



**K2 Analytics**  
Building Skills, Building Individuals

## **Business Analytics using Data Mining (BADM)**

**- Rajesh Jakhotia**

**12-Oct-2015**

**Earning is in Learning  
- Rajesh Jakhotia**



**K2 Analytics**  
Building Skills, Building Individuals

## **Agenda**

*Introduction*

*Data Mining in a nut shell*

*Basic number skills*

*Classification Tree*

*CHAID, CART, C4.5 & Random Forest*

# About K2 Analytics

*At K2 Analytics, we believe that skill development is very important for the growth of an individual, which in turn leads to the growth of Society & Industry and ultimately the Nation as a whole. For this it is important that access to knowledge and skill development trainings should be made available easily and economically to every individual.*

**Our Vision:** *“To be the preferred partner for training and skill development”*

**Our Mission:** *“To provide training and skill development training to individuals, make them skilled & industry ready and create a pool of skilled resources readily available for the industry”*

*We have chosen Business Intelligence and Analytics as our focus area. With this endeavour we make this presentation on “**Business Analytics using Data Mining (BADM)**” accessible to all those who wish to learn Analytics. We hope it is of help to you. For any feedback / suggestion or if you are looking for job in analytics then feel free to write back to us at [ar.jakhotia@k2analytics.co.in](mailto:ar.jakhotia@k2analytics.co.in)*

*Welcome to BADM!!!*



**K2 Analytics**  
Building Skills, Building Individuals

## *Data Mining in a nut shell*

# Statistics Vs. Data Mining

## Statistics

- the practice or science of collecting and analysing numerical data in large quantities, especially for the purpose of inferring proportions in a whole from those in a representative sample.
- Infer / Describe
- Data Collection
- Large Dataset implies hundred / thousand data points
- Population / Sample Level
- Charts & Table
- Makes many assumptions

## Data Mining

- Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.
- Predict
- Data Preparation
- Large Datasets implies millions / billions data points
- Customer (Granular) Level
- Visualizations
- Makes few / no assumptions

<http://www.cs.csi.cuny.edu/~imberman/DataMining/Statistics%20vs.pdf>

# Types of Data Mining Techniques



- **Supervised learning:** The target output expected is clearly defined

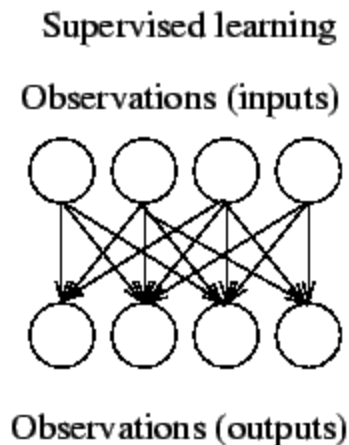


- **Unsupervised learning:** The data have no target attribute.
  - We want to explore the data to find some intrinsic structures in them.

# Types of Data Mining techniques

## Supervised Techniques

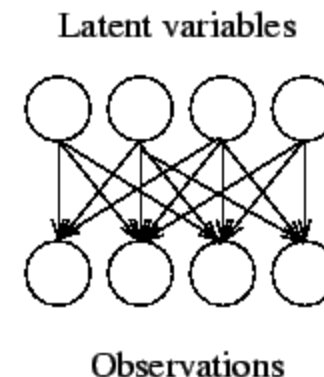
- In supervised learning, the model defines the effect one set of observations, called inputs, has on another set of observations, called outputs



- Prediction (numerical Y)
- Classification (Categorical Y)

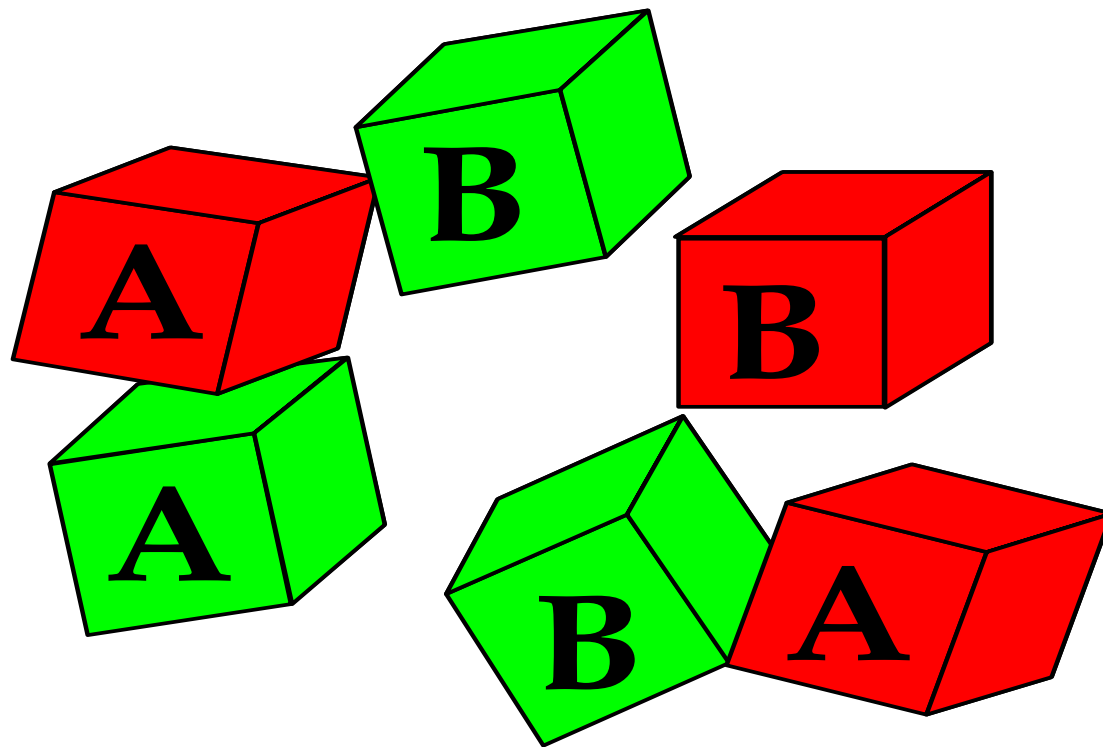
## Unsupervised Techniques

- In unsupervised learning, all the observations are assumed to be caused by latent variables, that is, the observations are assumed to be at the end of the causal chain



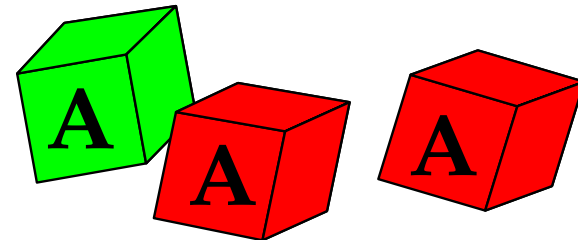
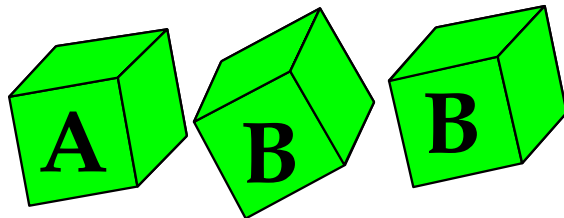
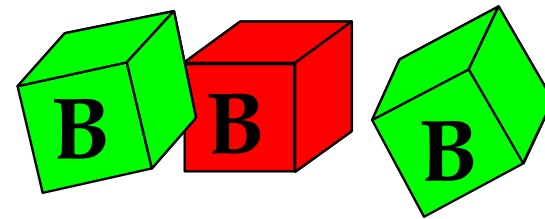
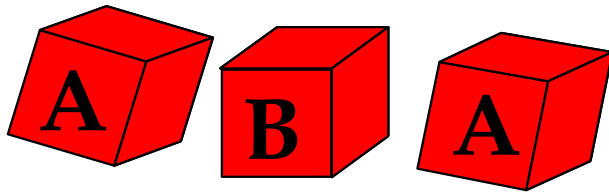
- Dimension Reduction
- Clustering
- Association Analysis

# Understanding Supervised and Unsupervised Learning



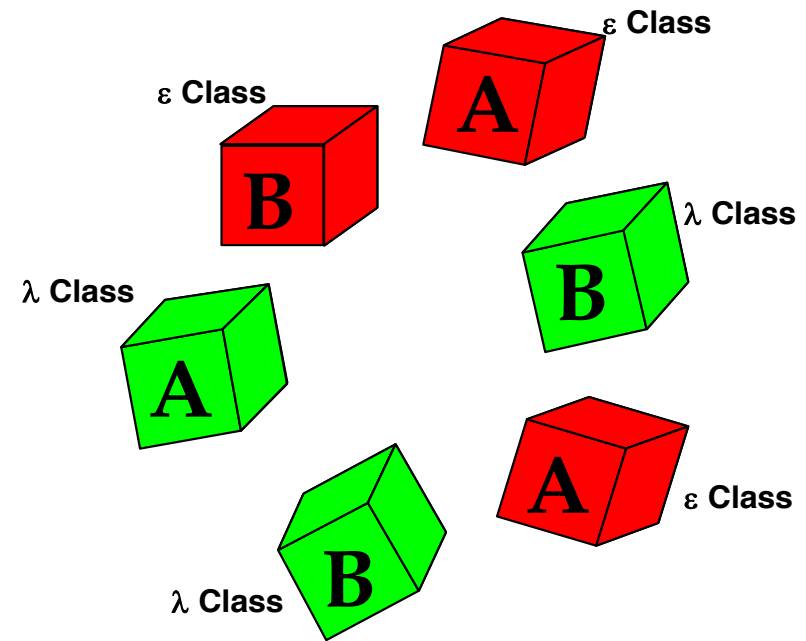


# Two Possible Solutions



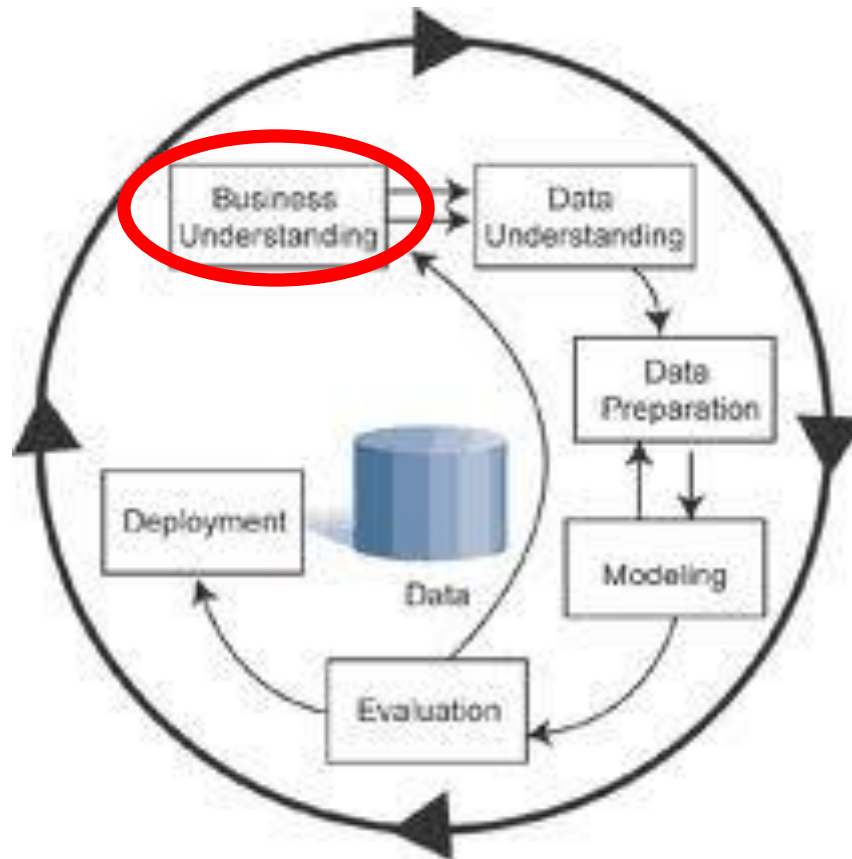
# Supervised Learning

- It is based on a labeled training set.
- The class of each piece of data in training set is known.
- Class labels are pre-determined and provided in the training phase.



# Modeling Process: CRISP-DM

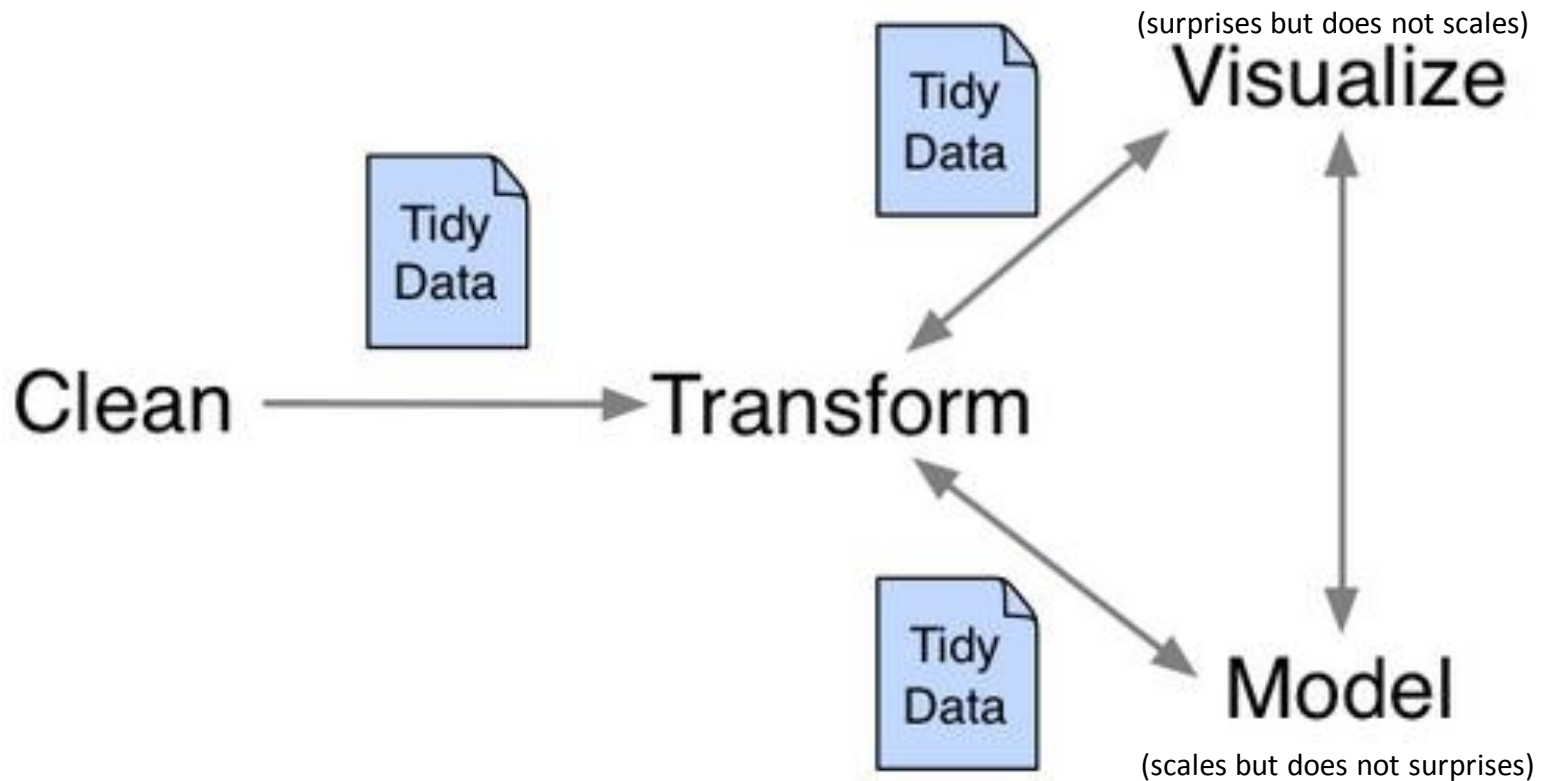
- CRISP – DM stands for Cross Industry Standard Process for Data Mining



Most challenging part in modeling is defining the objective

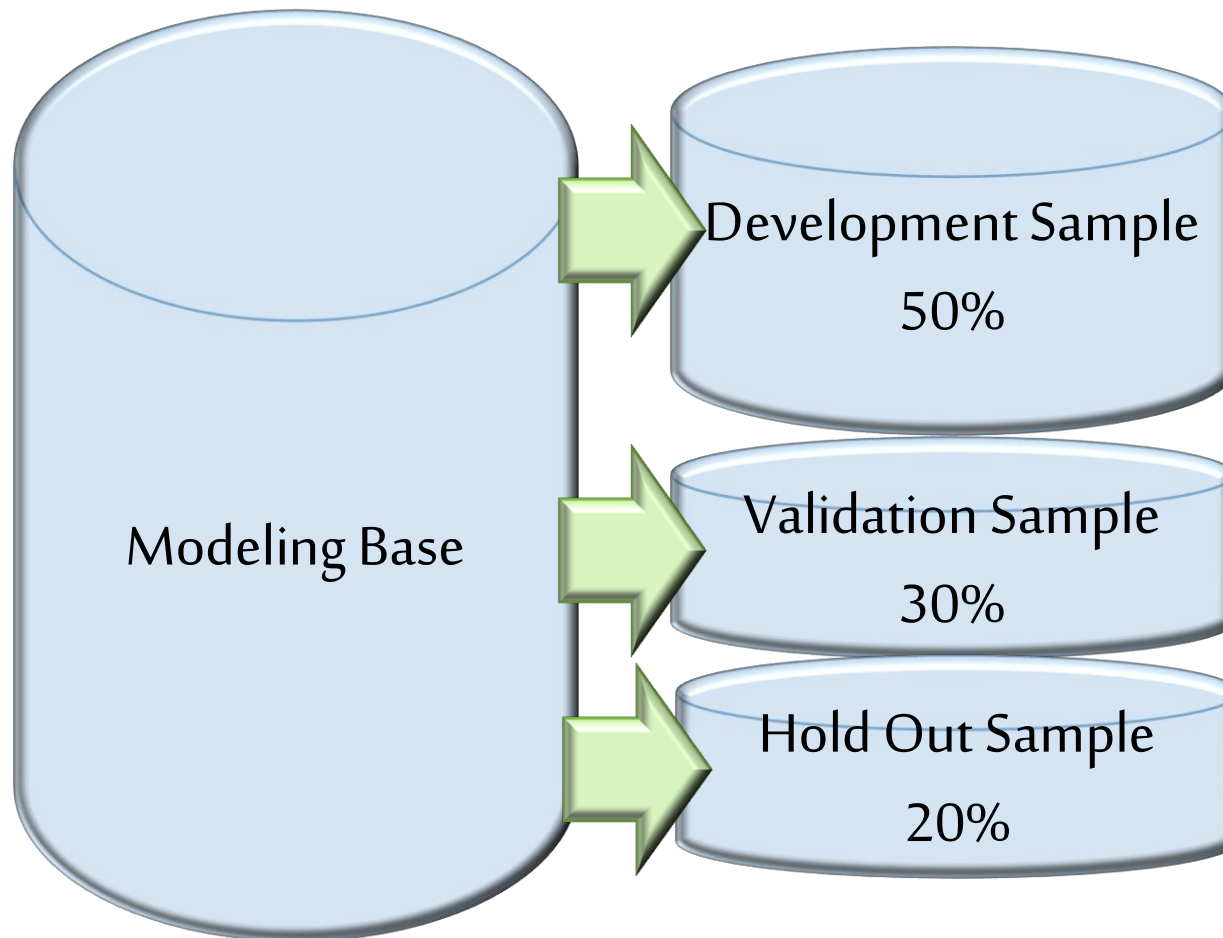
FIGURE 1 Data-Mining Process Model

# Data Preparation & Modeling Process



Note: Figure adapted and modified from a [presentation](#) by Hadley Wickham.

# Hold Out concept in Model Evaluation





**K2 Analytics**  
Building Skills, Building Individuals

***Basic Number Skills***  
***Modeling Techniques***

# Basic Number Skills

- Measures of Central Tendency
  - Mean, Median, Mode
- Measures of Dispersion
  - Std. Deviation, Variance
- Correlation and Covariance
- Chi-Sq Test
- Additive Variables, Count and Ratio

# Standardization & Normalization

- Standardization & Normalization are 2 commonly used method for rescaling
- Normalization*, which scales all numeric variables in the range [0,1]. One possible formula is given below:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Standardization transforms data to mean zero and unit variance

$$x_{new} = \frac{x - \mu}{\sigma}$$



# Hypothesis Testing

Hypothesis testing is the use of statistics to determine the probability that a given hypothesis is true. The usual process of hypothesis testing consists of four steps.

1. Formulate the **null hypothesis** (commonly, that the observations are the result of pure chance) and the **alternate hypothesis** (commonly, that the observations show a real effect combined with a component of chance variation).
2. Identify a **test statistic** that can be used to assess the truth of the **null hypothesis**
3. Compute the **p-value**, which is the probability that a test statistic at least as significant as the one observed would be obtained assuming that the **null hypothesis** were true. The smaller the p-value, the stronger the evidence against the null hypothesis.
4. Compare the p-value to an acceptable significance value  $\alpha$  (sometimes called an **alpha value** ). If  $\alpha \leq p$ , that the observed effect is statistically significant, the null hypothesis is ruled out, and the alternative hypothesis is valid.

# Uni, Bi & Multi-variate analysis

- Univariate Analysis – Descriptive statistics like Mean, Median, Mode, STD Deviation, Variance, Frequency Distribution
  
- Bi & Multi-variate analysis –
  - Differences of Group ( Chi-Sq, t-Test, ANOVA)
  
  - Relationship (Correlation & Regression)

# Cardinal, Ordinal, & Nominal Numbers



A **cardinal number** tells "**how many**." Cardinal numbers are also known as "counting numbers," because they **show quantity**.

Here are some examples using cardinal numbers:

- 8 puppies
- 14 friends

**Ordinal numbers** tell the **order of things in a set**—first, second, third, etc. Ordinal numbers do not show quantity. They only **show rank or position**.

Here are some examples using ordinal numbers:

- 3rd fastest
- 6th in line



A **nominal number names something**—a telephone number, a player on a team. Nominal numbers do not show quantity or rank. They are used only to **identify something**.

Here are some examples using nominal numbers:

- jersey number 4
- zip code 02116

<http://www.factmonster.com/ipka/A0875618.html>



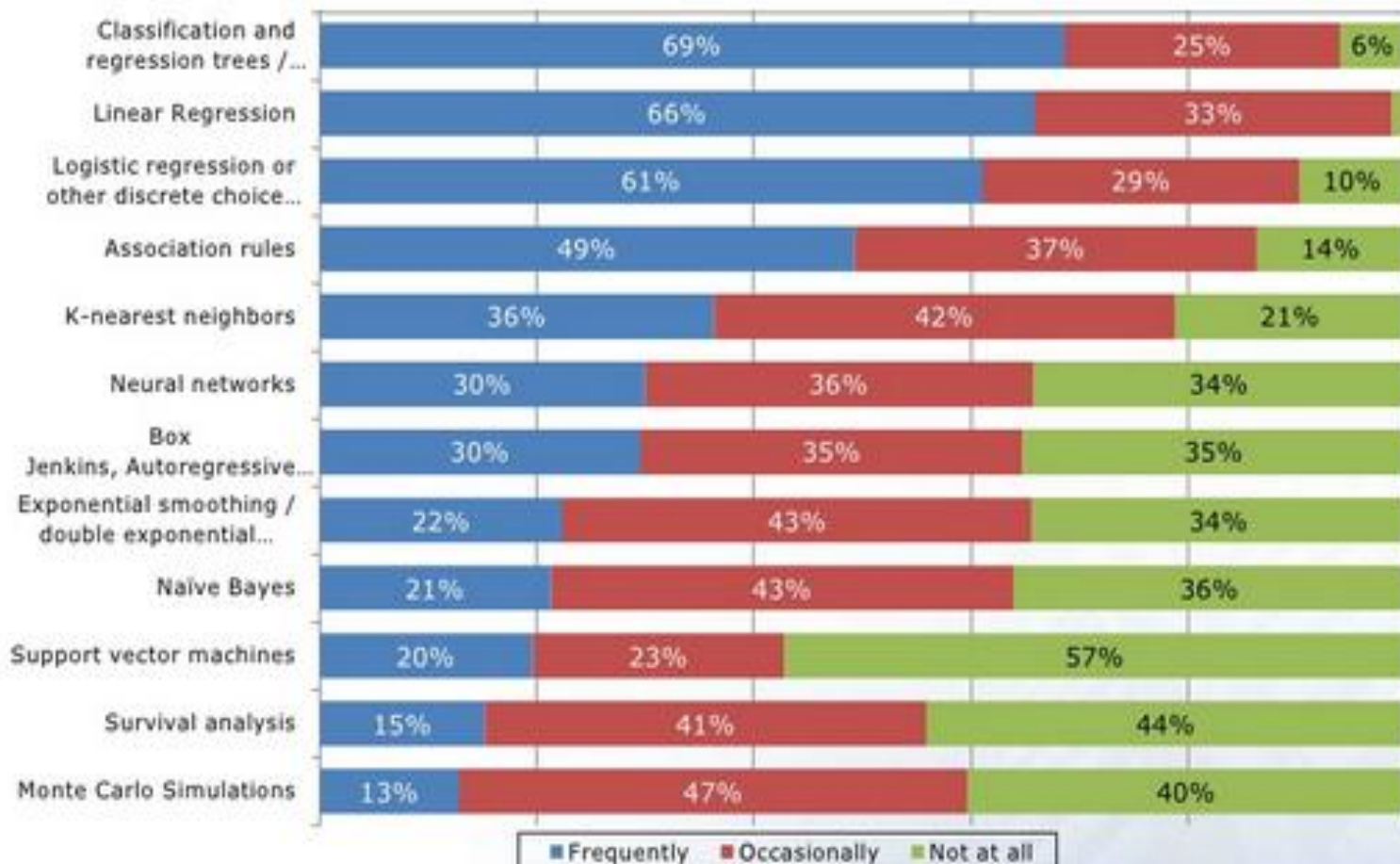
**K2 Analytics**  
Building Skills, Building Individuals

## ***Classification Techniques***

CHAID

CART

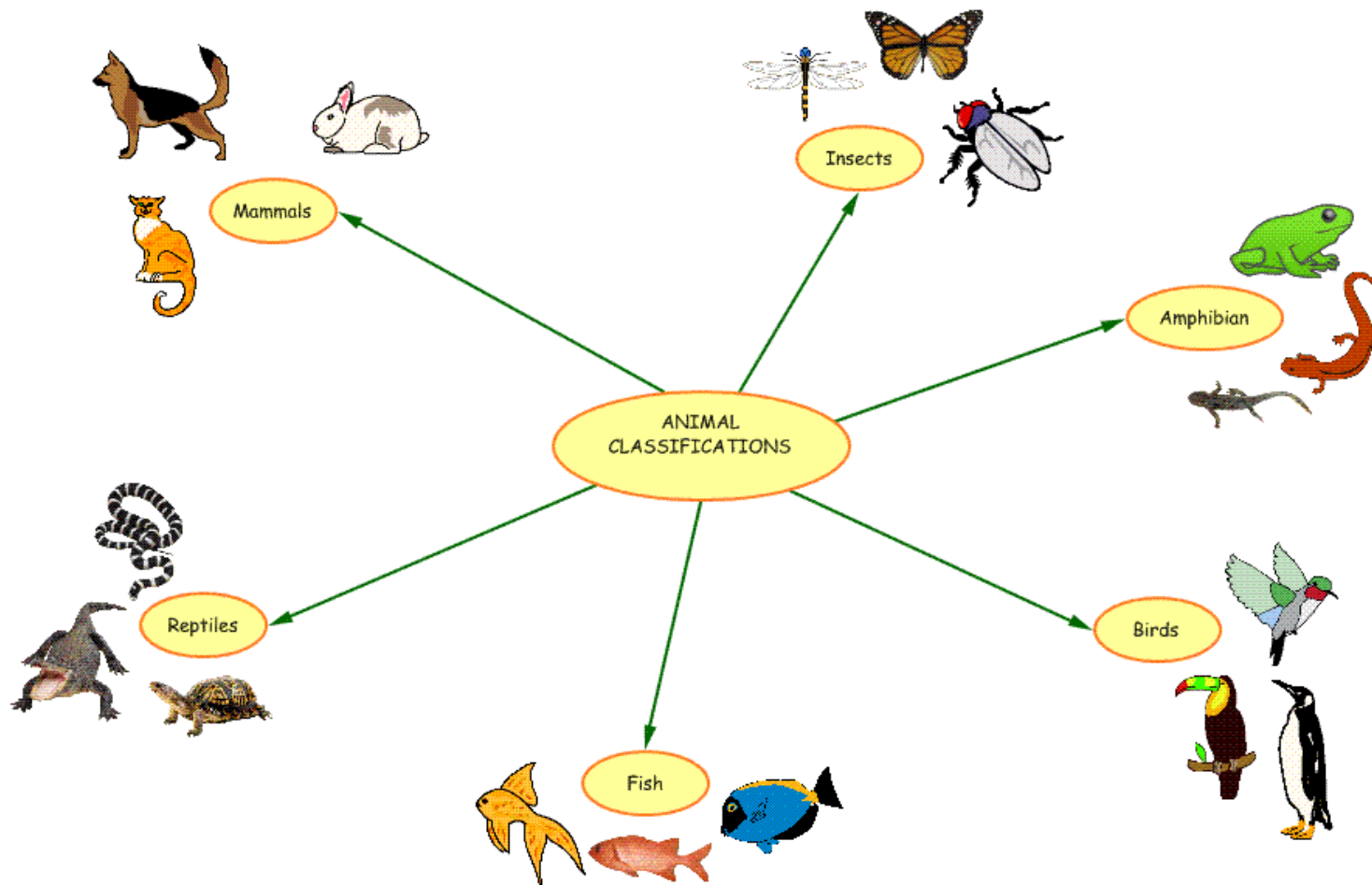
# Analytics that are Actually Used



*Classification and regression trees / decision trees and Linear Regression are the most popular predictive analytics techniques used.*

# What is Classification?

The action or process of classifying something according to shared qualities or characteristics.



# Defining Characteristics of each animal classification

- Mammals – Mammals are vertebrates (backboned animals). Mammals are warm-blooded and have hair. Mammals are able to move around using limbs
- Birds – Birds are warm-blooded vertebrates, having a body covered with feathers, forelimbs modified into wings, scaly legs, a beak, and no teeth, and bearing young ones in a hard-shelled egg
- Insects – any of small invertebrate animals which typically have a well defined head, thorax, and abdomen, only three pairs of legs, and typically one or two pair of wings
- Amphibian - any cold-blooded vertebrate that live on land but breed in water
- Reptiles - class of cold-blooded air-breathing vertebrates with completely ossified skeleton and a body usually covered with scales or horny plates
- Fish - A limbless cold-blooded vertebrate animal with gills and fins and living wholly in water

# Why Classify?

To Explain (Profile)

*Explaining in the classification world is called Profiling*

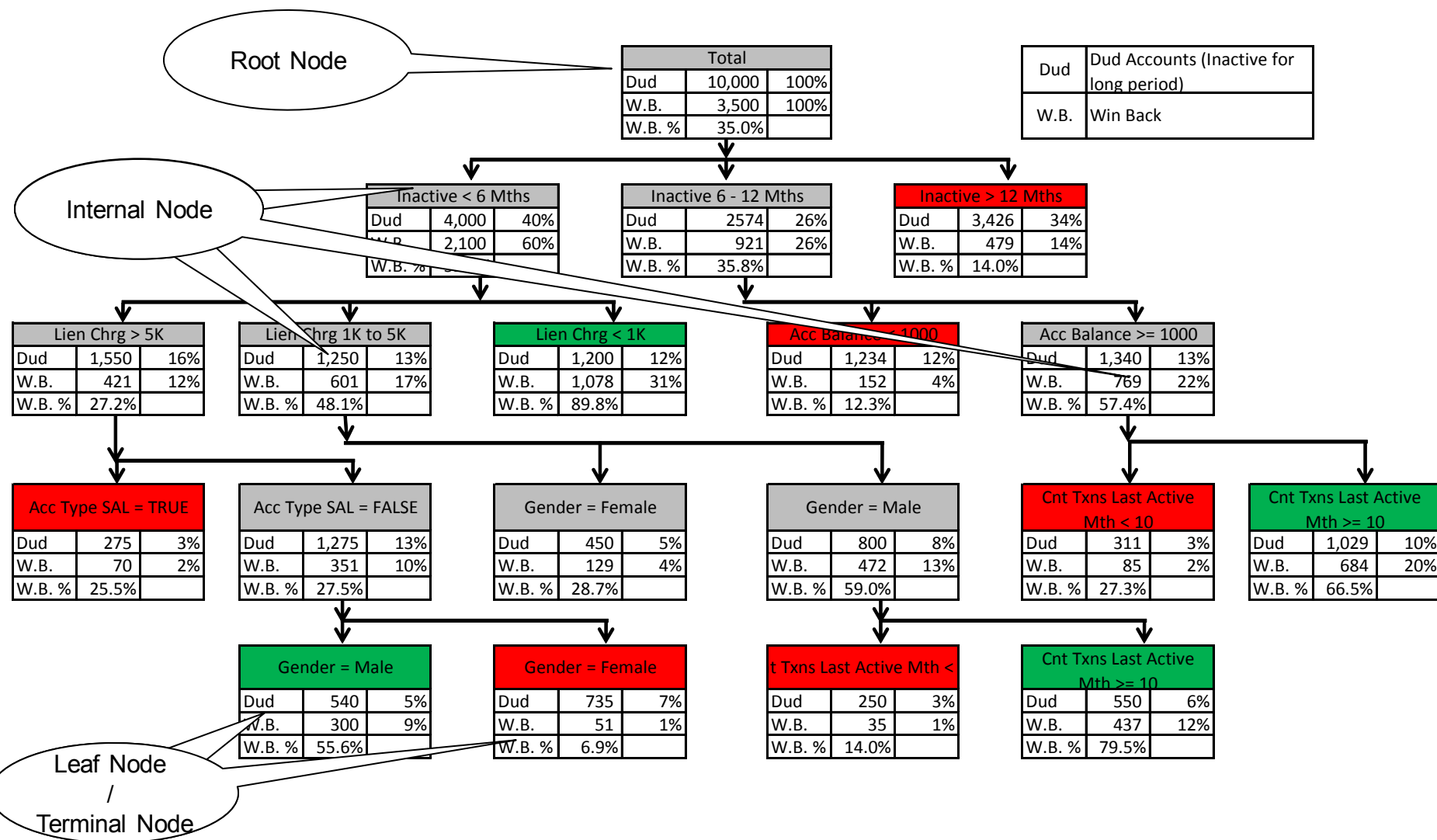
or

To Predict (Classify)

*Predicting the class of new records is called Classifying*



# Win Back Campaign Classification Analysis



# Main issues of classification tree learning

- Choosing the splitting criterion
  - Impurity based criteria
  - Information gain
  - Statistical measures of association
  
- Binary or multiway splits
  - Multiway split
  - Binary split
  
- Finding the right sized tree
  - Pre-pruning
  - Post-pruning

# Popular Classification Techniques

- **CHAID - CHi-squared Automatic Interaction Detector.** The “*Chi-squared*” part of the name arises because the technique essentially involves automatically constructing many cross-tabs, and working out statistical significance of the proportions. The most significant relationships are used to control the structure of a tree diagram
  - CHAID is a non-binary decision tree; **Recursive Partitioning Algorithm**
  - Continuous variables must be grouped into a finite number of bins to create categories.
- **CLASSIFICATION AND REGRESSION TREES (CART)** are binary decision trees, which split a single variable at each node.
  - The CART algorithm recursively goes through an exhaustive search of all variables and split values to find the optimal splitting rule for each node.
- **C4.5** builds decision trees from a set of training data using the concept of information entropy

# CHAID I Splitting Criteria

## Contingency table

Cross tabulation between Y and X

$Y / X$	$x_1$	$x_l$	$x_L$	$\Sigma$
$y_1$		$\vdots$		
$y_k$	$\dots$	$n_{kl}$	$\dots$	$n_{k.}$
$y_K$		$\vdots$		
$\Sigma$		$n_{.l}$		$n$

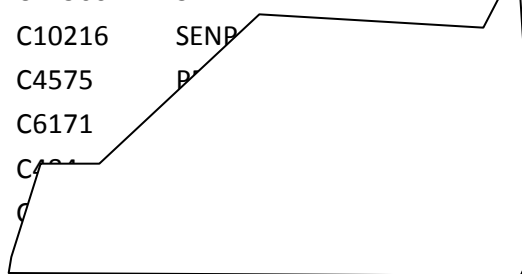
## Measures of association

Comparing the observed and theoretical frequencies  
(under the null hypothesis : Y and X are independent)

$$\chi^2 = \sum_{k=1}^K \sum_{l=1}^L \frac{\left( n_{kl} - \frac{n_{k.} \times n_{.l}}{n} \right)^2}{\frac{n_{k.} \times n_{.l}}{n}}$$

# CHAID I Merging Criteria

Cust_ID	Occupation	Target
C16505	SELF-EMP	1
C17241	SAL	0
C18802	SENP	1
C19289	PROF	0
C14028	SELF-EMP	0
C17960	SAL	
C10216	SENP	
C4575	P	
C6171		
C		
C		



Example:

**Should we Merge the Occupation Categories or treat each separately?**

- Cycle through predictors to determine for each predictor the pair of categories that is least significantly different with respect to the dependent variable
- It will compute a Chi-Square test; if the respective test for a given pair of categorical variables is not statistically significant as defined by alpha-to-merge, then it will merge the respective predictor categories
- Repeat the steps to find next pair of categories, which now may include previously merged categories

# R Code ... Data Import & Data Understanding

## Let us first set the working directory path

```
setwd ("C:/K2-Analytics/Colleges/GLIM/")  
getwd()
```

## Let us import the data that we intend to use for modeling

```
CTDF <- read.table("datafile/CF_TREE_SAMPLE.csv", sep = ",", header = T)  
head(CTDF)
```

## Let us quickly understand the structure of our data

```
str(CTDF)  
CTDF$Target = as.factor(CTDF$Target)
```

```
summary(CTDF)
```

# CHI-SQ Calculation...e.g.

## Cross Tab between Occupation and Target

```
tbl <- table(CTDF$Occupation, CTDF$Target)
```

```
tbl
```

```
chisq.test(tbl)
```

	0	1
PROF	5028	435
SAL	5426	413
SELF-EMP	2858	508
SENP	4955	377

Pearson's Chi-squared test

```
data: tbl
```

```
X-squared = 214.92, df = 3, p-value < 2.2e-16
```

CHI-SQ Calculations	Observed		Row Total	Expected		(O - E)^2/E	
	0	1		0	1	0	1
PROF	5,028	435	5,463	4,990	473	0.30	3.11
SAL	5,426	413	5,839	5,333	506	1.62	17.08
SELF-EMP	2,858	508	3,366	3,074	292	15.22	160.46
SENP	4,955	377	5,332	4,870	462	1.48	15.64
Total	18,267	1,733	20,000	18,267	1,733	CHI-SQ	215
Total Proportions	0.913	0.087					

Degree of Freedom = (m-1) \* (n-1) = (4-1) \* (2-1) = 3

p-value is arrived by seeing the CHI Sq Table based on CHI-SQ and Degree of Freedom

# R Code

```
## Let us first set the working directory path
```

```
setwd ("C:/K2-Analytics/Colleges/GLIM/")  
getwd()
```

```
## Let us import the data that we intend to use for modeling
```

```
CTDF <- read.table("datafile/CF_TREE_SAMPLE.csv", sep = ",", header = T)  
head(CTDF)
```

```
## Let us quickly understand the structure of our data
```

```
str(CTDF)  
CTDF$Target = as.factor(CTDF$Target)
```

```
## Installing the CHAID package
```

```
## install.packages("partykit")
```

```
## install.packages("CHAID", repos="http://R-Forge.R-project.org")
```

```
library(CHAD)
```

```
ctrl <- chaid_control(minbucket = 100, minsplit = 100, alpha2=.05, alpha4 = .05)
```

```
chaid.tree <-chaid(Target~Gender+AGE_BKT,data=CTDF, control = ctrl)
```

```
print(chaid.tree)
```

```
plot(chaid.tree)
```

**Merging Threshold:** merge sub-categories in predictor variable if p-value is above alpha2 threshold

**Splitting Threshold:** If p-value below alpha4 consider the predictor for splitting of the node

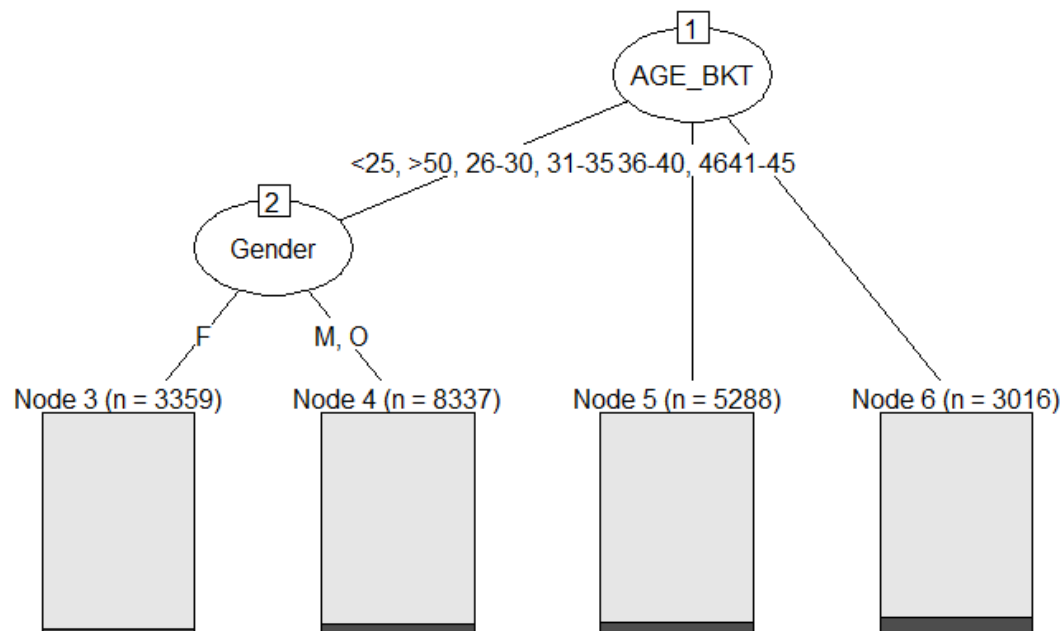


# CHAID Output

Model formula:  
 Target ~ Gender + AGE\_BKT

Fitted party:  
 [1] root  
 | [2] AGE\_BKT <25, >50, 26-30, 31-35  
 | | [3] Gender in F: 0 (n = 3359, err = 2.1%)  
 | | [4] Gender in M, O: 0 (n = 8337, err = 4.0%)  
 | [5] AGE\_BKT in 36-40, 46-50: 0 (n = 5288, err = 5.0%)  
 | [6] AGE\_BKT in 41-45: 0 (n = 3016, err = 7.3%)

Number of inner nodes: 2  
 Number of terminal nodes: 4



Note:

- CHAID package in R is still not fully developed... as such we will move on to another technique

## CART

Run the CHAID with **Occupation** field also and see this plot

# CART | Splitting Criteria

- CART uses the Gini Index as measure of impurity
- Gini of a Node

$$GINI(t) = 1 - \sum_j [p(j | t)]^2$$

(NOTE:  $p(j | t)$  is the relative frequency of class  $j$  at node  $t$ ).

- Gini of Split Node is computed as Weighted Avg Gini of each Node at Split Node level

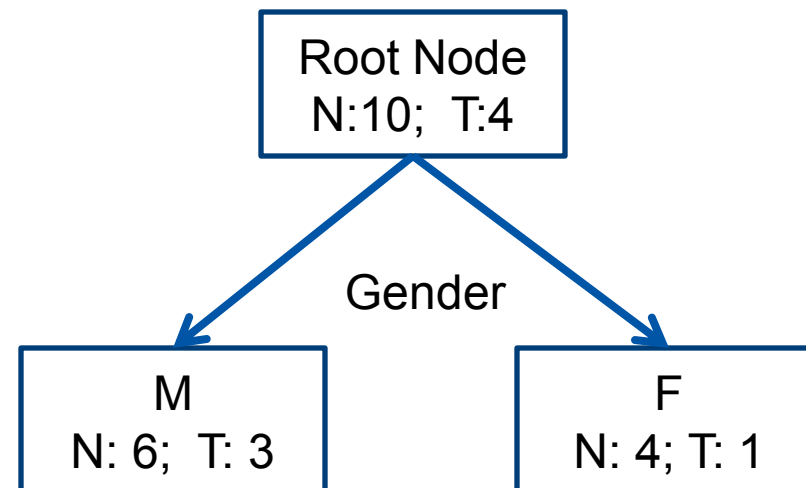
$$GINI_{split} = \sum_{i=1}^k \frac{n_i}{n} GINI(i)$$

$n_i$  = number of records at child  $i$ ,  
 $n$  = Total number of records in parent node

- Gini Gain =  $GINI(t) - GINI(split)$

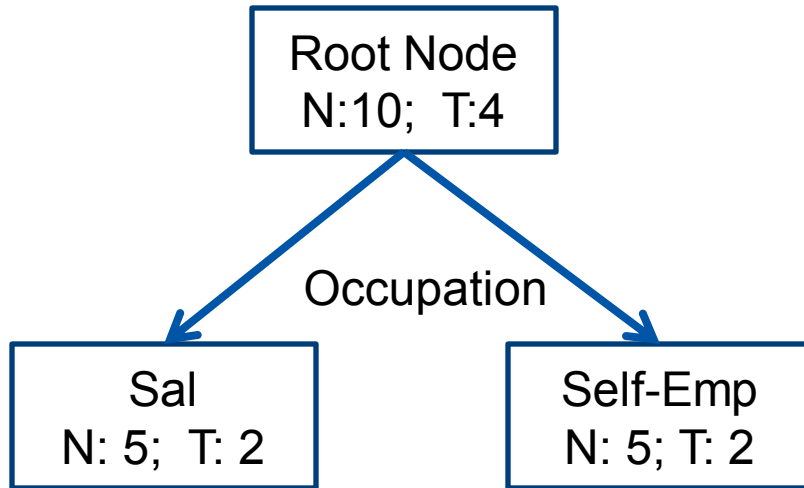
# Gini calculations

Cust_ID	Gender	Occupation	Age	Target
1	M	Sal	22	1
2	M	Sal	22	0
3	M	Self-Emp	23	1
4	M	Self-Emp	23	0
5	M	Self-Emp	24	1
6	M	Self-Emp	24	0
7	F	Sal	25	1
8	F	Sal	25	0
9	F	Sal	26	0
10	F	Self-Emp	26	0



Node	Gini Computation Formula	Gini Index
Overall	$= 1 - ( (4/10)^2 + (6/10)^2 )$	0.48
Gender = M	$= 1 - ( (3/6)^2 + (3/6)^2 )$	0.50
Gender = F	$= 1 - ( (1/4)^2 + (3/4)^2 )$	0.375
Gender	$= (6/10) * 0.5 + (4/10) * 0.375$	0.45
Gini Gain	$= \text{Gini (Overall)} - \text{Gini (Gender)}$	0.03

# Gini calculations



Node	Gini Computation Formula	Gini Index
Overall	$= 1 - ( (4/10)^2 + (6/10)^2 )$	0.48
Occ = Sal	$= 1 - ( (2/5)^2 + (2/5)^2 )$	0.48
Occ = Self-Emp	$= 1 - ( (2/5)^2 + (2/5)^2 )$	0.48
Occupation	$= (5/10) * 0.48 + (5/10) * 0.48$	0.48
Gini Gain	$= \text{Gini (Overall)} - \text{Gini (Occupation)}$	0.0

Age	<=22	<=23	<=24	<=25
Gini (Left)	0.5	0.5	0.5	0.5
Gini (Right)	0.47	0.44	0.38	0
Gini Split	0.48	0.47	0.45	0.40
Gini Gain	0.0	0.01	0.03	0.08

# Sampling...

```
## Creating Development and Validation Sample  
## CTDF$random <- runif (nrow(CTDF), 0, 1);  
## CTDF.dev <- CTDF [which(CTDF$random <= 0.7),]  
## CTDF.holdout <- CTDF [which(CTDF$random > 0.7),]  
## c (nrow(CTDF.dev), nrow(CTDF.holdout))
```

Sampling Code

Separate Dev & Val  
samples are provided as  
such we will directly  
import them rather than  
use sampling code

```
CTDF.dev <- read.table ("datafile/DEV_SAMPLE.csv", sep = ",", header = T)  
CTDF.holdout <- read.table ("datafile/HOLDOUT_SAMPLE.csv", sep = ",", header = T)
```

# rpart code to build CART Tree

```
## installing rpart package for CART
```

```
## install.packages("rpart")
```

```
## install.packages("rpart.plot")
```

```
## loading the library
```

```
library(rpart)
```

```
library(rpart.plot)
```

```
## setting the control parameter inputs for rpart
```

```
r.ctrl = rpart.control(minsplit=100, minbucket = 10, cp = 0, xval = 10)
```

```
## calling the rpart function to build the tree
```

```
m1 <- rpart(formula = Target ~ ., data = CTDF.dev[,-1], method = "class", control = r.ctrl)
```

```
m1
```

Complexity Parameter  
Initially set to Zero to allow  
the full tree to be grown

Cross Validation  
Parameter

# rpart.control arguments

- **minsplit**: the minimum number of observations that must exist in a node in order for a split to be attempted.
- **minbucket**: the minimum number of observations in any terminal leaf node. If only one of minbucket or minsplit is specified, the code either sets minsplit to  $\text{minbucket} \times 3$  or minbucket to  $\text{minsplit} / 3$ , as appropriate.
- **cp complexity parameter**: Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile. Essentially, the user informs the program that any split which does not improve the fit by cp will likely be pruned off by cross-validation, and that hence the program need not pursue it.
- **xval**: number of cross-validations

n= 14000

# rpart output

node), split, n, loss, yval, (yprob)  
 \* denotes terminal node

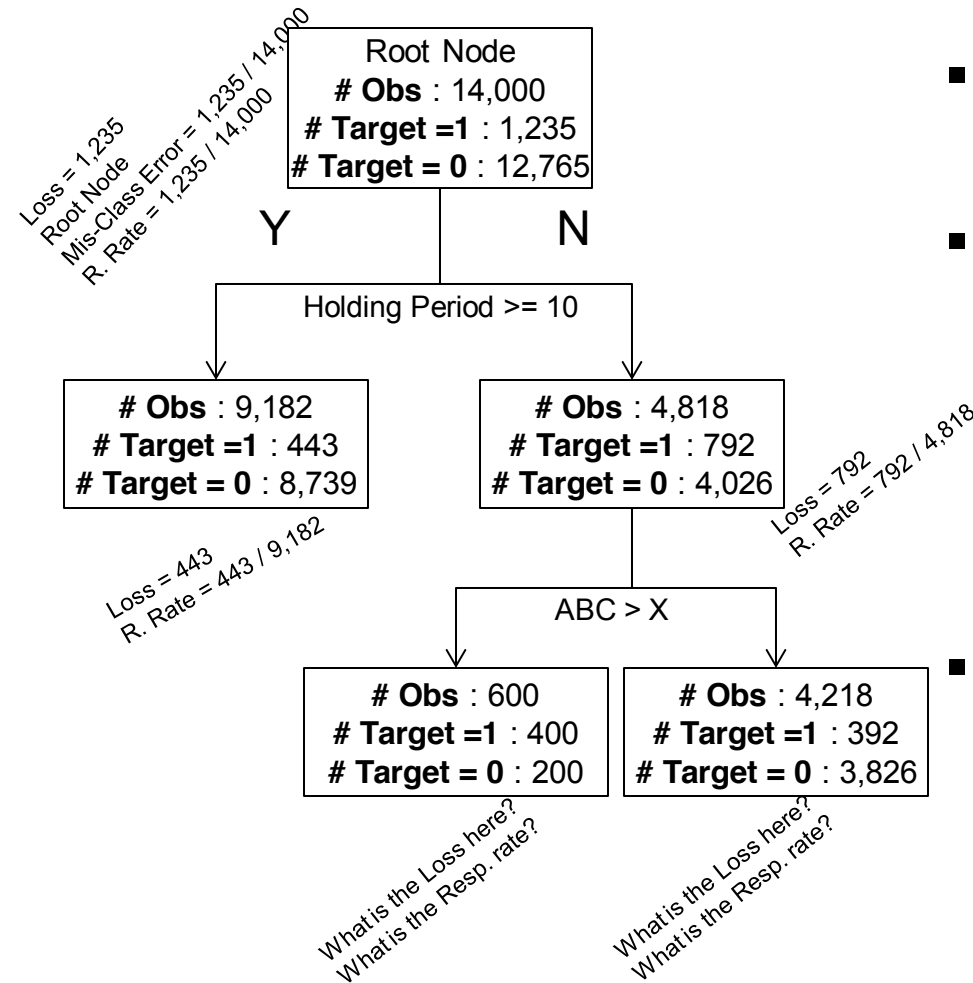
```

1) root 14000 1235 0 (0.91178571 0.08821429)
2) Holding_Period>=10.5 9182 443 0 (0.95175343 0.04824657)
4) No_OF_CR_TXNS< 20.5 6858 212 0 (0.96908720 0.03091280) *
5) No_OF_CR_TXNS>=20.5 2324 231 0 (0.90060241 0.09939759)
10) Occupation=PROF,SAL 1814 124 0 (0.93164278 0.06835722) *
11) Occupation=SELF-EMP,SENP 510 107 0 (0.79019608 0.20980392)
22) SCR< 334.5 120 9 0 (0.92500000 0.07500000) *
23) SCR>=334.5 390 98 0 (0.74871795 0.25128205)
46) Gender=M,0 370 85 0 (0.77027027 0.22972973) *
47) Gender=F 20 7 1 (0.35000000 0.65000000) *
3) Holding_Period< 10.5 4818 792 0 (0.83561644 0.16438356)
6) Occupation=PROF,SAL,SENP 3971 546 0 (0.86250315 0.13749685)
12) No_OF_CR_TXNS< 20.5 2832 317 0 (0.88806497 0.11193503)
24) Balance>=10853.5 2551 259 0 (0.89847119 0.10152881)
48) SCR< 697.5 1618 129 0 (0.92027194 0.07972806) *
49) SCR>=697.5 933 130 0 (0.86066452 0.13933548)
98) Holding_Period>=1.5 791 98 0 (0.87610619 0.12389381)
196) SCR>=732.5 712 81 0 (0.88623596 0.11376404)
392) AGE_BKT>=50,26-30,31-35 358 30 0 (0.91620112 0.08379888) *
393) AGE_BKT<=25,36-40,41-45,46-50 354 51 0 (0.85593220 0.14406780)
786) Balance>=34458.97 278 33 0 (0.88129496 0.11870504)
1572) Balance< 48554.42 28 0 0 (1.00000000 0.00000000) *
1573) Balance>=48554.42 250 33 0 (0.86800000 0.13200000)
3146) Balance>=212700.4 103 8 0 (0.92233010 0.07766990) *
3147) Balance< 212700.4 147 25 0 (0.82993197 0.17006803)
6294) Holding_Period< 2.5 23 0 0 (1.00000000 0.00000000) *
6295) Holding_Period>=2.5 124 25 0 (0.79838710 0.20161290)
12590) Holding_Period>=3.5 108 16 0 (0.85185185 0.14814815) *
12591) Holding_Period< 3.5 16 7 1 (0.43750000 0.56250000) *
787) Balance< 34458.97 76 18 0 (0.76315789 0.23684211) *
197) SCR< 732.5 79 17 0 (0.78481013 0.21518987) *
99) Holding_Period< 1.5 142 32 0 (0.77464789 0.22535211) *
25) Balance< 10853.5 281 58 0 (0.79359431 0.20640569) *
13) No_OF_CR_TXNS>=20.5 1139 229 0 (0.79894644 0.20105356)
26) Occupation=PROF,SAL 1048 174 0 (0.83396947 0.16603053) *
27) Occupation=SENP 91 36 1 (0.39560440 0.60439560) *
7) Occupation=SELF-EMP 847 246 0 (0.70956316 0.29043684)
14) SCR< 725 538 107 0 (0.80111524 0.19888476)
28) No_OF_CR_TXNS< 29.5 434 68 0 (0.84331797 0.15668203) *
29) No_OF_CR_TXNS>=29.5 104 39 0 (0.62500000 0.37500000)
58) Balance>=4171.13 93 31 0 (0.66666667 0.33333333) *
59) Balance< 4171.13 11 3 1 (0.27272727 0.72727273) *
15) SCR>=725 309 139 0 (0.55016181 0.44983819)
30) Balance>=13166.99 205 72 0 (0.64878049 0.35121951)
60) No_OF_CR_TXNS< 20.5 149 39 0 (0.73825503 0.26174497) *
61) No_OF_CR_TXNS>=20.5 56 23 1 (0.41071429 0.58928571) *
31) Balance< 13166.99 104 37 1 (0.35576923 0.64423077)
62) Age>=48 17 6 0 (0.64705882 0.35294118) *
63) Age< 48 87 26 1 (0.29885057 0.70114943) *

```



# Loss, Mis-Classification Error and Response Rate



- Loss is the number of cases mis-classified in a given node
- Mis-Classification Error is the ratio of total number of cases mis-classified to total number of cases
  - We are interested in mis-classification error for the full tree
- Response Rate is the ratio of number of responders (Target = 1) to the total number of cases
  - We are interested in finding nodes where the response rate is very high

What is the mis-classification error for the above tree?

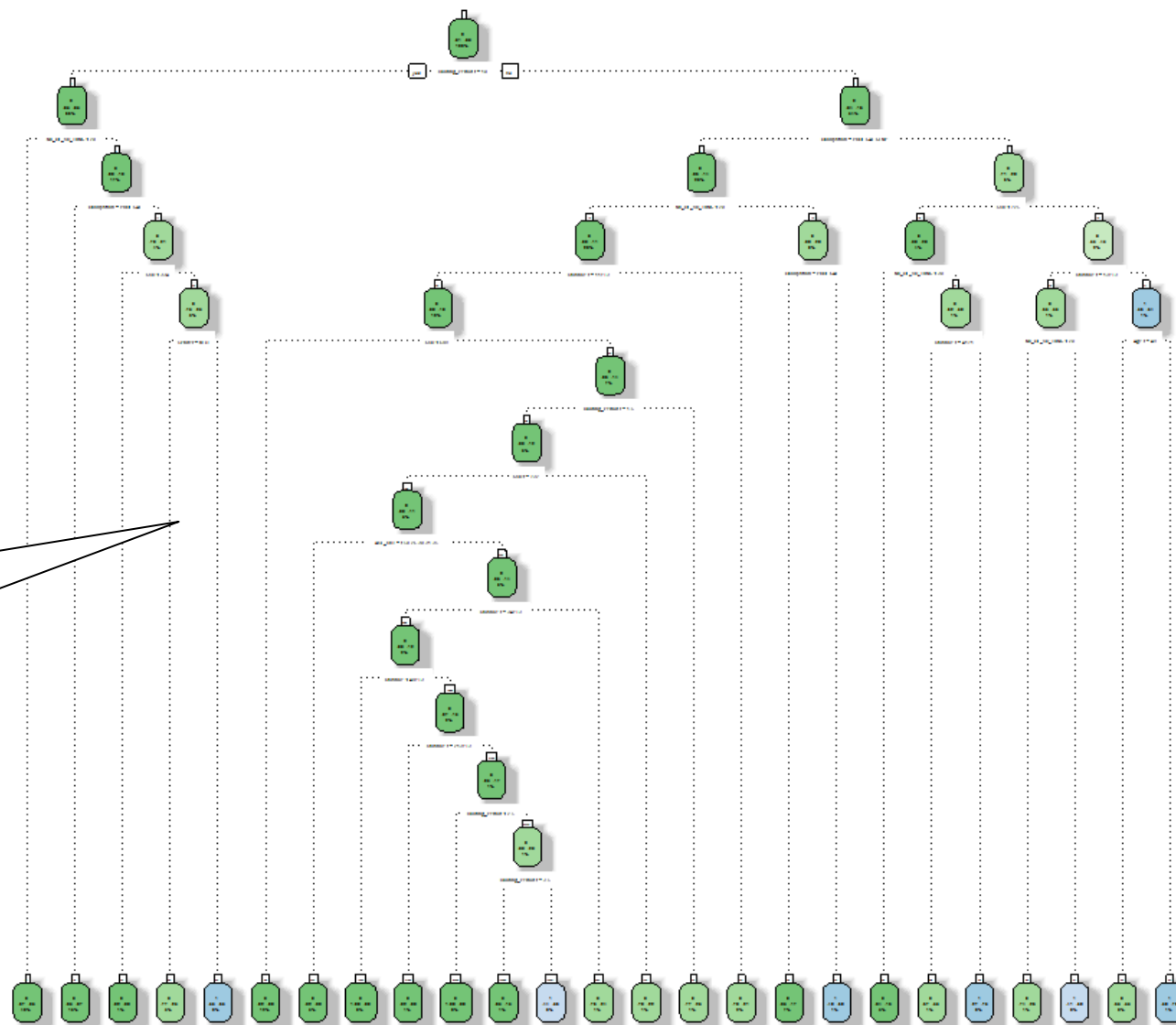
# Plotting the Classification Tree

```
library(rattle)
```

```
library(RColorBrewer)
```

```
fancyRpartPlot(m1)
```

Let us export the  
output to PDF  
format to have a  
clear view of the  
tree



# Concepts | Greedy Algorithm



Make 31 Paise using any combination of above coins

Optimal solution with few coins :  $25 + 5 + 1$

What if the 5 paise coin is not there?

Optimal solution with few coins :  $10 * 3 + 1$

Greedy Algorithm solution:  $25 + 1 * 6$

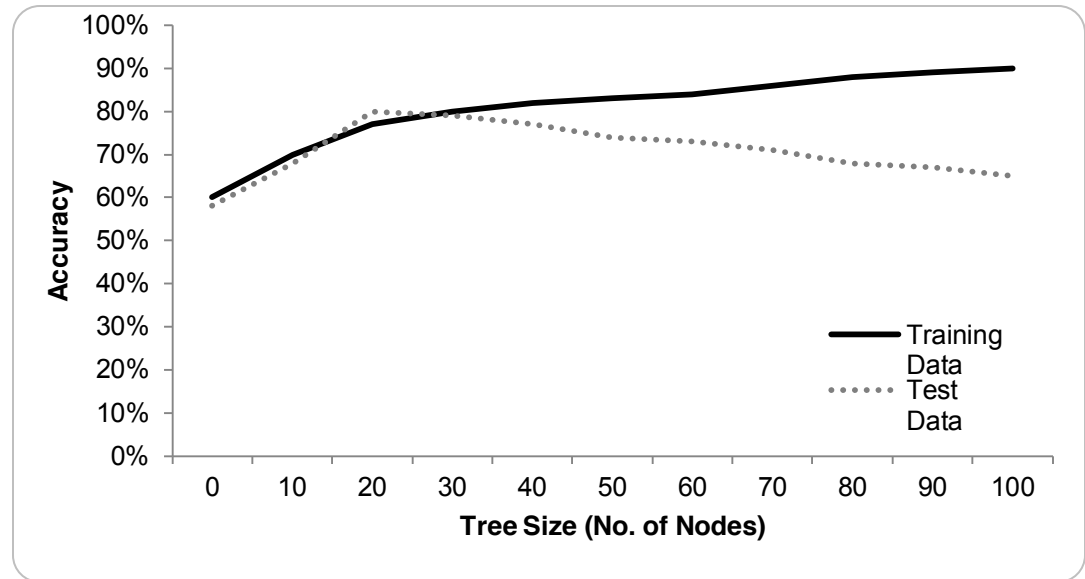
# Concepts | Cross Validation

K Fold CV	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Fold 1	Train	Train	Train	Train	Train	Train	Train	Train	Train	Test
Fold 2	Train	Train	Train	Train	Train	Train	Train	Train	Test	Train
Fold 3	Train	Train	Train	Train	Train	Train	Train	Test	Train	Train
Fold 4	Train	Train	Train	Train	Train	Train	Test	Train	Train	Train
Fold 5	Train	Train	Train	Train	Train	Test	Train	Train	Train	Train
Fold 6	Train	Train	Train	Train	Test	Train	Train	Train	Train	Train
Fold 7	Train	Train	Train	Test	Train	Train	Train	Train	Train	Train
Fold 8	Train	Train	Test	Train	Train	Train	Train	Train	Train	Train
Fold 9	Train	Test	Train	Train	Train	Train	Train	Train	Train	Train
Fold 10	Test	Train	Train	Train	Train	Train	Train	Train	Train	Train

- Cross Validation is part of the CART algorithm
- Method to see how well the model performs to unseen data
- Typically xval parameter for cross-validation is set to 10

# Concepts | Over-fitting

- If you grow the tree too long you will run the risk of over-fitting
- Classification model may not work well on unseen data



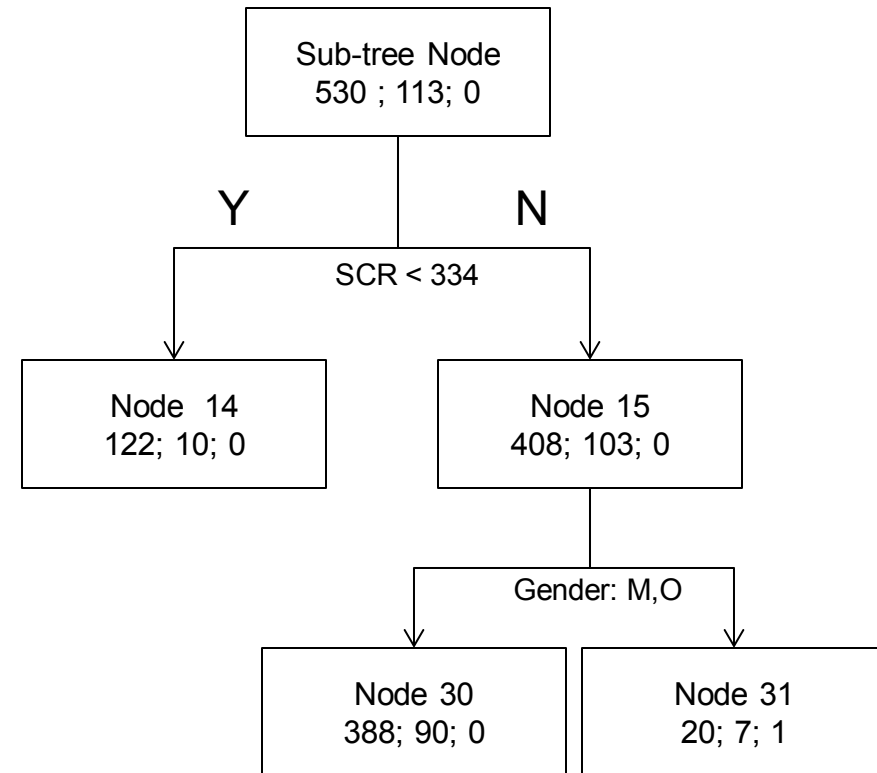
## How do we avoid Over-fitting?

**Stopping Rule:** don't expand a node if the impurity reduction of the best split is below some threshold

**Pruning:** grow a very large tree and merge back nodes

## Concepts | Parsimony Principle & Re-substitution Error

- **Parsimony principle** is basic to all science and tells us to choose the simplest scientific explanation that fits the evidence.
- **Resubstitution Error**: It measures what fraction of the cases in a node is classified incorrectly if we assign every case to the majority class in that node; It always favours large tree
- To counter balance the resubstitution error we need a penalty component that favours smaller tree

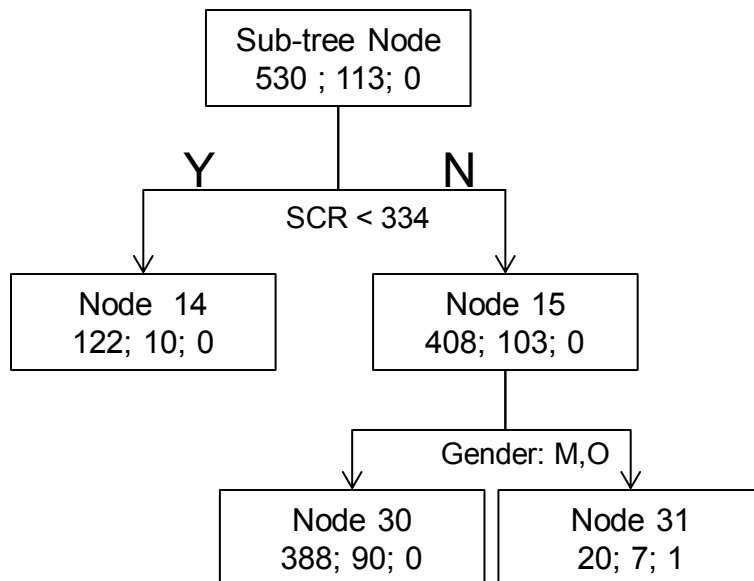


$$\text{Re (pruned)} = 113 / 530$$

$$\text{Re (leaves)} = 107 / 530$$

# Cost Component Pruning

- “cost-complexity” – a measure of avg. error reduced per leaf
- Calculate number of errors for each node if collapsed to leaf
- Compare to errors in leaves, taking into account more nodes used



$$\begin{aligned} \text{Re (pruned)} + 1 \alpha \\ = \text{Re (leaves)} + 3 \alpha \end{aligned}$$

$$113 / 530 + 1 \alpha = 107 / 530 + 3 \alpha$$

$$\alpha = 0.0056$$

# Print Optimal Pruning table based on Complexity Parameter



## to find how the tree performs

```
printcp(m1)
```

```
plotcp(m1)
```

```
> printcp(m1)
```

Classification tree:

```
rpart(formula = Target ~ ., data = CT_DF.dev[, -1], method = "class",  
      control = r.ctl)
```

Variables actually used in tree construction:

```
[1] Age          AGE_BKT          Balance          Gender          Holding_Period No_OF_CR_TXNS  
[7] Occupation    SCR
```

Root node error: 1235/14000 = 0.088214

n= 14000

	CP	nsplit	rel error	xerror	xstd
1	0.00607287	0	1.00000	1.00000	0.027171
2	0.00404858	7	0.95223	0.98138	0.026941
3	0.00202429	8	0.94818	0.98057	0.026931
4	0.00121457	10	0.94413	0.97895	0.026911
5	0.00016194	14	0.93927	0.99514	0.027112
6	0.00000000	24	0.93765	0.99676	0.027132



# Pruning criteria based on cp table

	CP	nsplit	rel error	xerror	xstd
1	0.00607287	0	1.00000	1.00000	0.027171
2	0.00404858	7	0.95223	0.98138	0.026941
3	0.00202429	8	0.94818	0.98057	0.026931
4	0.00121457	10	0.94413	0.97895	0.026911
5	0.00016194	14	0.93927	0.99514	0.027112
6	0.00000000	24	0.93765	0.99676	0.027132

- xerror – Prune the tree at cp where xerror is minimum
- 1 SE Rule : Look form minimum xerror and then
- ... or probably use business rule to decide the number of nodes

## Pruning Code

```
ptree<- prune(m1, cp= 0.0015,"CP")
```

```
printcp(ptree)
```

```
fancyRpartPlot(ptree, uniform=TRUE, main="Pruned Classification Tree")
```

# 1 SE rule example

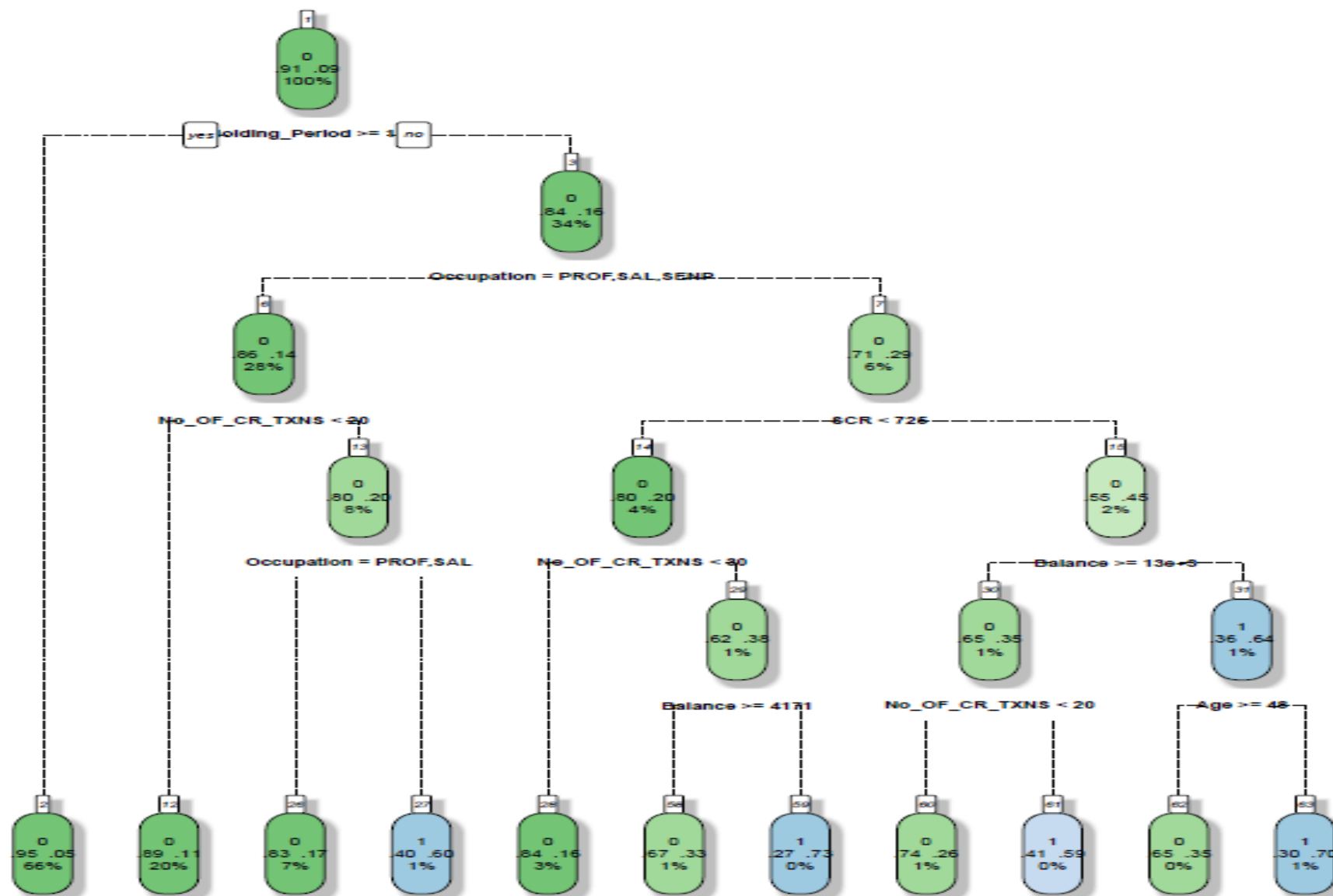
	CP	nsplit	rel error	xerror	xstd
1	0.161992664	0	1.0000000	1.0002790	0.01853630
2	0.043985638	1	0.8380073	0.8385070	0.01749290
3	0.030278222	2	0.7940217	0.7963870	0.01709283
4	0.013881619	3	0.7637435	0.7695997	0.01653832
5	0.010181164	4	0.7498619	0.7560406	0.01606136
6	0.008004043	5	0.7396807	0.7466449	0.015600352
7	0.007026176	6	0.7316767	0.7356289	0.01549501
8	0.006614587	8	0.7176243	0.7388091	0.01559568
9	0.005312278	10	0.7043951	0.7254237	0.01522645
10	0.004883811	11	0.6990828	0.7248227	0.01526605

e.g. taken for explanation purpose only

- Based on xerror criteria we will stop at row 10
- Using column xstd, that would suggest using  $0.7248227 + 1 \times 0.01526605 = 0.7400887$  and thus pruning should occur at row 7

<http://stats.stackexchange.com/questions/92547/r-part-cross-validation-and-1-se-rule-why-is-the-column-in-cptable-called-xst>  
<https://stats.stackexchange.com/questions/13471/how-to-choose-the-number-of-splits-in-rpart>

# Pruned Classification Tree

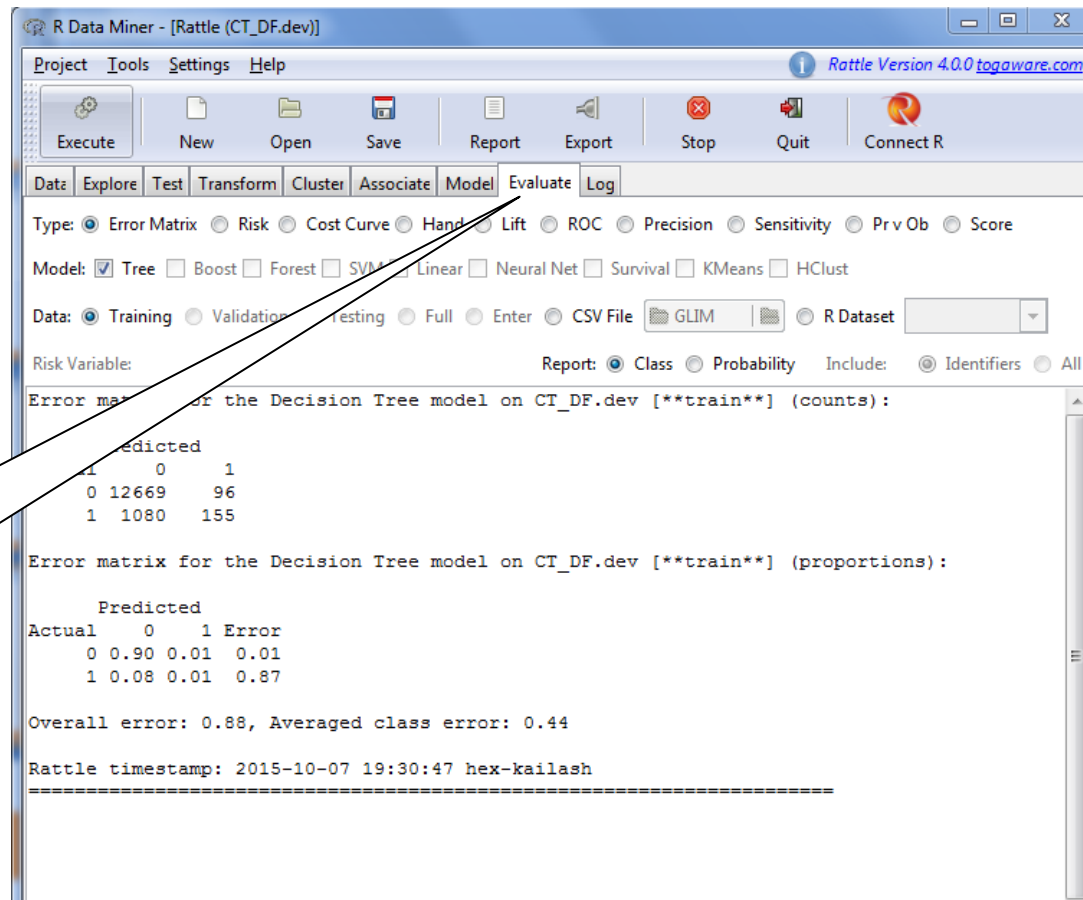


# Model Evaluation

Various measures to see the model performance

- Error Matrix
- Gini Coefficient
- AUC
- KS
- Lift Chart

Demo of Rattle interface to build model and generate various model evaluation measures



<https://www.youtube.com/watch?v=OAI6eAyP-yo>

# Just in case you are interested in coding... 😊😊😊



```
## scoring step
CTDF.dev$predict.score <- predict(m1, CTDF.dev)

## deciling code
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
    ifelse(x<deciles[2], 2,
    ifelse(x<deciles[3], 3,
    ifelse(x<deciles[4], 4,
    ifelse(x<deciles[5], 5,
    ifelse(x<deciles[6], 6,
    ifelse(x<deciles[7], 7,
    ifelse(x<deciles[8], 8,
    ifelse(x<deciles[9], 9, 10
    ))))))))
}

## deciling
CTDF.dev$deciles <- decile(CTDF.dev$predict.score[,2])
```

# Some coding continued... 😊😊😊

```
## Ranking code
library(data.table)
tmp_DT = data.table(CTDF.dev)
rank <- tmp_DT[, list(
  cnt = length(Target),
  cnt_resp = sum(Target),
  cnt_non_resp = sum(Target == 0) ,
  by=deciles][order(deciles)]
rank$rrate <- rank$cnt_resp * 100 / rank$cnt;
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_rel_resp <- rank$cum_resp / sum(rank$cnt_resp);
rank$cum_rel_non_resp <- rank$cum_non_resp /
sum(rank$cnt_non_resp);
rank$ks <- abs(rank$cum_rel_resp -
rank$cum_rel_non_resp);
rank
```

```
> with(CTDF.dev, table(Target, predict.class))
```

	predict.class	
Target	0	1
0	12663	102
1	1056	179

**Mis-Class = 8.3%**

```
> auc
[1] 0.7578043
> KS
[1] 0.4009363
> gini
[1] 0.4701245
```

```
## Plotting ROC Curve and AUC
library(ROCR)
pred <- prediction(CTDF.dev$predict.score[,2],
CTDF.dev$Target)
perf <- performance(pred, "tpr", "fpr")
plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
```

```
## Computing Gini Index
library(ineq)
gini = ineq(CTDF.dev$predict.score[,2], type="Gini")
```

```
## Output all the values
with(CTDF.dev, table(Target, predict.class))
auc
KS
gini
```



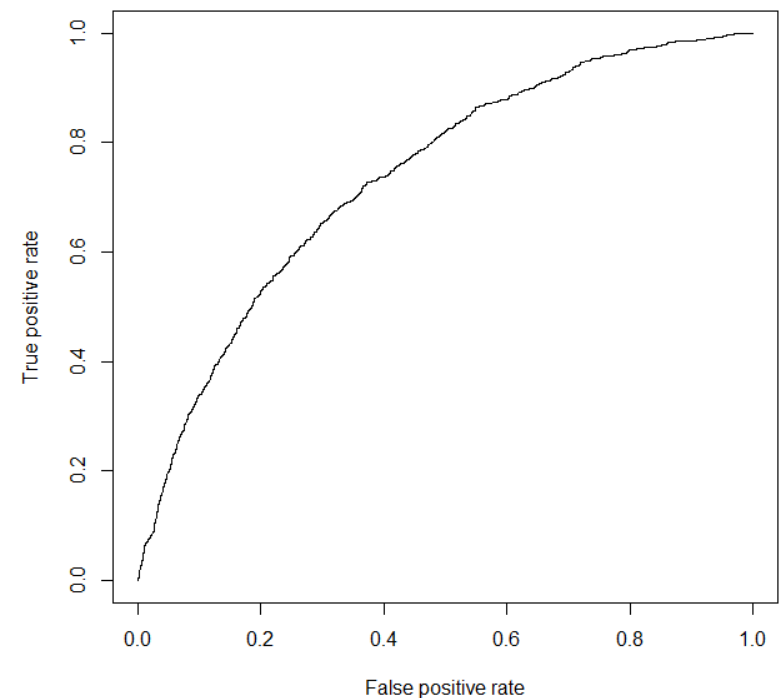
# Area Under Curve

Classification Matrix		Predicted	
		Y	N
Actual	Y	a	b
	N	c	d

Sensitivity = True Positive Rate  
= True Positive / Total Positive  
=  $a / (a + b)$

Specificity = True Negative / Total Negative  
=  $d / (c + d)$

False Positive Rate =  $1 - \text{Specificity}$



# Scoring

## Syntax to get the node path

```
tree.path <- path.rpart (ptree, node = c(2, 12))
```

## Scoring syntax

```
CTDF.dev$predict.class <- predict(m1, CTDF.dev, type="class")
```

```
CTDF.dev$predict.score <- predict(m1, CTDF.dev)
```

## We can use the above syntax for scoring the Hold Out Sample also

```
CTDF.holdout$predict.class <- predict(m1, CTDF.holdout, type="class")
```

```
CTDF.holdout$predict.score <- predict(m1, CTDF.holdout)
```

## Checking performance of model on Hold Out Sample

```
with(CTDF.holdout, table(Target, predict.class))
```



```
> with(CTDF.holdout, table(Target, predict.class))
```

	predict.class	
Target	0	1
0	5454	48
1	443	55

Mis-Class = 8.2%

What to ensure when scoring a new dataset?

The **predictor variables** which are included in the final classification model should be present in the dataset to be scored





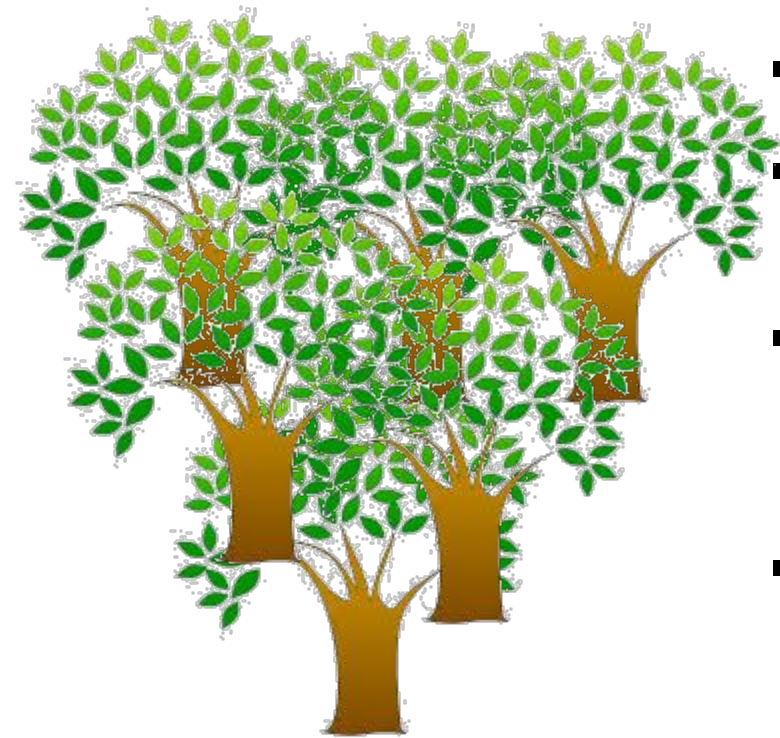
**K2 Analytics**  
Building Skills, Building Individuals

***Random Forest***

# Some Concepts

- **Ensemble** : use of *multiple learning algorithms* to obtain better *predictive performance* than could be obtained from any of the constituent learning algorithms
- **Bootstrap aggregating**, also called **bagging**: Given a standard training set  $D$  of size  $n$ , bagging generates  $m$  new training sets  $D_i$ , each of size  $n'$ , by sampling from  $D$  uniformly with replacement. By sampling with replacement, some observations may be repeated in each  $D_i$

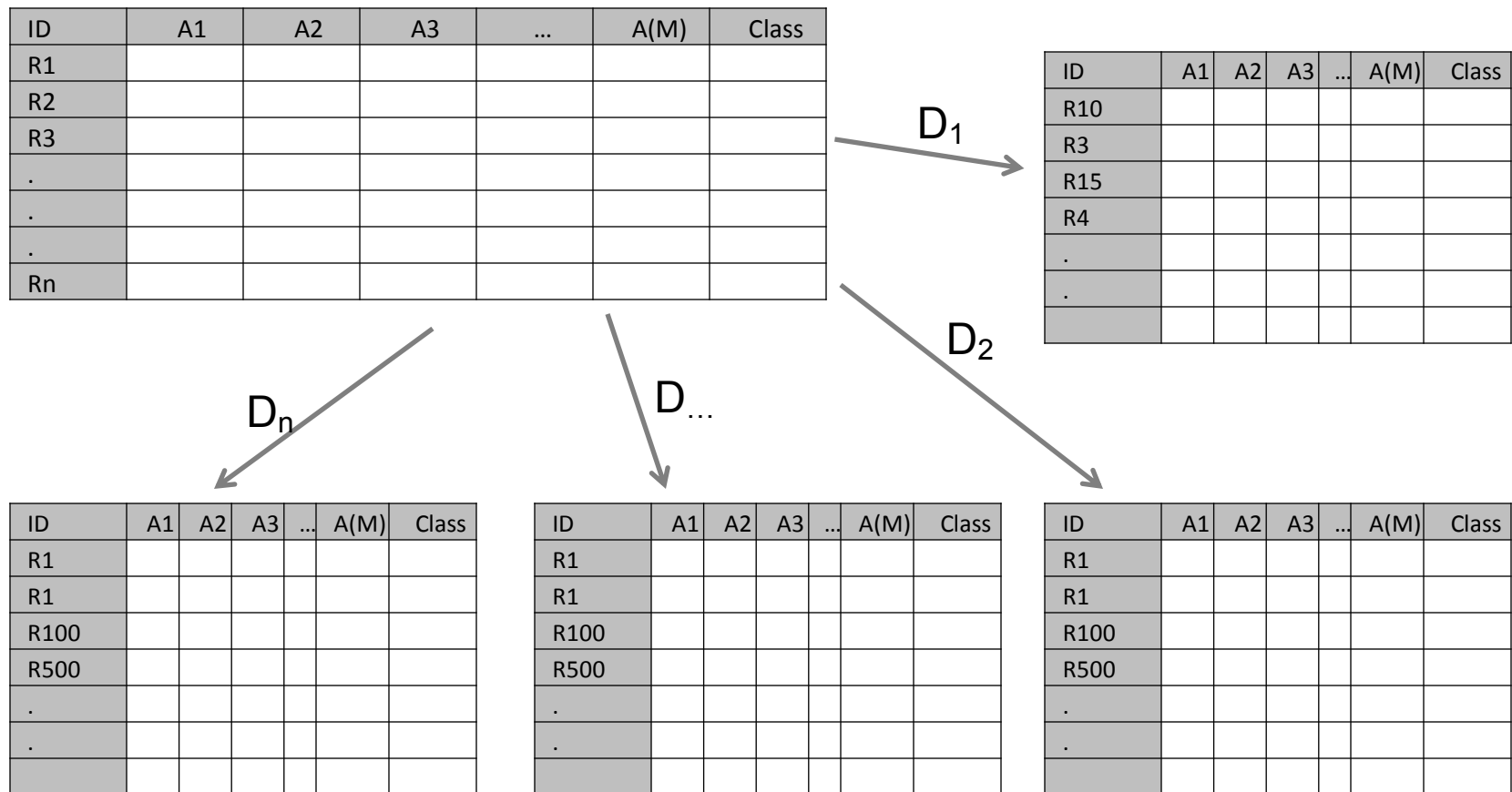
# Random Forest



- Ensemble Technique
- Involves constructing multitude of decision trees at training time
- Prediction is based on mode for classification tree and mean for regression tree
- Help reduce over-fitting
  - Note: there is possibility of high over-fitting at individual tree level but averaging removes the bias

# RF Algorithm

- Step 1: Random Sampling with replacement



# RF Algorithm... contd

- Step 2: Building the tree for each sample with only partial set of 'm' variable being considered at each node
- $m \ll M$  where M is total number of predictor variables

ID	A1	A5	A7	Class
R10				
R3				
R15				
R4				
.				
.				
.				

Only a partial list of variables are considered for splitting based on the best variable from the partial list

ID	A2	A6	A9	Class
R10				
R3				
R15				
R4				
.				
.				
.				

A different set of partial list of variables considered

ID	A1	A3	A4	Class
R10				
R3				
R15				
R4				
.				
.				
.				

A different set of partial list of variables considered

# RF Algorithm... contd

## Step 3: Classifying

- Based on 'n' samples... 'n' tree are built
- Each records is classified based on the n tree
- Final class for each record is decided based on voting

**Note: We do not have the pruning step in RF**

**Some original papers on RF proved that the RF error rate depends on two factors**

1. The *correlation* between any two trees in the forest. Increasing the correlation increases the forest error rate.
2. The *strength* of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.
3. Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide

[https://www.stat.berkeley.edu/~breiman/RandomForests/cc\\_home.htm](https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm)

# Building Random Forest in R

```
## Building the model using Random Forest
```

```
## importing the data
```

```
RFDF.dev <- read.table("datafile/DEV_SAMPLE.csv", sep = ",", header = T)
```

```
RFDF.holdout <- read.table("datafile/HOLDOUT_SAMPLE.csv", sep = ",", header = T)
```

```
c(nrow(RFDF.dev), nrow(RFDF.holdout))
```

```
##install.packages("randomForest")
```

```
library(randomForest)
```

```
## Calling syntax to build the Random Forest
```

```
RF <- randomForest(as.factor(Target) ~ ., data = RFDF.dev[,-1],
```

```
  ntree=100, ## number of trees to be built
```

```
  mtry = 3, ## number of variables randomly sampled as candidate at each split
```

```
  nodesize = 10, ## minimum number of records in terminal node
```

```
  importance=TRUE ) ## should importance of predictors be assessed
```

```
print(RF)
```

# OOB Estimate of error rate

```
> print(RF)

Call:
 randomForest(formula = as.factor(Target) ~ ., data = RFDF.dev[,      -1],
 ntree = 100, mtry = 3, importance = TRUE)
      Type of random forest: classification
      Number of trees: 100
No. of variables tried at each split: 3

OOB estimate of error rate: 8.29%

Confusion matrix:
      0   1 class.error
0 12667  98 0.007677242
1  1063 172 0.860728745
```

RF does not need cross-validation

## OOB Error Rate Computation Steps

- Sample left out (out-of-bag) in  $K^{\text{th}}$  tree is classified using the  $K^{\text{th}}$  tree
- Assume  $j$  cases are mis-classified
- Proportion of time that  $j$  is not equal to true class averaged over all cases is the oob estimate of error rate



# OOB Error Rate ... contd

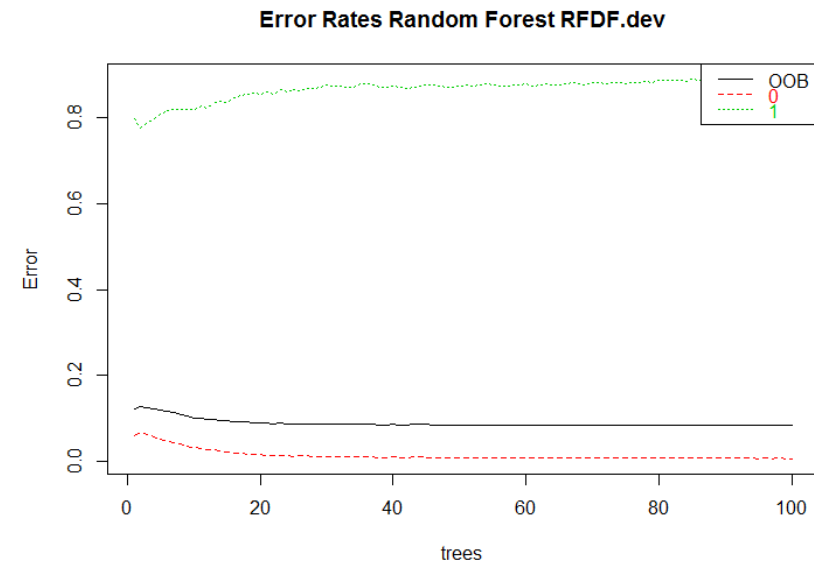
- OOB Estimate of Error Rate is dependent on two key factors
  - nTree
  - Mtry

```
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 RFDF.dev")
```

Tip: One easy way to get these code scripts is to work on rattle and then copy-paste-alter the code generated in its log



# Finding optimal mtry value

```
## Tuning Random Forest
```

```
tRF <- tuneRF(x = RFDF.dev[,-c(1,2)],  
             y=as.factor(RFDF.dev$Target),  
             mtryStart = 3,  
             ntreeTry=100,  
             stepFactor = 1.5,  
             improve = 0.0001,  
             trace=TRUE,  
             plot = TRUE,  
             doBest = TRUE,  
             nodesize = 10,  
             importance=TRUE  
)
```

```
## Parameter Explanation
```

```
## x – predictor variables
```

```
## y – Target Variable
```

```
## mtryStart – starting value of mtry
```

```
## ntreeTry – 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
```

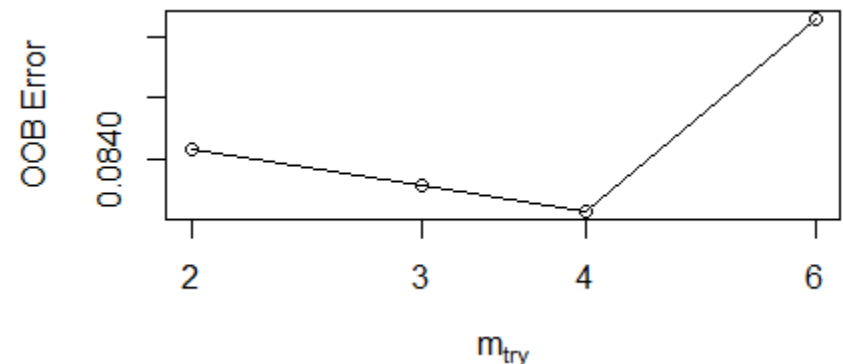
```
## plot – plot OOB vs mtry graph or not
```

```
## doBest – Finally build the RF using optimal mtry
```

```
## nodesize – min terminal node size
```

```
## importance – compute variable importance or not
```

```
mtry = 3 OOB error = 8.38%  
Searching left ...  
mtry = 2 OOB error = 8.41%  
-0.00341006 0.0001  
Searching right ...  
mtry = 4 OOB error = 8.36%  
0.002557545 0.0001  
mtry = 6 OOB error = 8.51%  
-0.01880342 0.0001
```



# Measuring RF Model performance

## Syntax remains same as for the earlier model

## Scoring syntax

```
RFDF.dev$predict.class <- predict(tRF, RFDF.dev, type="class")
```

```
RFDF.dev$predict.score <- predict(tRF, RFDF.dev, type="prob")
```

## deciling

```
RFDF.dev$deciles <- decile(RFDF.dev$predict.score[,2])
```

## Ranking code

```
library(data.table)
```

```
tmp_DT = data.table(RFDF.dev)
```

```
rank <- tmp_DT[, list(  
  cnt = length(Target),  
  cnt_resp = sum(Target),  
  cnt_non_resp = sum(Target == 0)) ,  
  by=deciles][order(deciles)]
```

```
rank$rrate <- rank$cnt_resp * 100 / rank$cnt;
```

```
rank$cum_resp <- cumsum(rank$cnt_resp)
```

```
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
```

```
rank$cum_rel_resp <- rank$cum_resp / sum(rank$cnt_resp);
```

```
rank$cum_rel_non_resp <- rank$cum_non_resp /  
sum(rank$cnt_non_resp);
```

```
rank$ks <- abs(rank$cum_rel_resp - rank$cum_rel_non_resp);
```

```
rank
```

# ...contd

```
## AUC Computation
library(ROCR)
pred <- prediction(RFDF.dev$predict.score[,2],
  RFDF.dev$Target)
perf <- performance(pred, "tpr", "fpr")
plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
```

```
## Gini Compuation
library(ineq)
gini = ineq(RFDF.dev$predict.score[,2], type="Gini")
```

```
## Printing the model performance statistics
with(RFDF.dev, table(Target, predict.class))
auc
KS
gini
```



```
> with(RFDF.dev, table(Target, predict.class))
      predict.class
Target    0      1
  0 12758    7
  1   797  438

> auc
[1] 0.9949836
> KS
[1] 0.9571465
> gini
[1] 0.7276404
```

**Mis-Class = 5.7%**

Compare RF Model  
Performance with  
CART Model

# Hold Out Sample Testing

## Scoring syntax

```
RFDF.holdout$predict.class <- predict(tRF, RFDF.holdout, type="class")
```

```
RFDF.holdout$predict.score <- predict(tRF, RFDF.holdout, type="prob")
```

```
with(RFDF.holdout, table(Target, predict.class))
```



```
> with(RFDF.holdout, table(Target, predict.class))
```

	predict.class	
Target	0	1
0	5474	28
1	437	61

**Mis-Class = 7.8%**

Comparing Mis-Classification Rate of Dev & Hold Out we can say that there is Over-Fitting... however, this mis-classification would have been low if we would have many predictor variables

# Why I like RF technique?

- ... very good technique to pacify Business Users

Variable Category	Variable Name	Variable Description	Variable Name	Variable Description
Txn Mode	no_of_cash_dep_txns_in_mth	Number of cash deposit transactions	tot_cash_dep_amt_in_mth	Total cash deposit amount
	no_of_u_non_cash_or_txns_in_mth	Number of all user initiated non-cash credit (deposit) transactions	tot_u_non_cash_or_amt_in_mth	Total cheque deposit amount
	no_of_chq_or_txns_in_mth	Number of cheque deposit transactions	tot_chq_or_amt_in_mth	Total user initiated non-cash credit (deposit) amount
	no_of_cash_wdl_txns_in_mth	Number of cash withdrawal transactions	tot_cash_wdl_amt_in_mth	Total cash withdrawal amount
	no_of_u_non_cash_dr_txns_in_mth	Number of all user initiated non-cash debit transactions	tot_u_non_cash_dr_amt_in_mth	Total cheque issued amount
	no_of_chq_dr_txns_in_mth	Number of cheque issued transactions	tot_chq_dr_amt_in_mth	Total user initiated non-cash debit amount
Cr/Dr	no_of_cr_txns_in_mth	Number of all credit transactions in month	tot_cr_amt_in_mth	Total Credit Amount in month
	no_of_dr_txns_in_mth	Number of all debit transactions in month	tot_dr_amt_in_mth	Total Debit Amount in month
	no_of_u_cr_txns_in_mth	Number of all user initiated credit transactions	tot_u_cr_amt_in_mth	Total user initiated credit deposit
	no_of_u_dr_txns_in_mth	Number of all user initiated debit transactions	tot_u_dr_amt_in_mth	Total user initiated debit amount
Channel	no_of_atm_cash_wdl_txns_in_mth	Number of ATM cash withdrawal transactions		
	no_of_atm_cash_dep_txns_in_mth	Number of ATM cash deposit transactions		
	no_of_br_cash_wdl_txns_in_mth	Number of Branch cash withdrawal transactions		
	no_of_br_cash_dep_txns_in_mth	Number of Branch cash deposit transactions		
	no_of_atm_chq_dep_txns_in_mth	Number of ATM cheque deposit transactions		
	no_of_atm_cr_txns_in_mth	Number of deposits (Cash or cheque)		
	no_of_br_cr_txns_in_mth	Number of credit transactions		
	no_of_net_cr_txns_in_mth	Number of credits received		
	no_of_net_dr_txns_in_mth	Number of transfers done		
	no_of_br_dr_txns_in_mth	Number of debit transactions		
	no_of_mb_txns_in_mth	Number of Mobile transactions		
	no_of_pb_txns_in_mth	Number of Payments transactions		
	no_of_si_txns_in_mth	Number of Savings transactions		
	no_of_pos_txns_in_mth	Number of POS transactions		
Purpose (Penal Charges)	no_of_aqb_chq_txns_in_mth	Number of AQB cheque transactions		
	no_of_iw_chq_bno_txns_in_mth	Number of inward cheque bounce transactions		
	no_of_ow_chq_bno_txns_in_mth	Number of outward cheque bounce transactions		
Commission & Other Charges				
Purpose of Account				
Source				

• Typically you will have 300 – 500 variables for modeling

• With techniques like Logistic Regression you will be forced to drop variables because of multi-collinearity

• Business users will have their own hypothesis and would want collinear variables to be part of the model

• Ensemble techniques like RF helps you build models by considering multitude of predictor variable permutations

## Challenges

- You do not get a Equation
- Somewhat of Black Box and hence not used in some industries like Banks for Risk Modeling



**K2 Analytics**  
Building Skills, Building Individuals

## ***Classification Technique***

C4.5

## C4.5 | Splitting Criteria

- C4.5 makes use of Entropy and Information Gain as splitting criteria
- Entropy of a Node

$$Entropy(t) = -\sum_j p(j | t) \log p(j | t)$$

(NOTE:  $p(j | t)$  is the relative frequency of class  $j$  at node  $t$ ).

- Entropy of Split Node is computed as Weighted Avg Entropy of each Node at Split Node level
- Gain also called **Information Gain** is Difference of Entropy Parent Node and Entropy Split Node

$$GAIN_{split} = Entropy(p) - \left( \sum_{i=1}^k \frac{n_i}{n} Entropy(i) \right)$$

$n_i$  = number of records at child  $i$ ,  
 $n$  = Total number of records in parent node  $p$



# Information Gain Ratio

- Information Gain tends to prefer large number of partitions, each being small and pure
- Gain Ratio: Adjustment to Information Gain; Penalizes more splits

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO}$$

$$SplitINFO = -\sum_{i=1}^k \frac{n_i}{n} \log \frac{n_i}{n}$$

Parent Node, p is split into k partitions  
 $n_i$  is the number of records in partition i

# R Code

```
m1 <- rpart(formula = Target ~ .,  
            data = CTDF.dev[,-1],  
            method = "class",  
            control = r.ctrl  
  
            parms = list(split = "information")  
            )
```

# C5.0 an extension of C4.5

```
## C50
```

```
##install.packages("C50")
```

```
library(C50)
```

```
C50DF.dev <- read.table("datafile/DEV_SAMPLE.csv", sep = ",", header = T)
```

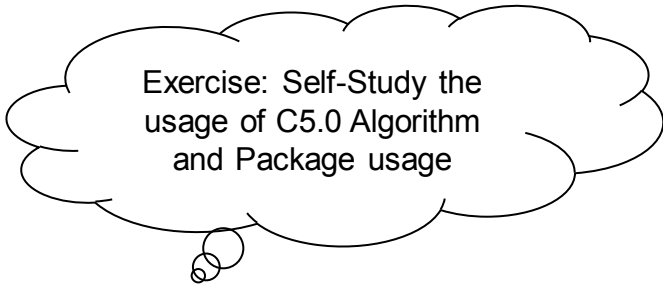
```
C50model <- C5.0(x = C50DF.dev[,-c(1,2)],  
                 y=as.factor(C50DF.dev$Target))
```

```
summary(C50model)
```

```
plot(C50model)
```

```
predict.class <- predict(C50model, C50DF.dev, type="class")
```

```
table(C50DF.dev$Target, predict.class)
```



Exercise: Self-Study the  
usage of C5.0 Algorithm  
and Package usage



**K2 Analytics**  
Building Skills, Building Individuals

**Questions?? ... Thankyou**

**Contact Us**  
**[ar.jakhotia@k2analytics.co.in](mailto:ar.jakhotia@k2analytics.co.in)**