max.   :0.019284   Max.   :0.480634   Max.   :0.014925   Max.   :0.015385   Max.   :1.353e-03   Max.   :0.0053476
      ASTV            MSTV            ALTV             MLTV            Width             Min              Max             NSP
 Min.   :12.00   Min.   :0.200   Min.   : 0.000   Min.   : 0.000   Min.   :  3.00   Min.   : 50.00   Min.   :122   Min.   :1.000
 1st Qu.:32.00   1st Qu.:0.700   1st Qu.: 0.000   1st Qu.: 4.600   1st Qu.: 37.00   1st Qu.: 67.00   1st Qu.:152   1st Qu.:1.000
 Median :49.00   Median :1.200   Median : 0.000   Median : 7.400   Median : 67.50   Median : 93.00   Median :162   Median :1.000
 Mean   :46.99   Mean   :1.333   Mean   : 9.847   Mean   : 8.188   Mean   : 70.45   Mean   : 93.58   Mean   :164   Mean   :1.304
 3rd Qu.:61.00   3rd Qu.:1.700   3rd Qu.:11.000   3rd Qu.:10.800   3rd Qu.:100.00   3rd Qu.:120.00   3rd Qu.:174   3rd Qu.:1.000
 Max.   :87.00   Max.   :7.000   Max.   :91.000   Max.   :50.700   Max.   :180.00   Max.   :159.00   Max.   :238   Max.   :3.000
```

```
str(cardio_data)
```

```
> str(cardio_data)
'data.frame':   2126 obs. of  15 variables:
 $ BPM  : int  120 132 133 134 132 134 134 122 122 122 ...
 $ APC  : num  0 0.00638 0.00332 0.00256 0.00651 ...
 $ FMPS : num  0 0 0 0 0 0 0 0 0 0 ...
 $ UCPS : num  0 0.00638 0.00831 0.00768 0.00814 ...
 $ DLPS : num  0 0.00319 0.00332 0.00256 0 ...
 $ SDPS : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PDPS : num  0 0 0 0 0 ...
 $ ASTV : int  73 17 16 16 16 26 29 83 84 86 ...
 $ MSTV : num  0.5 2.1 2.1 2.4 2.4 5.9 6.3 0.5 0.5 0.3 ...
 $ ALTV : int  43 0 0 0 0 0 0 6 5 6 ...
 $ MLTV : num  2.4 10.4 13.4 23 19.9 0 0 15.6 13.6 10.6 ...
 $ Width: int  64 130 130 117 117 150 150 68 68 68 ...
 $ Min  : int  62 68 68 53 53 50 50 62 62 62 ...
 $ Max  : int  126 198 198 170 170 200 200 130 130 130 ...
 $ NSP  : int  2 1 1 1 1 3 3 3 3 3 ...
```

5. Check the dimension and number of points of the “**cardio_data**” dataset

```
nrow(cardio_data)
ncol(cardio_data)
dim(cardio_data)
```

```
> nrow(cardio_data)
[1] 2126
> ncol(cardio_data)
[1] 15
> dim(cardio_data)
[1] 2126   15
```

6. Since we need categorical (Factor) data to class variable for prediction, hence we should convert the NSP variable to categorical form by running the following command.

```
# as.factor function convert a column into a factor column.

cardio_data$NSPF <- as.factor(cardio_data$NSP)
```

7. Check whther the changes has been made.

Run `str(cardio_data)` command

```
Console D:/ASDM/
> cardio_data$NSPF <- as.factor(cardio_data$NSP)
> str(cardio_data)
'data.frame':   2126 obs. of  16 variables:
 $ BPM  : int  120 132 133 134 132 134 134 122 122 122 ...
 $ APC  : num  0 0.00638 0.00332 0.00256 0.00651 ...
 $ FMPS : num  0 0 0 0 0 0 0 0 0 0 ...
 $ UCPS : num  0 0.00638 0.00831 0.00768 0.00814 ...
 $ DLPS : num  0 0.00319 0.00332 0.00256 0 ...
 $ SDPS : num  0 0 0 0 0 0 0 0 0 0 ...
 $ PDPS : num  0 0 0 0 0 ...
 $ ASTV : int  73 17 16 16 16 26 29 83 84 86 ...
 $ MSTV : num  0.5 2.1 2.1 2.4 2.4 5.9 6.3 0.5 0.5 0.3 ...
 $ ALTV : int  43 0 0 0 0 0 0 6 5 6 ...
 $ MLTV : num  2.4 10.4 13.4 23 19.9 0 0 15.6 13.6 10.6 ...
 $ Width: int  64 130 130 117 117 150 150 68 68 68 ...
 $ Min  : int  62 68 68 53 53 50 50 62 62 62 ...
 $ Max  : int  126 198 198 170 170 200 200 130 130 130 ...
 $ NSP  : int  2 1 1 1 1 3 3 3 3 3 ...
 $ NSPF : Factor w/ 3 levels "1","2","3": 2 1 1 1 1 3 3 3 3 3 ...
```

We have created our class variable to test the heart condition based on different variables.

8. Now we will be specifying our train and validate(test) data from our data. Take small number of observations to test the model.

Now we will divide our sample into 80% Training and 20% Validation parts for implementing our trees. Run the following Commands

```
set.seed(1234)
```

```
#set.seed function in R is used to reproduce results i.e.  it produces the same sample again and again.
```

```
#sample function can be used to return a random permutation of a vector.
pd <- sample(2, nrow(cardio_data),replace=TRUE, prob=c(0.8,0.2))
```

```
pd
```

```
> pd <- sample(2,nrow(cardio_data),replace=TRUE, prob=c(0.8,0.2))
> pd
   [1] 1 2 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2
  [64] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1
 [127] 1 1 1 1 2 1 2 1 1 2 2 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 2
 [190] 1 1 2 1 1 1 1 1 2 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 2 1
 [253] 1 1 2 2 1 2 1 1 1 1 1 1 1 1 2 2 1 2 2 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 2 1 1 1 1
 [316] 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2
 [379] 2 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 2 1 2 2 1 1 1 2 1 1 1 1 1 2
 [442] 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 2 1 1 1 1
 [505] 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 2 1 2 1 1 1 1 1 1 1 1
 [568] 1 2 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 1
 [631] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 2 2 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1
 [694] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 2 2 1 1 1 1 1 1 2 1 2
 [757] 1 1 1 1 1 1 1 2 1 1 1 1 2 2 1 1 1 1 2 2 1 1 1 1 1 1 2 1 2 1 1 1 2 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2
 [820] 2 2 2 1 1 1 1 1 1 2 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [883] 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 2 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
 [946] 2 1 1 2 2 1 1 2 1 2 2 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 1 2 2 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1 2 1 2 1 1 1 1 1 1 1 1
 [ reached getOption("max.print") -- omitted 1126 entries ]
```

```
train <- cardio_data[pd==1,]
validate <- cardio_data[pd==2,]
```

9. To check how many observations are in train and validate data sets .

```
dim(train) # Retrieve the dimension of the train data set
dim(validate) # Retrieve the dimension of the validate data set
```

```
> dim(train) # Retrieve the dimension of the train data set
[1] 1685   16
> dim(validate) # Retrieve the dimension of the validate data set
[1] 441  16
```

10. Now since we have train and validate data sets, we can implement Decision trees with **party** package. We will need to install the package and load it to RStudio.

```
# install “party” package. Ignore instalation if “party” package was already installed. You can run library() to find out this.
```

```
install.packages("party")
library(party)              # activate “party” package
```

11. Train the tree using **ctree()** function in **party** package.

Note :for simplicity, we only use four variables, BPM, APC, FMPS and UCPS, to predict the NSP value.

*Usage of **ctree()** function*

```
ctree(formula, data, subset )
```

Arguments

formula	: a symbolic description of the model to be fit.
data	: a data frame containing the variables in the model.
Subset	: an optional vector specifying a subset of observations to be used in the fitting process.

```
cardio_tree <- ctree(NSPF ~ BPM + APC + FMPS + UCPS ,data = train)
cardio_tree
```

Results:

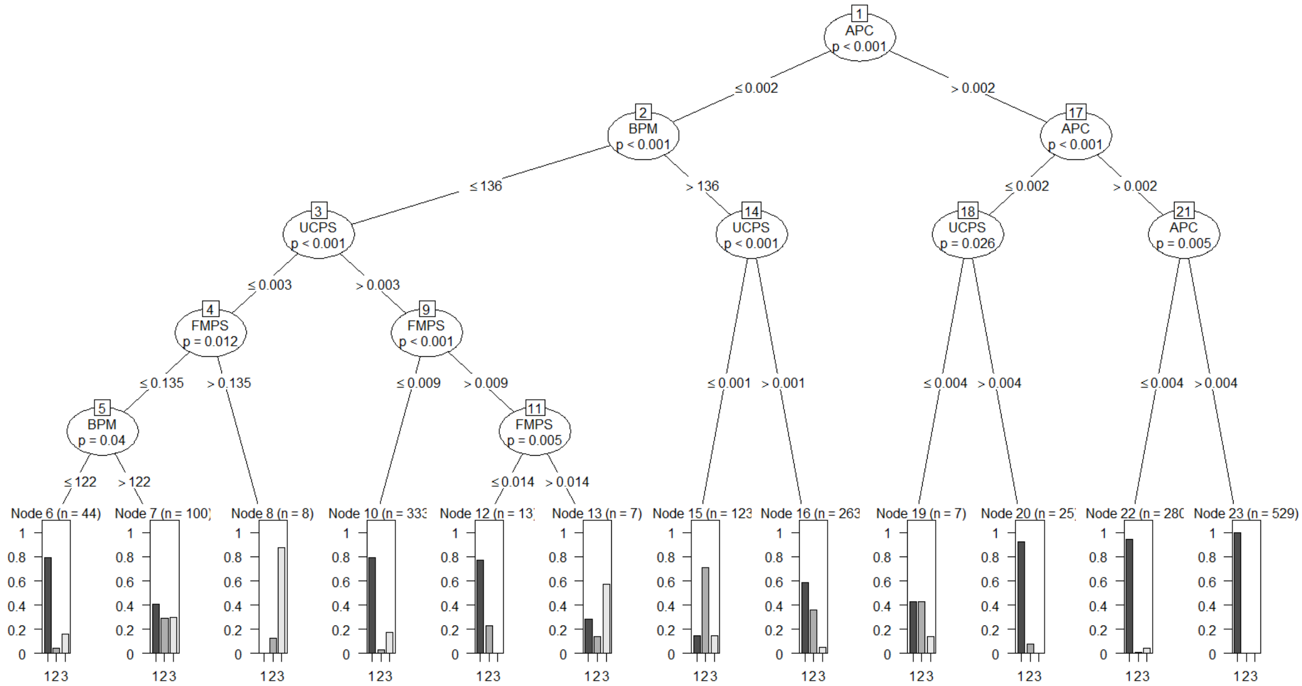
```
print(cardio_tree) #  Draw the tree
```

```
         Conditional inference tree with 18 terminal nodes

Response:  NSPF
Inputs:  BPM, APC, FMPS, UCPS
Number of observations:  1685

1) APC <= 0.000823723; criterion = 1, statistic = 260.946
  2) BPM <= 136; criterion = 1, statistic = 131.576
    3) UCPS <= 0.003562945; criterion = 1, statistic = 41.027
      4) FMPS <= 0.1351126; criterion = 0.998, statistic = 14.849
        5) BPM <= 122; criterion = 0.991, statistic = 12.147
          6)*  weights = 43
        5) BPM > 122
          7)*  weights = 88
      4) FMPS > 0.1351126
        8)*  weights = 8
    3) UCPS > 0.003562945
      9)*  weights = 267
  2) BPM > 136
    10) UCPS <= 0.000834028; criterion = 1, statistic = 35.973
      11)*  weights = 114
    10) UCPS > 0.000834028
      12)*  weights = 191
1) APC > 0.000823723
  13) APC <= 0.001841621; criterion = 1, statistic = 58.776
    14) UCPS <= 0.009538951; criterion = 1, statistic = 25.623
      15) BPM <= 136; criterion = 0.998, statistic = 15.561
        16) FMPS <= 0.001342282; criterion = 0.999, statistic = 16.949
          17)*  weights = 70
        16) FMPS > 0.001342282
          18)*  weights = 18
      15) BPM > 136
        19)*  weights = 73
    14) UCPS > 0.009538951
      20)*  weights = 7
  13) APC > 0.001841621
    21) APC <= 0.004444444; criterion = 1, statistic = 19.223
      22) BPM <= 110; criterion = 0.996, statistic = 13.644
        23)*  weights = 16
      22) BPM > 110
        24) UCPS <= 0.001404494; criterion = 0.988, statistic = 11.639
          25) BPM <= 130; criterion = 0.998, statistic = 15.199
            26)*  weights = 30
          25) BPM > 130
            27) BPM <= 134; criterion = 0.984, statistic = 11.033
              28)*  weights = 8
            27) BPM > 134
              29)*  weights = 17
        24) UCPS > 0.001404494
          30) UCPS <= 0.009578544; criterion = 1, statistic = 18.085
            31) FMPS <= 0.01004304; criterion = 1, statistic = 14.787
              32)*  weights = 195
            31) FMPS > 0.01004304
              33)*  weights = 19
          30) UCPS > 0.009578544
            34)*  weights = 7
    21) APC > 0.004444444
```

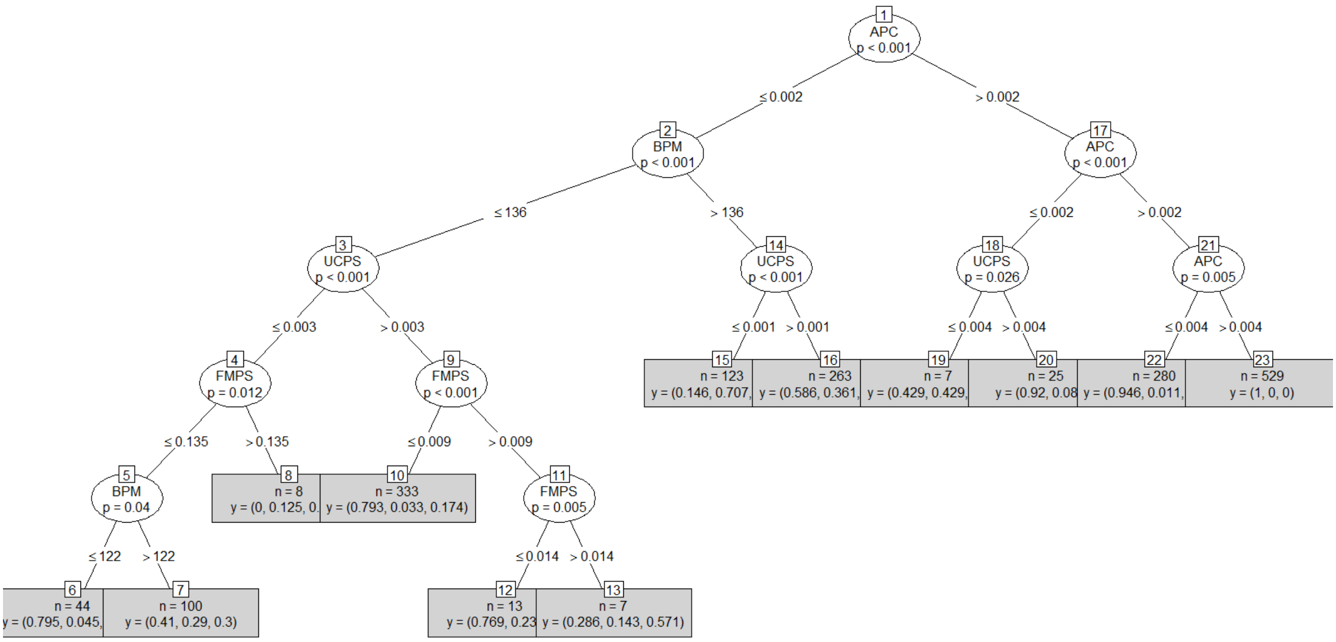
```
plot(cardio_tree)        # Plot the tree
```



The Tree shows that the most important variable in the class variable detection is APC as it is the root.

There are many other types of plotting and you can try theme for example use code `plot(cardio_tree, type="simple")`

Result



12. Following code is going to check the prediction on train data itself.

You can use the predict() function to make predictions from the model you build.

```
predict(cardio_tree)
```

```
> predict(cardio_tree)
  [1] 1 1 1 1 3 3 1 2 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [64] 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[127] 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[190] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2
[253] 2 2 2 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1
[316] 1 2 2 1 2 2 2 2 2 2 2 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 1 2 2
[379] 2 2 2 1 2 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1
[442] 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 2
[505] 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1
[568] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[631] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[694] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[757] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[820] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[883] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[946] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [ reached getOption("max.print") -- omitted 685 entries ]
Levels: 1 2 3
```

You can generate frequency tables using the **table( )** function. table() function simply creates tabular results of categorical variables.

```
tab <- table(predict(cardio_tree), train$NSPF)
print(tab)
```

Result :

```
> tab<-table(predict(cardio_tree), train$NSPF)
> print(tab)

       1    2    3
  1 1324  148  122
  2   18   87   18
  3    2    2   11
```

13. Calculate classification accuracy and error on train data itself.

```
#diag() function extracts the diagonal of a matrix
sum(diag(tab))/sum(tab)
```

```
> sum(diag(tab))/sum(tab)
[1] 0.8210162
```

This result shows that classification is 82.10% is accurate.

Or equivalently classification error is 17.89%

```
1-sum(diag(tab))/sum(tab)
```

```
> 1-sum(diag(tab))/sum(tab)
[1] 0.1789838
```

14. Do you think classification accuracy 82.10% is good?

Hold On !!!

We just validated training data. It does not make any sense, because same training data is used to build the model. Let's validate the model on test data set.

```
test_predict <- table(predict(cardio_tree, newdata= validate), validate$NSPF)
print(test_predict)
```

Result

```
> test_predict <- table(predict(cardio_tree, newdata= validate), validate$NSPF)
> print(test_predict)

      1   2   3
  1 301  41  18
  2   8  17   4
  3   2   0   3
```

15. Calculate classification accuracy and error on test data set.

```
sum(diag(test_predict)))/sum(test_predict))
```

```
> sum(diag(test_predict))/sum(test_predict)
[1] 0.8147208
```

This result shows that our classification is 81.47% is accurate on test data set.

Or equivalently classification error is 18.52%

```
1-sum(diag(tab))/sum(tab)
```

```
> 1-sum(diag(test_predict))/sum(test_predict)
[1] 0.1852792
```

16. You can use other available R packages for decision trees.
    Example: *rpart, tree, maptree, partykit, evtree, randomForest, varSelRF etc.*

17. You can do a variety of practice on other variables as you wish.

    Eg 1: Use BPM, DLPS, SDPS and PDPS to predict the NSP value.

    Eg 2: Use all the variables to predict the NSP value.