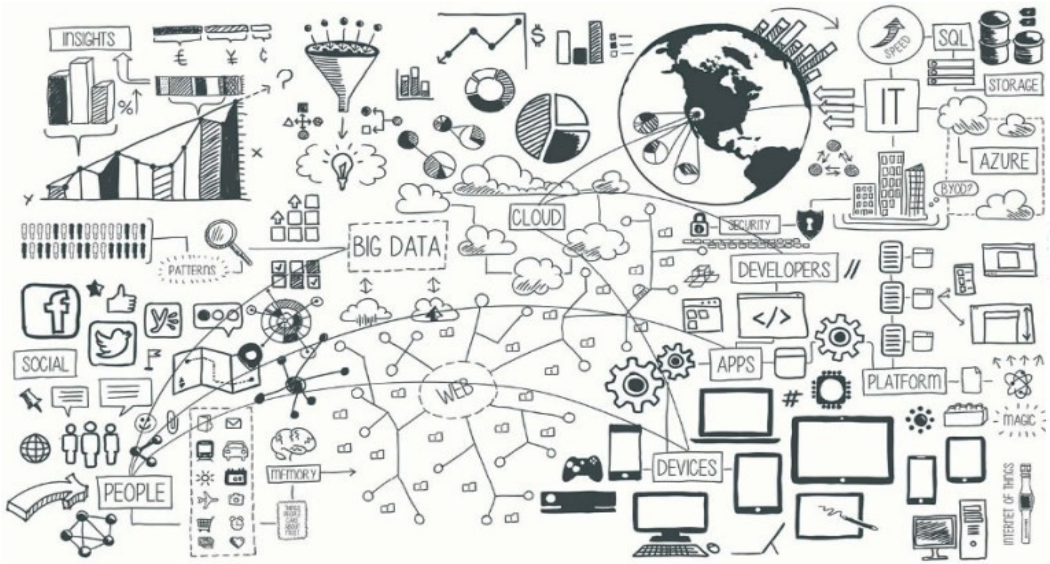
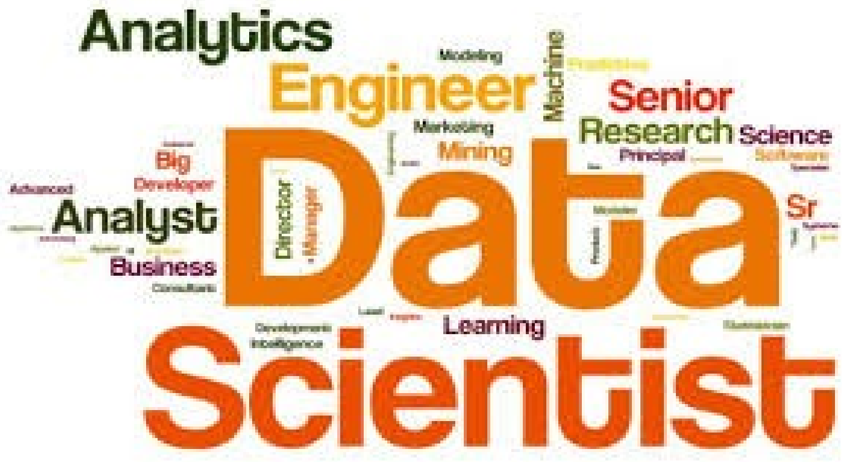
Data Mining (Minería de Datos)

Classification Trees

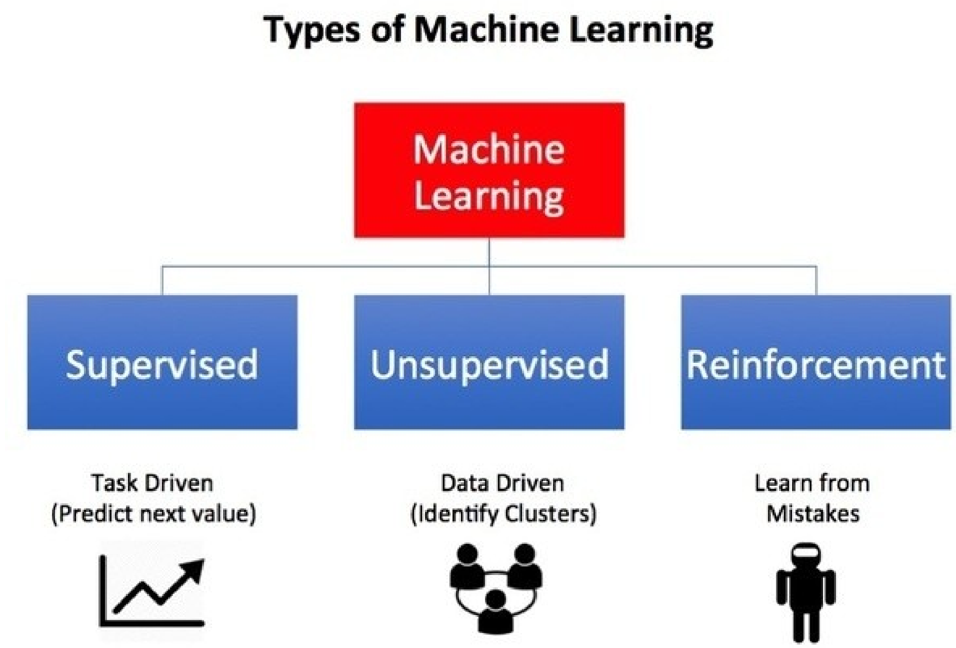




Sixto Herrera, Rodrigo Manzanas

Grupo de Meteorología

Univ. de Cantabria – CSIC
MACC / IFCA

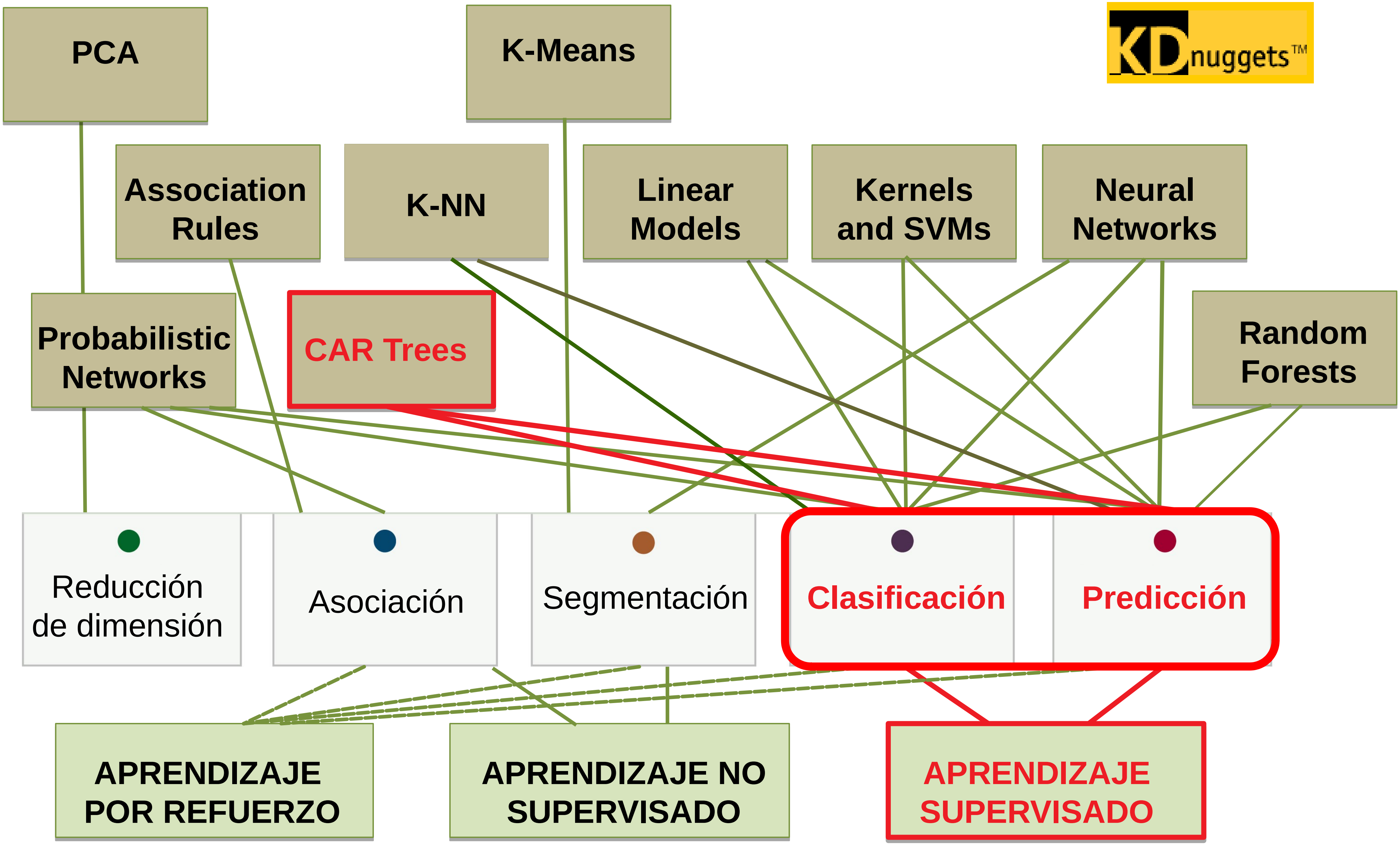


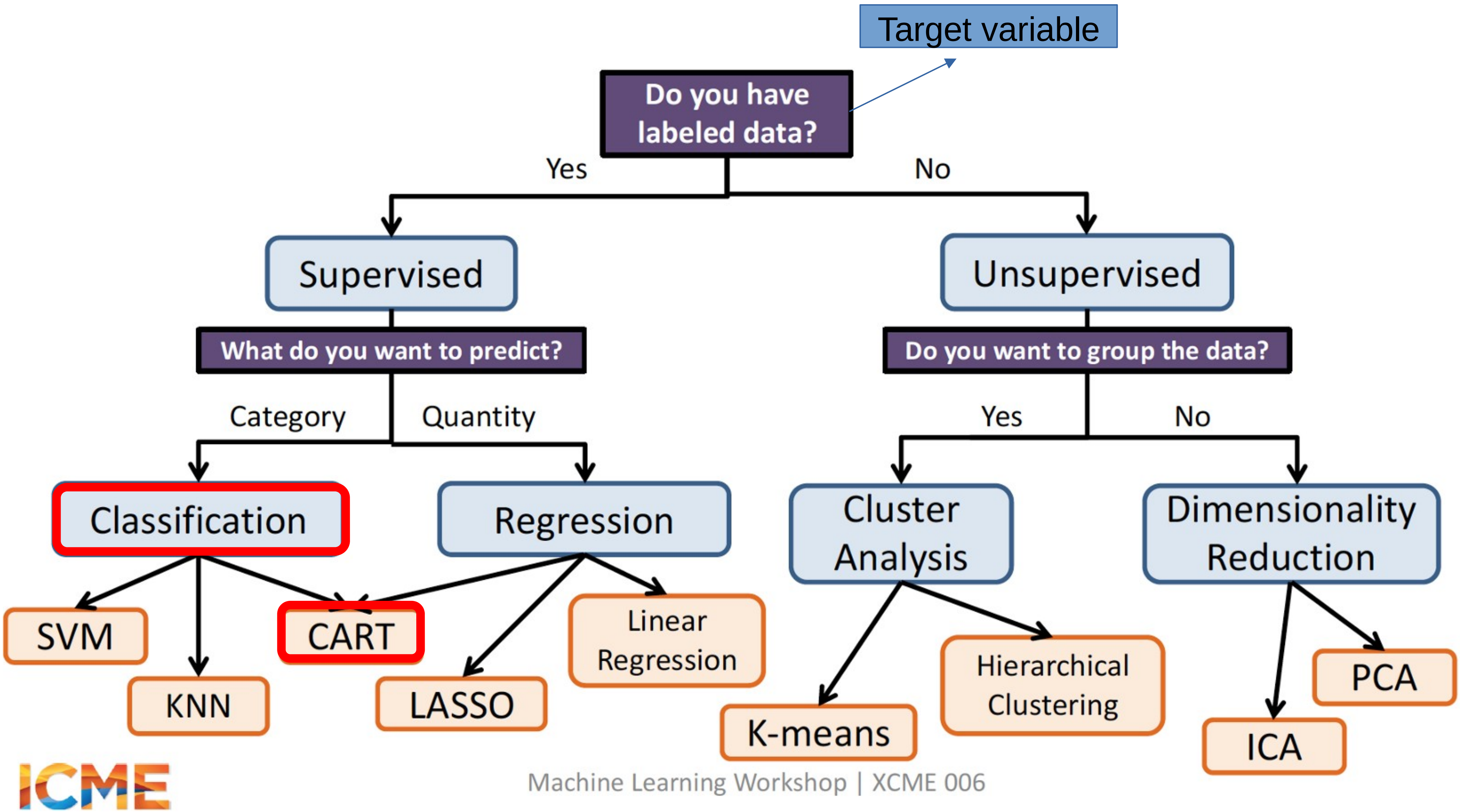
NOTA: Las líneas de código de R en esta presentación se muestran sobre un fondo gris.

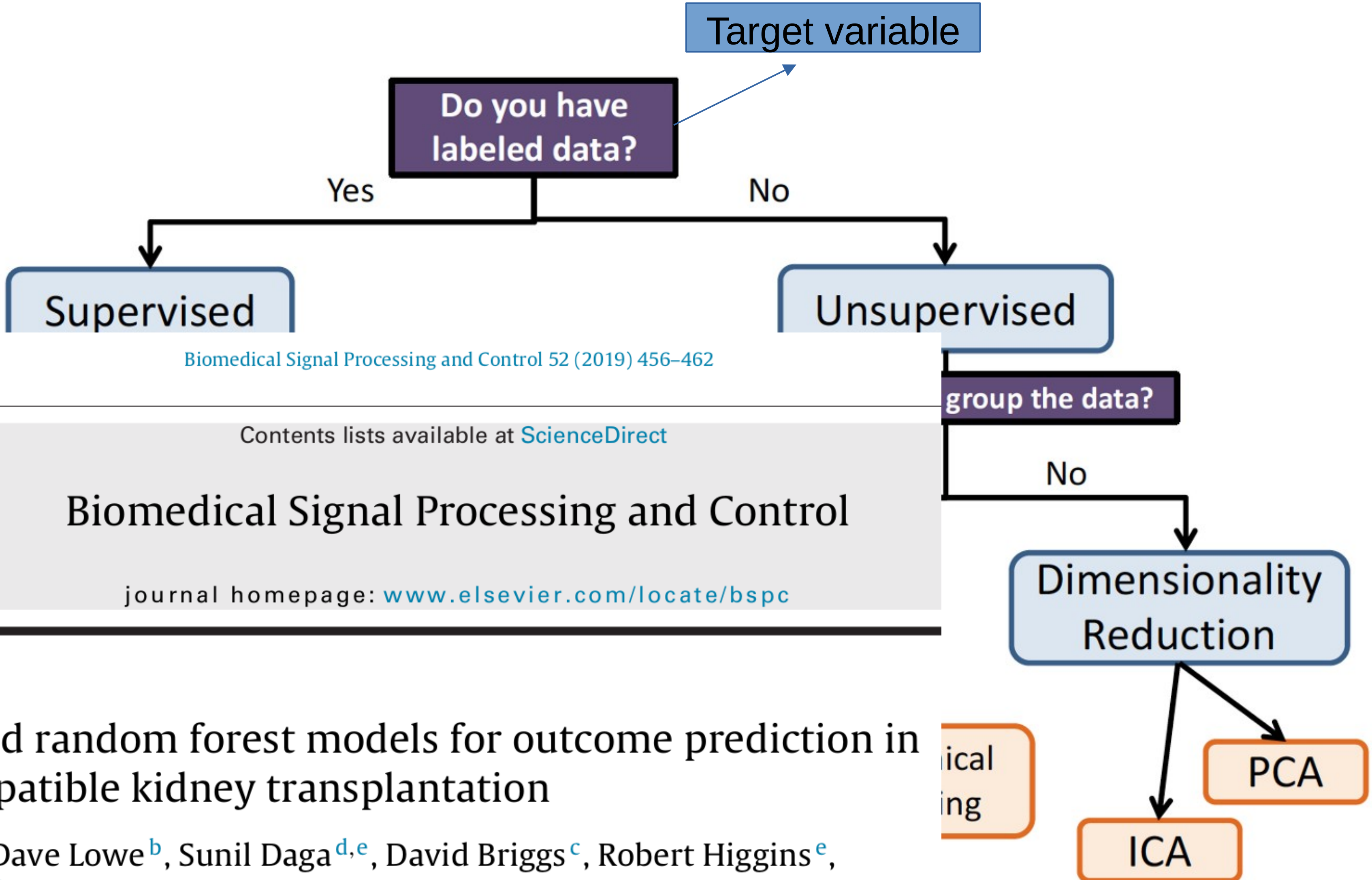
Nov	2	Presentación, introducción y perspectiva histórica
	4	Paradigmas, problemas canonicos y data challenges
	9	Reglas de asociación
	11	Practica: Reglas de asociación
	16	Evaluación, sobrejuste y crossvalidacion
	18	Practica: Crossvalidacion
	23	**Árboles de clasificacion y decision**
	25	**Practica: Árboles de clasificación**
		T01. Datos discretos
	30	Técnicas de vecinos cercano (k-NN)
Dic	2	Práctica: Vecinos cercanos
	9	Comparación de Técnicas de Clasificación.
	14	Reducción de dimensión no lineal
	16	Reducción de dimensión no lineal
		T02. Clasificación
	17	Árboles de clasificación y regresion (CART)
	20	Práctica: Árboles de clasificación y regresion (CART)
	21	Practica: El paquete CARET
		T03. Prediccion
Ene	11	Ensembles: Bagging and Boosting
	13	Random Forests y Gradient boosting
	14	Técnicas de agrupamiento
	20	Técnicas de agrupamiento
	24	Predicción Condicionada
	26	Sesión de refuerzo/repaso.
Ene	27	Examen

PROBLEMS:

https://www.kdnuggets.com/2017/10/top-10-machine-learning-algorithms-beginners.html




Target variable
Do you have labeled data?
Yes
No
Supervised
Unsupervised
What do you want to predict?
Do you want to group the data?
Category
Quantity
Yes
No
Classification
Regression
Cluster Analysis
Dimensionality Reduction
SVM
KNN
CART
LASSO
Linear Regression
K-means
Hierarchical Clustering
PCA
ICA
Machine Learning Workshop | XCME 006



Biomedical Signal Processing and Control 52 (2019) 456–462

Contents lists available at ScienceDirect

Biomedical Signal Processing and Control

journal homepage: www.elsevier.com/locate/bspc

Decision tree and random forest models for outcome prediction in antibody incompatible kidney transplantation

Torgyn Shaikhina[a], Dave Lowe[b], Sunil Daga[d,e], David Briggs[c], Robert Higgins[e], Natasha Khovanova[a,*]

https://doi.org/10.1016/j.bspc.2017.01.012

Classification trees

Aim:

- To classify a **categorical** target variable (R factor) based on a set of **categorical or continuous** predictors

Structure:

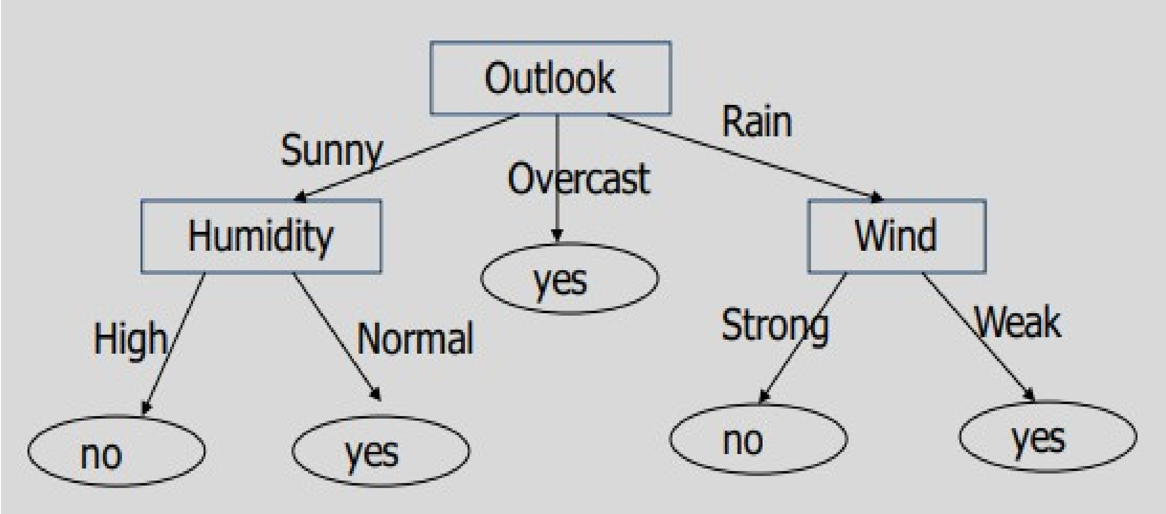
- Each **node** corresponds to a test on an **attribute**
- Each **branch** corresponds to an attribute value
- Each **leaf** (terminal node) represents a final class
- Each **path** is a conjunction of attribute values

Key points:

Due to their intuitive representation, they are easy to assimilate by humans
They can be constructed relatively fast as compared to other methods
In general, they provide as good results as other more complex methods.

PlayTennis dataset:
https://github.com/sjwhitworth/golearn/blob/master/examples/datasets/tennis.csv

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No



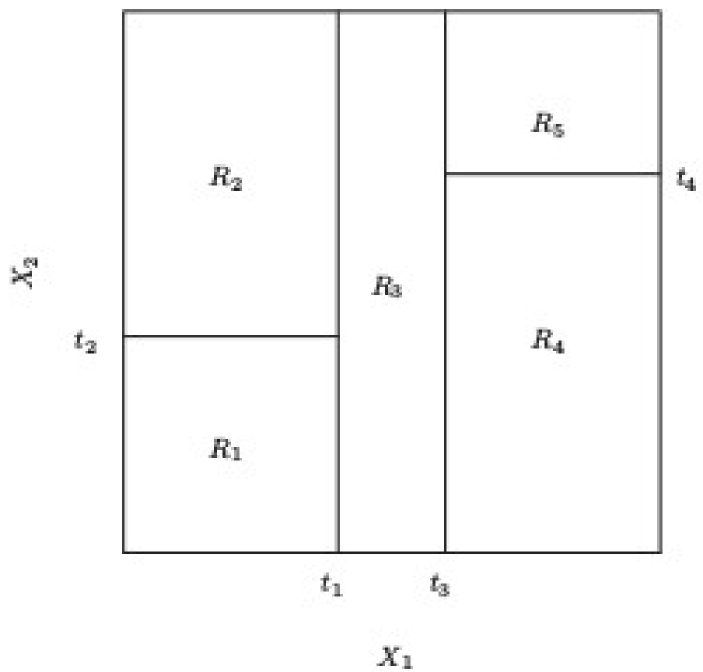
Classification trees

Aim:
- To classify a **categorical** target variable (R factor) based on a set of **categorical or continuous** predictors

Structure:
- Each **node** corresponds to a test on an **attribute**
- Each **branch** corresponds to an attribute value
- Each **leaf** (terminal node) represents a final class
- Each **path** is a conjunction of attribute values

Key points:
Due to their intuitive representation, they are easy to assimilate by humans
They can be constructed relatively fast as compared to other methods
In general, they provide as good results as other more complex methods.

Step 1: *Divide the predictor space (i.e. all the possible values for for X1,X2,…,Xp) into distinct regions:* ***R1, R2,…,Rk.***



Classification trees

Aim:

- To classify a **categorical** target variable (R factor) based on a set of **categorical or continuous** predictors

Structure:

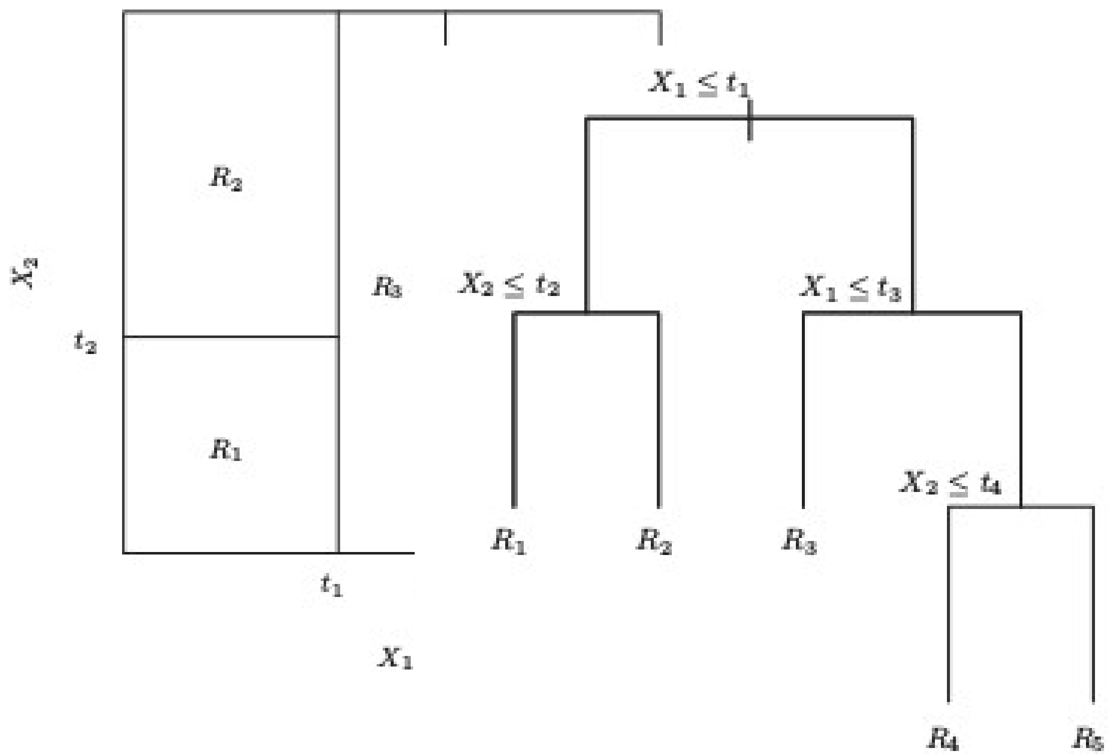
- Each **node** corresponds to a test on an **attribute**
- Each **branch** corresponds to an attribute value
- Each **leaf** (terminal node) represents a final class
- Each **path** is a conjunction of attribute values

Key points:

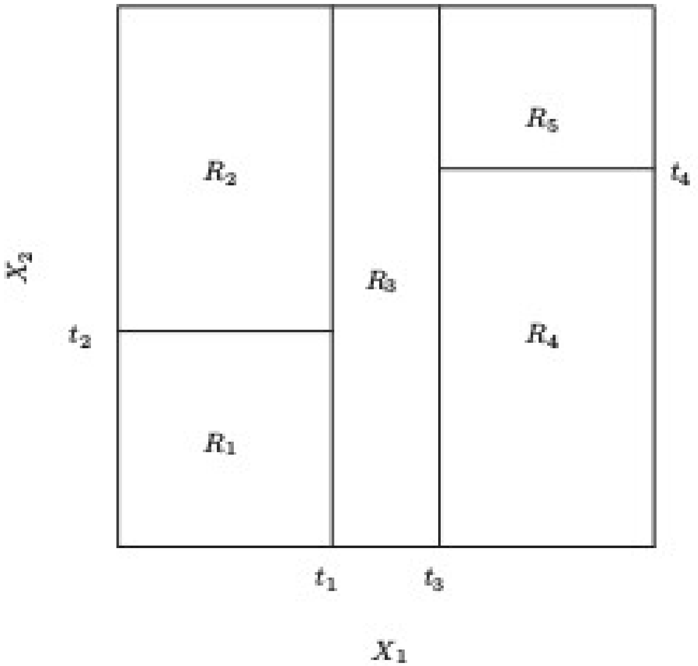
Due to their intuitive representation, they are easy to assimilate by humans
They can be constructed relatively fast as compared to other methods
In general, they provide as good results as other more complex methods.

Step 1: *Divide the predictor space (i.e. all the possible values for for X1,X2,…,Xp) into distinct regions:* ***R1, R2,…,Rk.***

Step 2: *For every X that falls in a particular region (say Rj) we make the same prediction.*

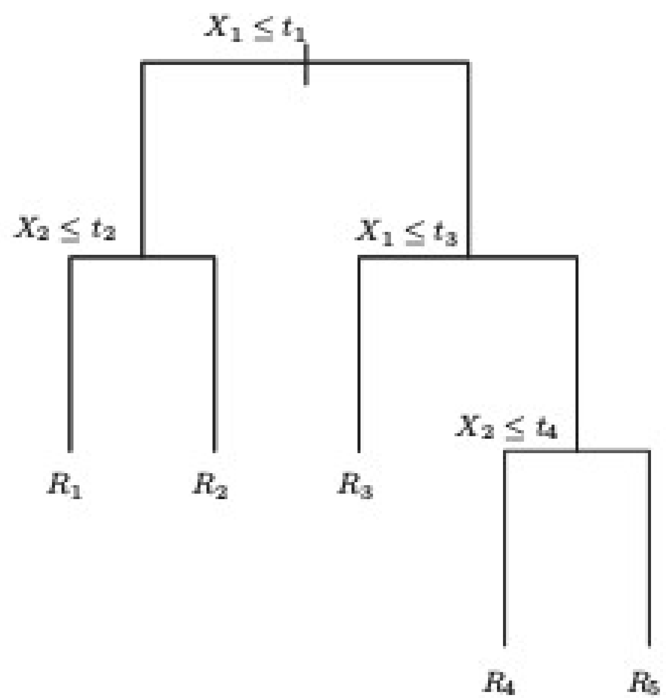


Classification trees



Step 1: *Divide the predictor space (i.e. all the possible values for for X1,X2,…,Xp) into distinct regions:* ***R1, R2,…,Rk.***

Step 2: *For every X that falls in a particular region (say Rj) we make the same prediction.*



Clasification: The most frequent training response value in Rj.

Regression: The mean of the training response in Rj. (**Dec 2020**)

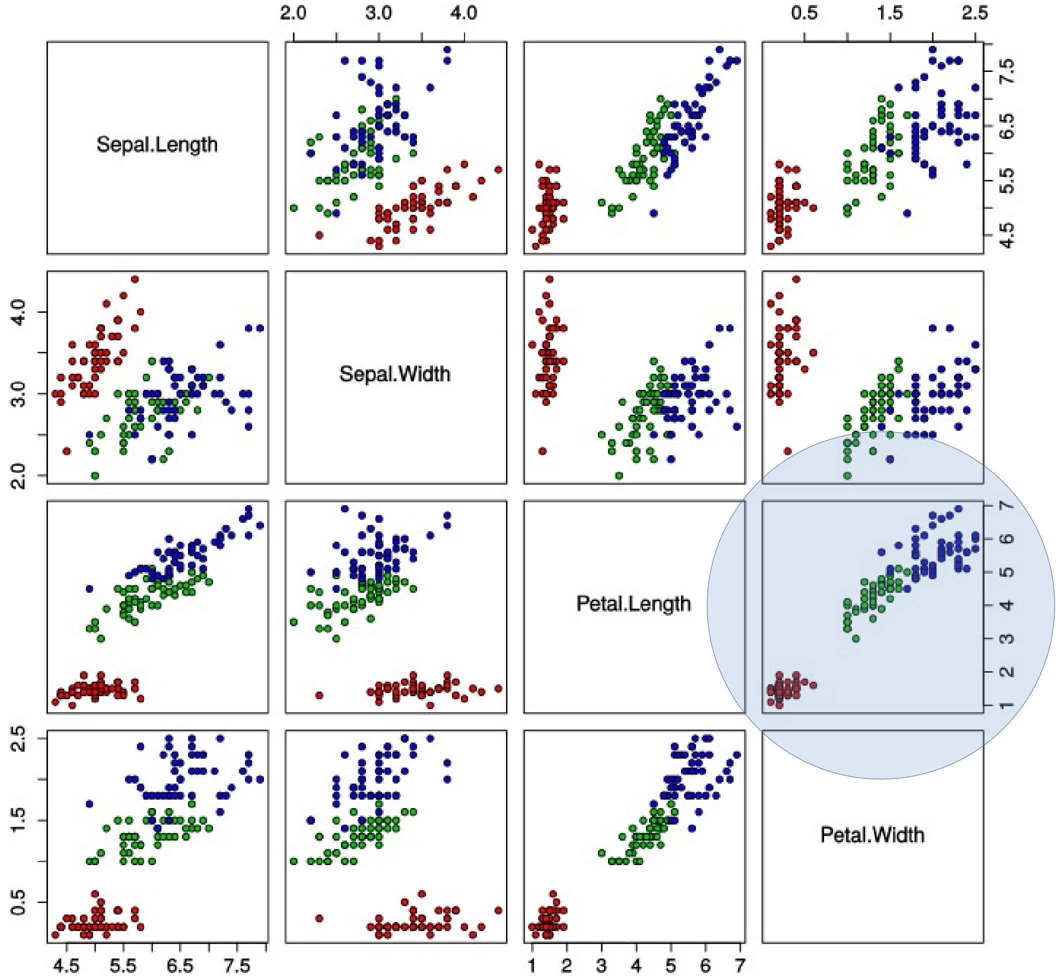
Iris Data Set

Download: Data Folder, Data Set Description

Abstract: Famous database; from Fisher, 1936

Data Set Characteristics:	Multivariate	**Number of Instances:**	150
Attribute Characteristics:	Real	**Number of Attributes:**	4
Associated Tasks:	Classification	**Missing Values?**	No

Iris Data (red=setosa,green=versicolor,blue=virginica)



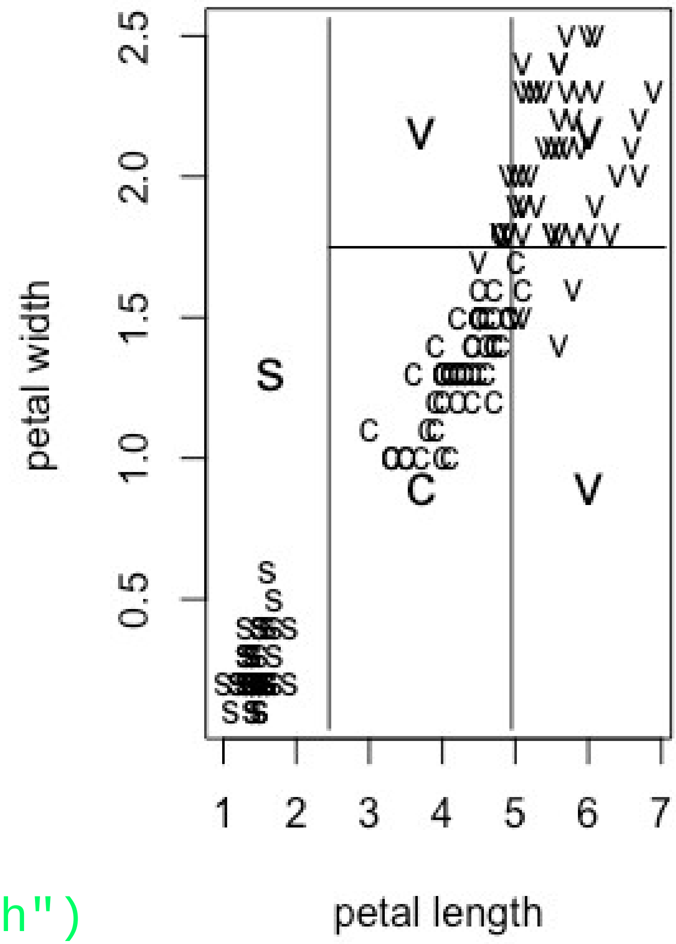
http://archive.ics.uci.edu/ml/datasets/Iris

```
str(iris)

library(ggplot2)

ggplot( data = iris,
  aes(x = Petal.Length,y = Petal.Width)) +
  geom_point(aes(color= Species)) +
  ggtitle("Petal Length Vs Width")
```

```
install.packages("tree"); library(tree)

#Selecting two attributes to visualize the tree
ir <- iris[,c(3,4,5)];  # Petal.Length and Petal.Width

tree <- tree(Species ~., ir); tree
```

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 150 329.600 s ( 0.33 0.33 0.33 )
   2) Petal.Length < 2.45 50   0.000 s ( 1.00 0.00 0.00 ) *
   3) Petal.Length > 2.45 100 138.600 c ( 0.00 0.50 0.50 )
     6) Petal.Width < 1.75 54  33.320 c ( 0.0 0.90 0.10 )
      12) Petal.Length < 4.95 48   9.721 c ( 0.00 0.98 0.02 ) *
      13) Petal.Length > 4.95 6   7.638 v ( 0.00 0.33 0.67 ) *
     7) Petal.Width > 1.75 46   9.635 v ( 0.00 0.021 0.98 )
      14) Petal.Length < 4.95 6   5.407 v ( 0.00 0.16 0.84 ) *
      15) Petal.Length > 4.95 40   0.000 v ( 0.00 0.00 1.00 ) *
```

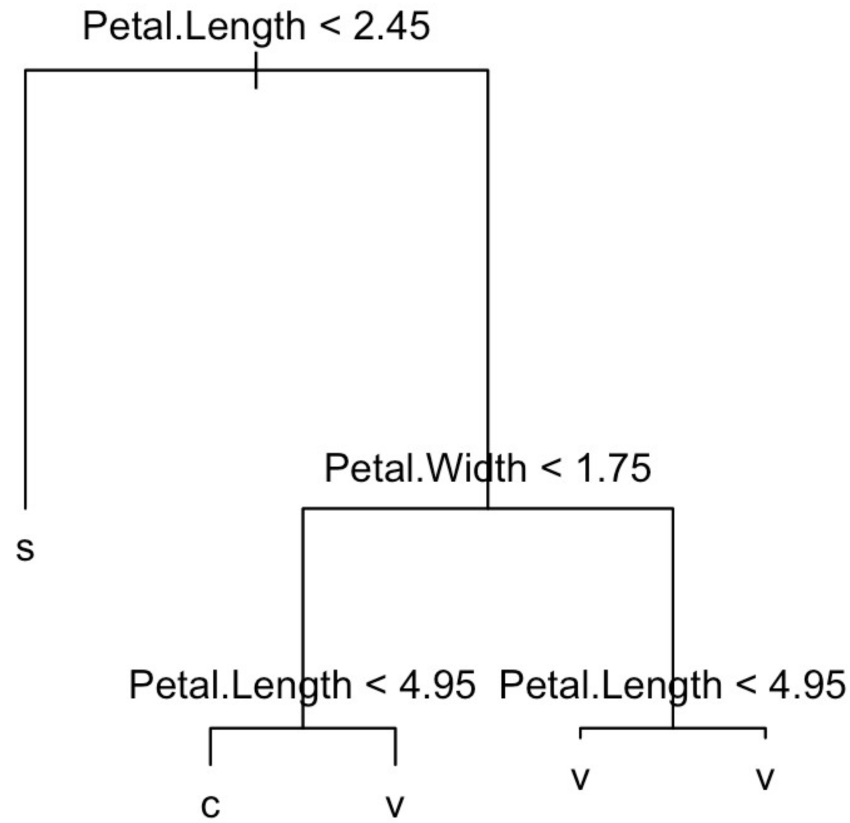
```
partition.tree(tree)

plot(ir[, 1],ir[, 2], type="n",
      xlab="petal length", ylab="petal width")
text(ir[, 1], ir[, 2], c("s", "c", "v")[ir[, 3]])
partition.tree(tree, add = TRUE, cex = 1.5)
```



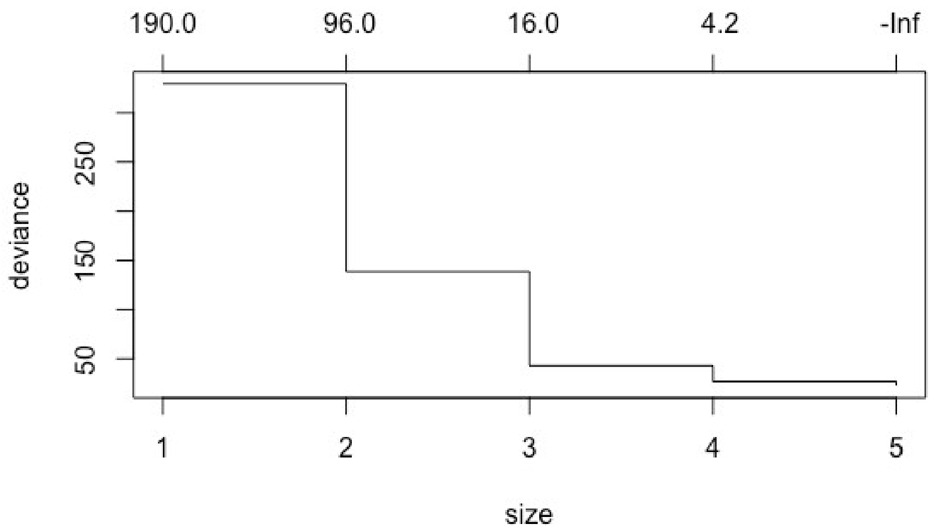
node), split, n, deviance, yval, (yprob)
   * denotes terminal node

```
1) root 150 329.600 s ( 0.33 0.33 0.33 )
  2) Petal.Length < 2.45 50   0.000 s ( 1.00 0.00 0.00 ) *
  3) Petal.Length > 2.45 100 138.600 c ( 0.00 0.50 0.50 )
    6) Petal.Width < 1.75 54  33.320 c ( 0.0 0.90 0.10 )
     12) Petal.Length < 4.95 48   9.721 c ( 0.00 0.98 0.02 ) *
     13) Petal.Length > 4.95 6   7.638 v ( 0.00 0.33 0.67 ) *
    7) Petal.Width > 1.75 46   9.635 v ( 0.00 0.021 0.98 )
     14) Petal.Length < 4.95 6   5.407 v ( 0.00 0.16 0.84 ) *
     15) Petal.Length > 4.95 40   0.000 v ( 0.00 0.00 1.00 ) *
```

```
plot(tree); text(tree)
```




```
pt <- prune.tree(tree); plot(pt)
```



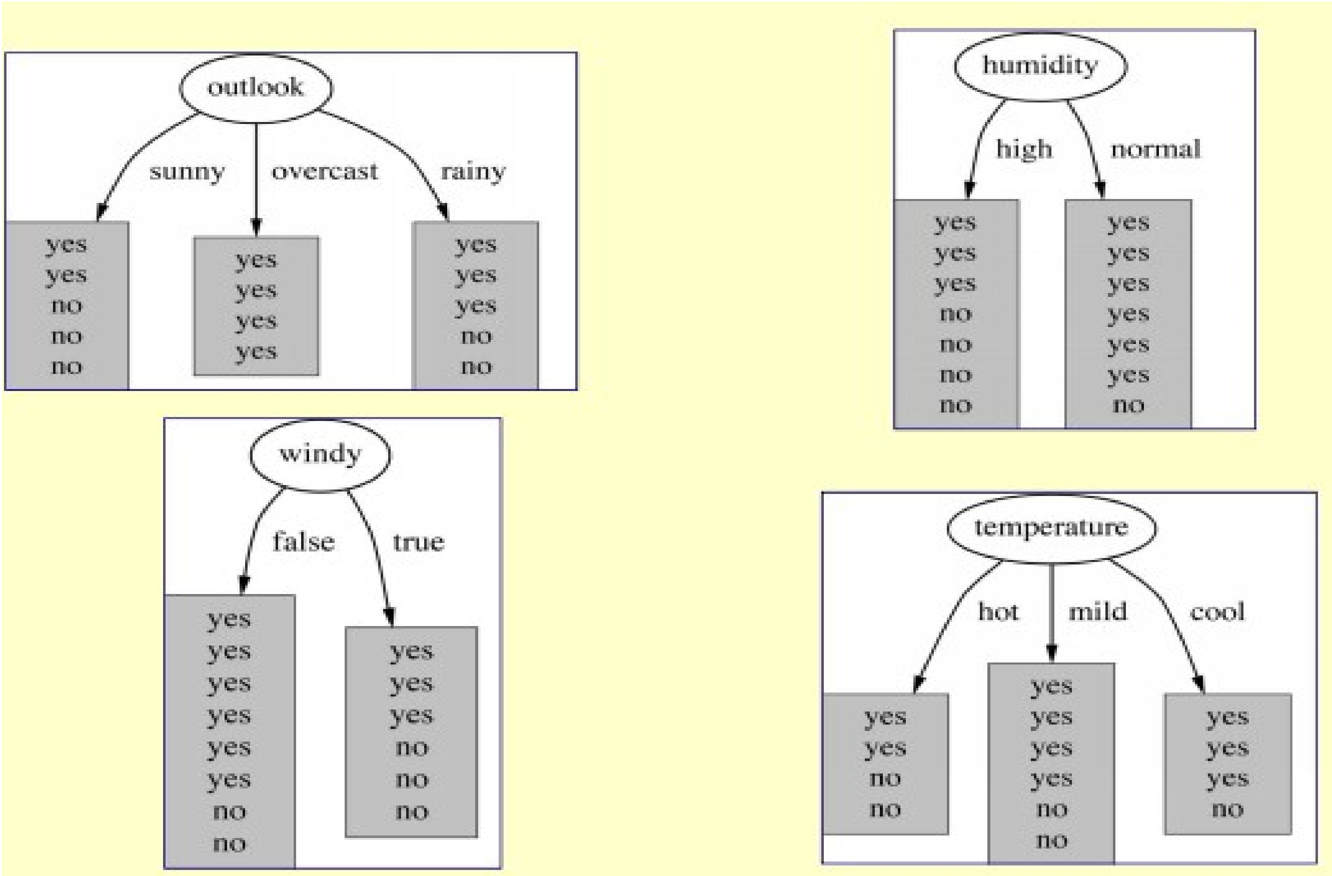

```
summary(tree)
```

Classification tree:
Number of terminal nodes: 5
Residual mean deviance: 0.157 = 22.77 / 145
Misclassification error rate: 0.02667 = 4 / 150

Tree construction

There are several algorithms to build up the tree. However, the idea of all of them is the same: evaluate attribute according to its **power of separation**.

Best discriminating attribute



Many variants for attribute selection:

- from machine learning: **ID3** (**I**terative **D**ichotomizer), **C4.5** and **C5.0** (Quinlan 86, 93)
- from statistics: **CART** (**C**lassification **A**nd **R**egression **T**rees) (Breiman et al. 84)
- from pattern recognition: **CHAID** (**CH**i-squared **A**utomated **I**nteraction **D**etection) (Magidson 94)

Their main difference is the criterion followed to perform the division of the node (splitting)

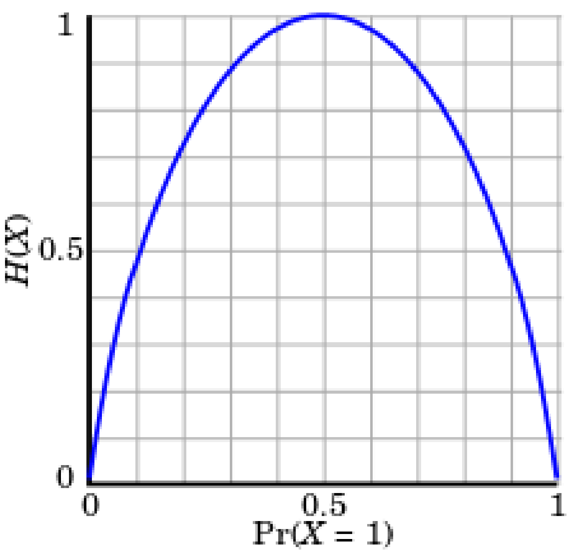
ID3 (the core algorithm)

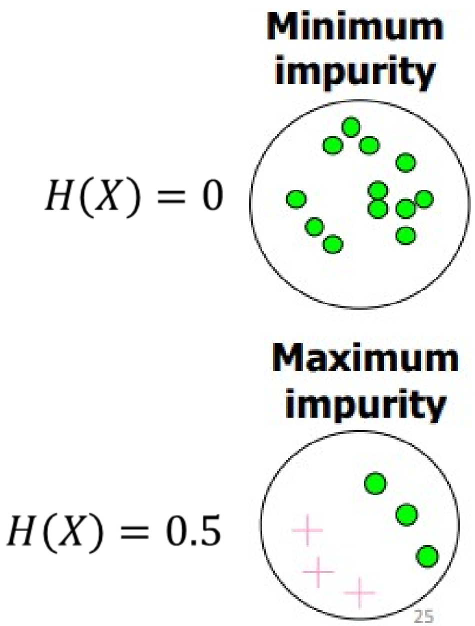
ID3 relies on the **information gain (IG)** to grow the tree. IG measures how important a given attribute is. The goal is to maximize the predictive power of the tree by reducing the uncertainty in the classified data (or **entropy, H)**. H can be seen as a measure of the **purity** of a node.

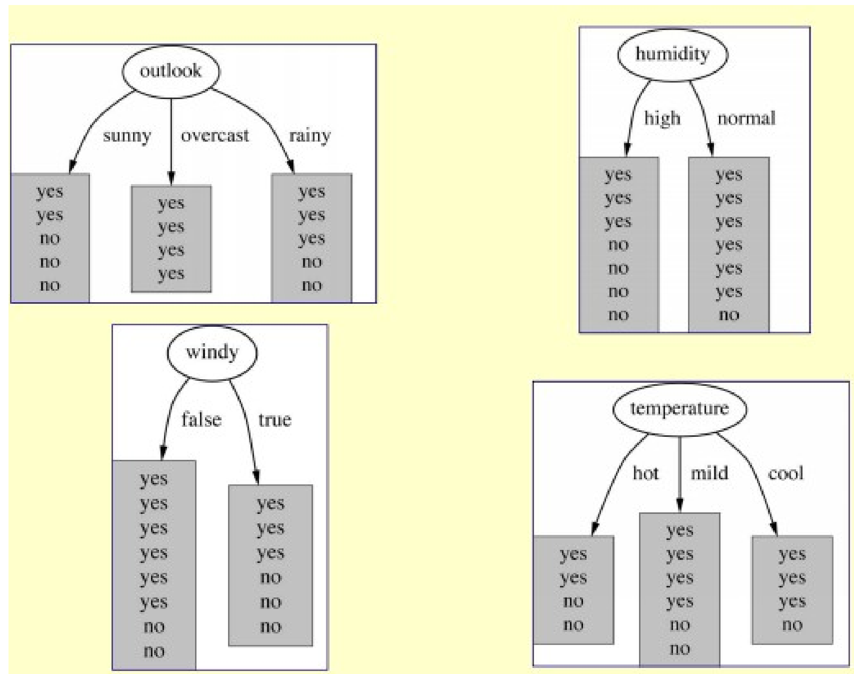
$$IG(X|Y) = H(X) - H(X|Y)$$

$$H(X) = -\sum_{X} p(x)\, log_2(p(x))$$

$$H(X|Y) = -\sum_{X}\sum_{Y} p(x,y)\, log_2(p(x|y))$$







Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$$H(PT) = -\frac{9}{14} log_2\left(\frac{9}{14}\right) - \frac{5}{14} log_2\left(\frac{5}{14}\right) = 0.940$$

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	***Weak***	No
2	Sunny	Hot	High	***Strong***	No
3	Overcast	Hot	High	***Weak***	Yes
4	Rain	Mild	High	***Weak***	Yes
5	Rain	Cool	Normal	***Weak***	Yes
6	Rain	Cool	Normal	***Strong***	No
7	Overcast	Cool	Normal	***Strong***	Yes
8	Sunny	Mild	High	***Weak***	No
9	Sunny	Cool	Normal	***Weak***	Yes
10	Rain	Mild	Normal	***Weak***	Yes
11	Sunny	Mild	Normal	***Strong***	Yes
12	Overcast	Mild	High	***Strong***	Yes
13	Overcast	Hot	Normal	***Weak***	Yes
14	Rain	Mild	High	***Strong***	No

$$H(PT|Wind) = -p(Yes, Strong)\ log_2\big(p(Yes|Strong)\big) - p(No, Strong)\ log_2\big(p(No|Strong)\big)$$
$$-p(Yes, Weak)\ log_2(p(Yes|Weak)) - p(No, Weak)\ log_2(p(No|Weak))$$

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	***Weak***	No
2	Sunny	Hot	High	***Strong***	No
3	Overcast	Hot	High	***Weak***	Yes
4	Rain	Mild	High	***Weak***	Yes
5	Rain	Cool	Normal	***Weak***	Yes
6	Rain	Cool	Normal	***Strong***	No
7	Overcast	Cool	Normal	***Strong***	Yes
8	Sunny	Mild	High	***Weak***	No
9	Sunny	Cool	Normal	***Weak***	Yes
10	Rain	Mild	Normal	***Weak***	Yes
11	Sunny	Mild	Normal	***Strong***	Yes
12	Overcast	Mild	High	***Strong***	Yes
13	Overcast	Hot	Normal	***Weak***	Yes
14	Rain	Mild	High	***Strong***	No

$$H(PT|Wind) = -\frac{3}{14}log_2\left(\frac{3}{6}\right) - \frac{3}{14}log_2\left(\frac{3}{6}\right) - \frac{6}{14}log_2\left(\frac{6}{8}\right) - \frac{2}{14}log_2\left(\frac{2}{8}\right) = 0.892$$

$$H(PT) = -\frac{9}{14}log_2\left(\frac{9}{14}\right) - \frac{5}{14}log_2\left(\frac{5}{14}\right) = 0.940$$

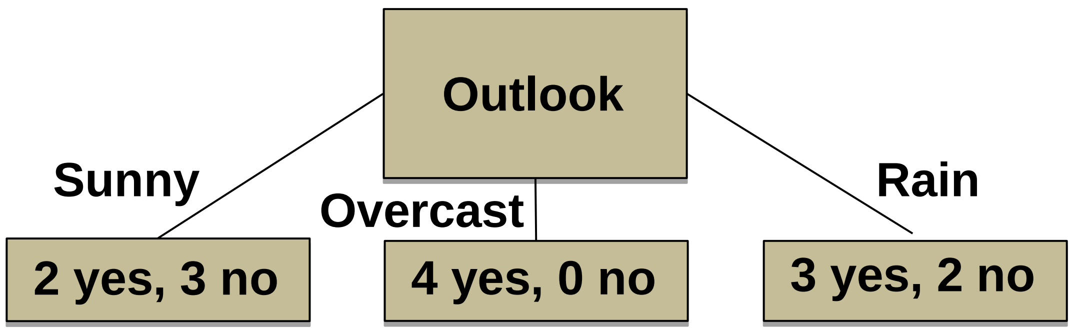
$$H(PT \vee Wind) = -\frac{3}{14}log_2\left(\frac{3}{6}\right) - \frac{3}{14}log_2\left(\frac{3}{14}\right) - \frac{6}{14}log_2\left(\frac{6}{8}\right) - \frac{2}{14}log_2\left(\frac{2}{14}\right) = 0.892$$

$$IG(PT \vee Wind) = 0.940 - 0.892 = 0.048$$

$$IG(PT \vee Humidity) = 0.151$$

$$\boxed{IG(PT \vee Outlook) = 0.246}$$ **Root node**

$$IG(PT \vee Temperature) = 0.029$$



$$H(PT) = -\frac{9}{14}log_2\left(\frac{9}{14}\right) - \frac{5}{14}log_2\left(\frac{5}{14}\right) = 0.940$$

$$H(PT \vee Wind) = -\frac{3}{14}log_2\left(\frac{3}{6}\right) - \frac{3}{14}log_2\left(\frac{3}{14}\right) - \frac{6}{14}log_2\left(\frac{6}{8}\right) - \frac{2}{14}log_2\left(\frac{2}{14}\right) = 0.892$$

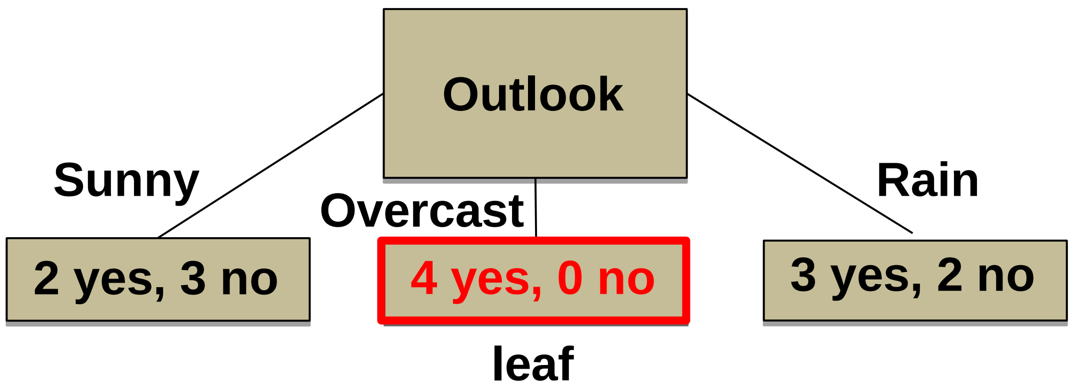
$$IG(PT \vee Wind) = 0.940 - 0.892 = 0.048$$

$$IG(PT \vee Humidity) = 0.151$$

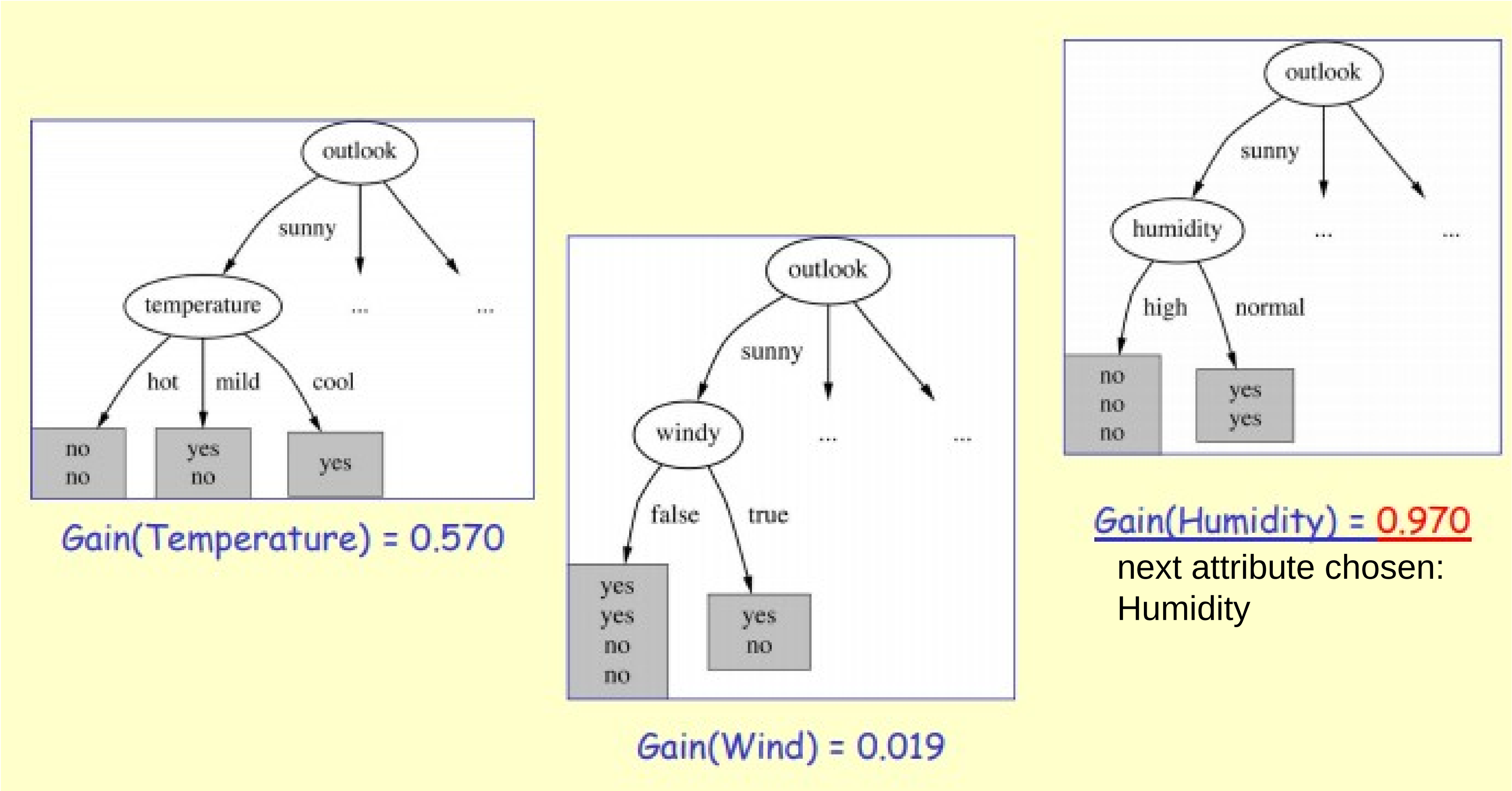
$$\boxed{IG(PT \vee Outlook) = 0.246}$$

Root node

$$IG(PT \vee Temperature) = 0.029$$

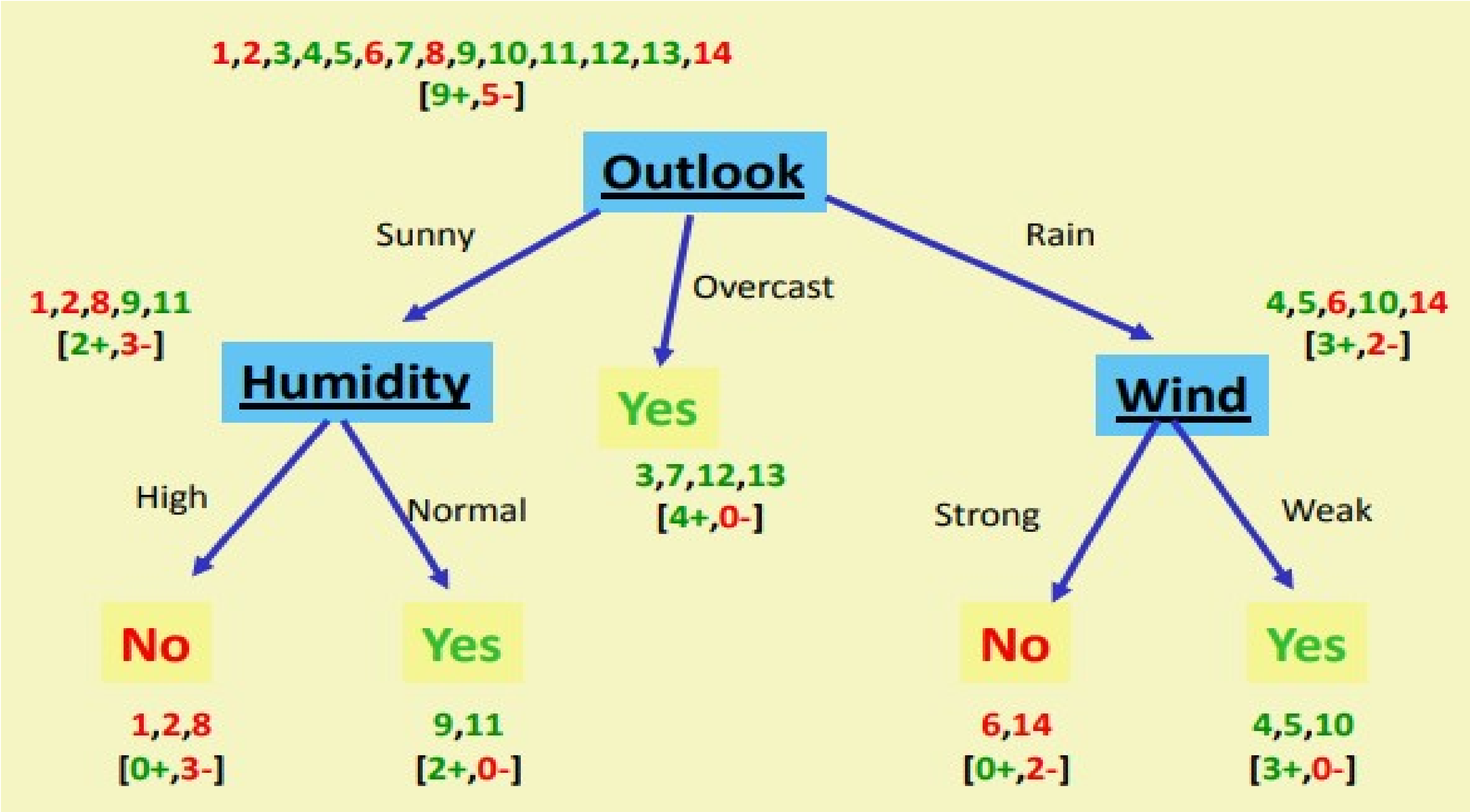


… continue to split…



ID3 performs a **greedy search** in which the algorithm **never backtracks** to reconsider earlier choices. This type of search is likely not to be a globally optimum solution, but generally works well

… final tree!



- If some attributes are not useful for classification, they will not be selected to grow the tree. For this reason, decision trees often used as a pre-processing for other learning algorithms which suffers from the presence of irrelevant information
- For a sufficiently complex (i.e. large) tree, all instances can be correctly classified. However, this can lead to **overfitting** (we will see this later)

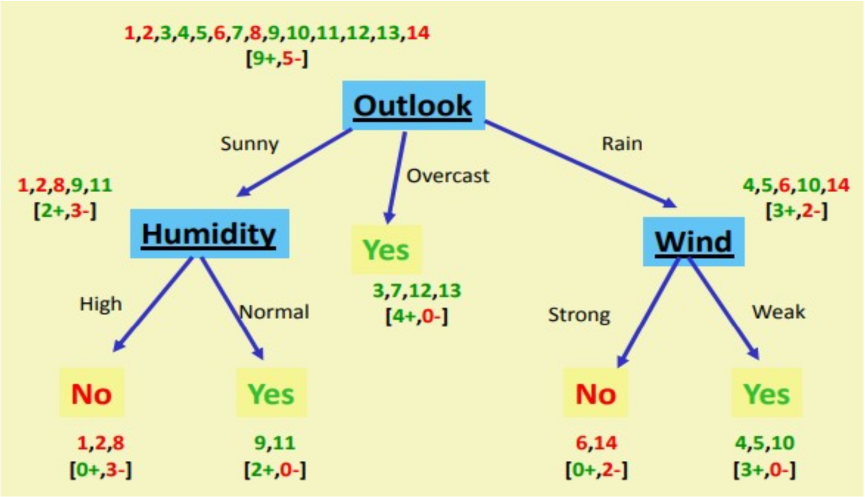
The C4.5 algorithm

The **information gain** is a measure that tends to prefer attributes with large number of possible values. To solve this, the successor of ID3, **C4.5** (Quinlan 93), uses the **gain ratio** as partitioning criterion. In addition, this new algorithm was improved to handle with missing data and continuous attributes (which are splitted into categories).

Gain ratio (GR): Takes into account the number and sizes of branches when choosing an attribute, penalizing those with many values and instances uniformly distributed.

$$GR = -\frac{IG}{Info} \qquad Info = -\sum_i p_i \vee \frac{\square}{N} \log_2 p_i \vee \frac{\square}{N}$$

$$Info_{Outlook} = -\frac{5}{14} log_2\left(\frac{5}{14}\right) - \frac{5}{14} log_2\left(\frac{5}{14}\right) - \frac{4}{14} log_2\left(\frac{4}{14}\right) = 1.577$$



$$GR_{Outlook} = -\frac{IG_{Outlook}}{Info_{Outlook}} = \frac{0.246}{1.577} = \boxed{0.157}$$

attribute chosen

$$GR_{Humidity} = 0.152$$

$$GR_{Wind} = 0.049$$

C5.0 is just a more efficient implementation of C4.5 (faster computing times).
Most of the algorithms that have been developed for learning classification trees are variations of ID3 and its successors C4.5 and C5.0.

Tree construction in R

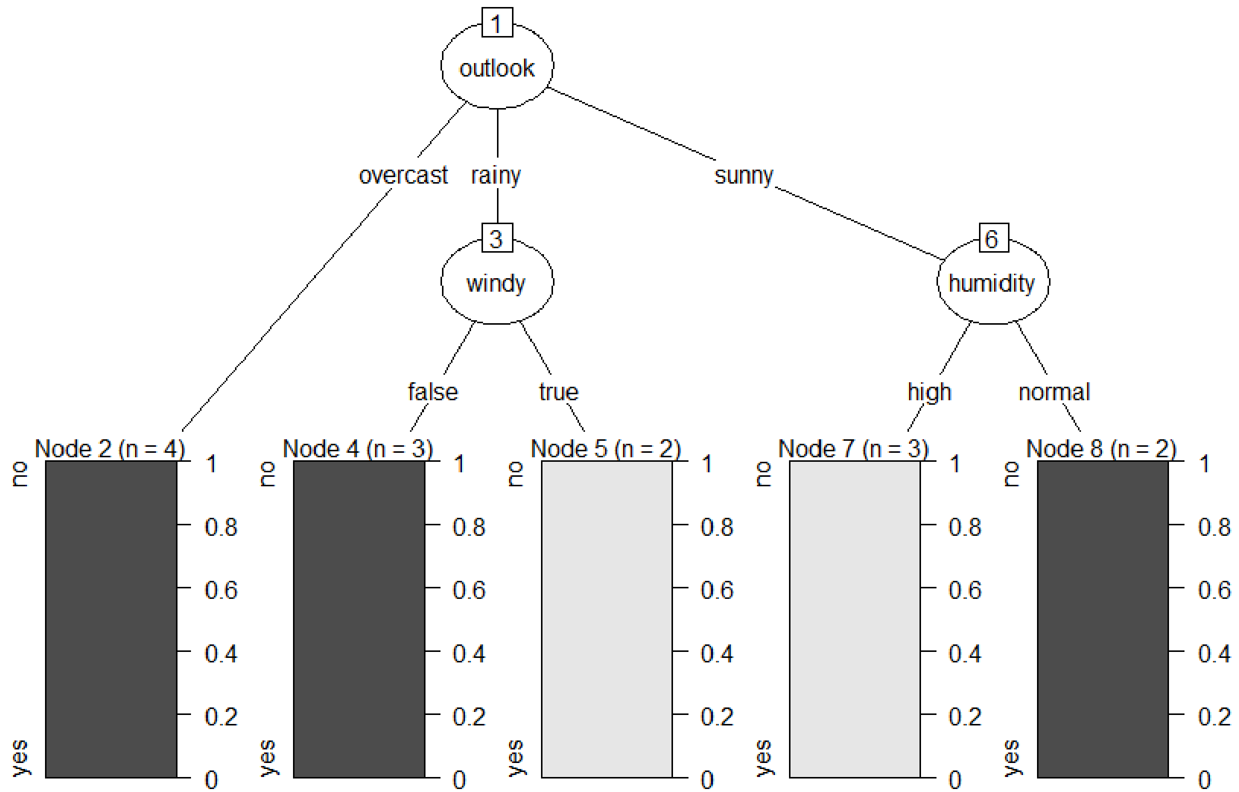
There are many packages in R to build classification tress: *tree*, *rpart*, *rpart2*, *C5.0*, etc.

outlook	*100.00*
humidity	*35.71*
windy	*35.71*
temp	*0.00*

Use of C5.0 (based on GR):

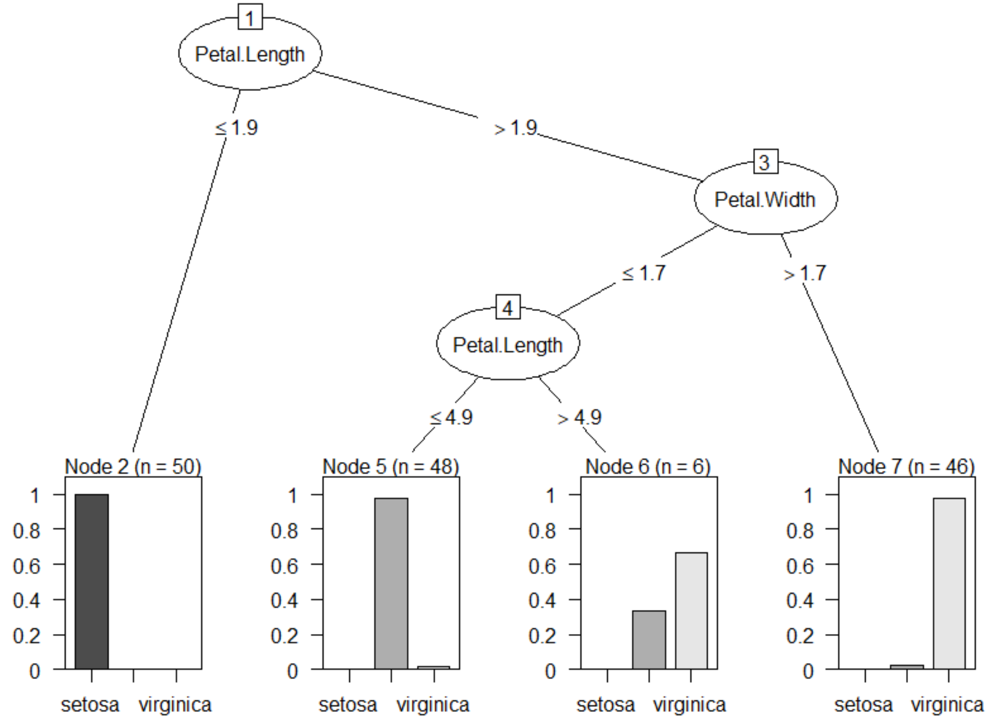
Example for playTennis (categorical attributes)

```
## read dataset
tennis = read.csv(“…/tennis.csv")
## grow the tree
library(C50)
t = C5.0(formula = play ~ .,
data = tennis)
## plot the tree
plot(t)
summary(t)

## percentage of training samples that
fall into all the terminal nodes after the split
C5imp(t)
        Overall
```

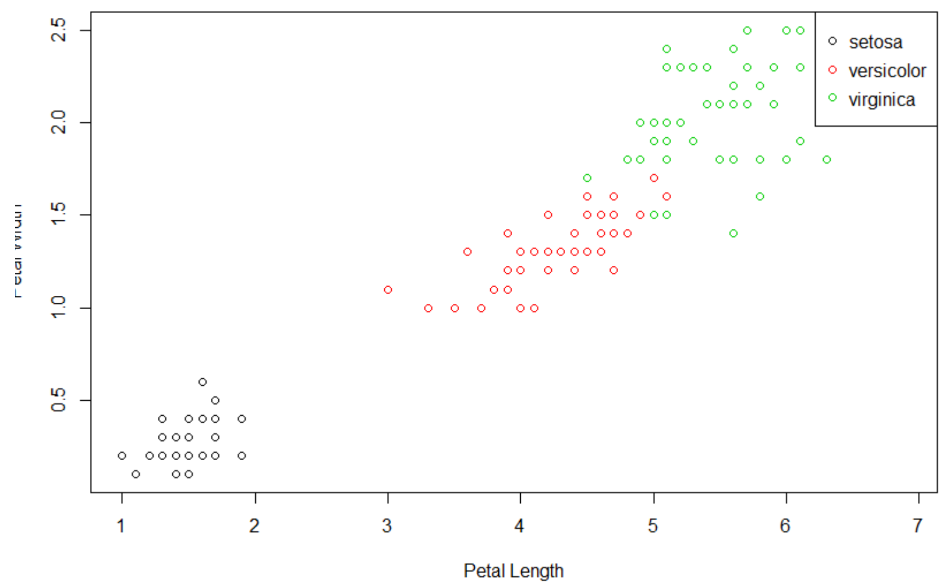


Example for iris (continuous attributes)

```
## continuous attributes are splitted into
categories based on thresholds
t = C5.0(formula = Species ~ ., data = iris)
plot(t)
summary(t)

## there are only two relevant predictors
t = C5.0(formula = Species ~ Petal.Length +
Petal.Width, data = iris)
plot(t)
summary(t)
```



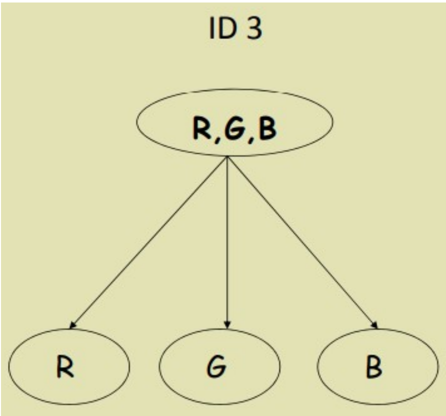

```
## the total space is partitioned according to the
thresholds determined by the tree
with(iris, plot(Petal.Length, Petal.Width, col =
Species, xlab = "Petal Length", ylab = "Petal Width"))
legend("topright", levels(iris$Species), col =
1:length(levels(iris$Species)), pch = 1)
```

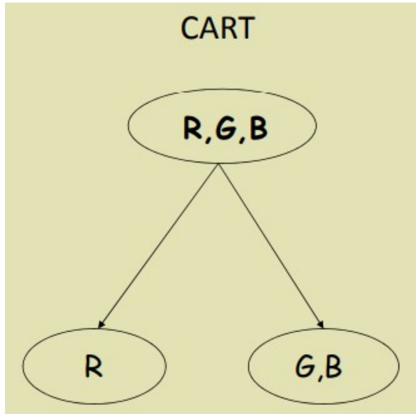


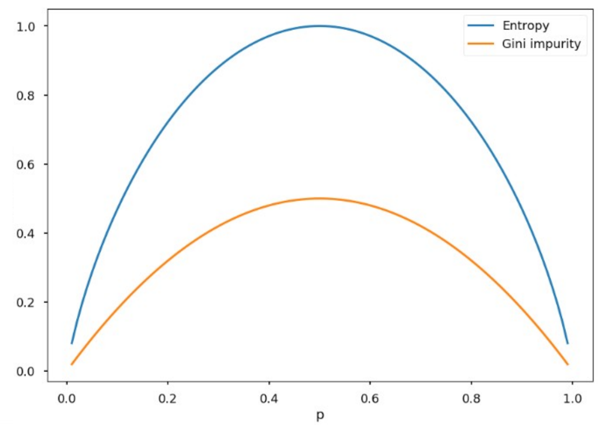
CART (Classification And Regression Trees)

Splitting of **binary** attributes, based on the **Gini index** (another measure of the purity of the node). Lower Gini values are preferred (perfect index = 0).

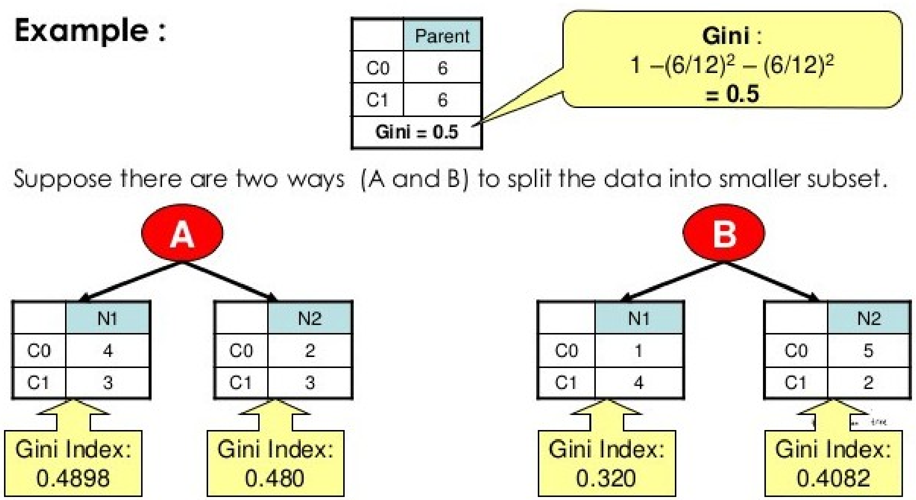
$$GINI = 1 - \sum_{i=1}^{n} {p_i}^2$$





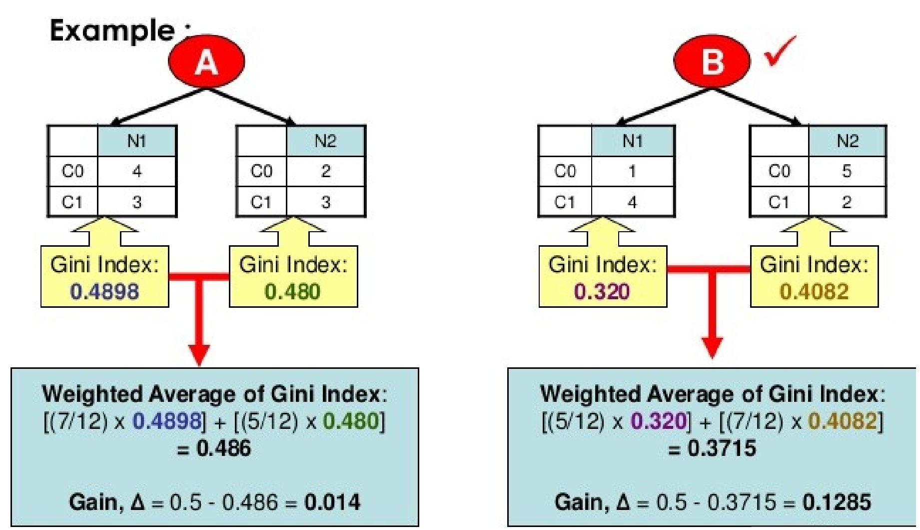


Splitting Binary Attributes (using Gini)



$$GINI_{N1} = 1 - \left(\frac{4}{7}\right)^2 - \left(\frac{3}{7}\right)^2 = 0.4898$$

Splitting Binary Attributes (using Gini)



Therefore, **B** is preferred

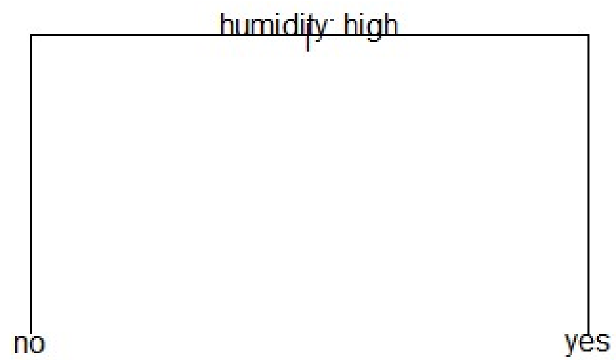
Tree construction in R

There are many packages in R to build classification tress: *tree*, *rpart*, *rpart2*, *C5.0*, etc.

Use of *tree* and *rpart* (CART; based on the Gini index):

Example for playTennis (categorical attributes)

```
## tree package
library(tree)
# default parameters
t = tree(formula = play ~ ., data = tennis)
plot(t)
text(t, pretty = F)
```



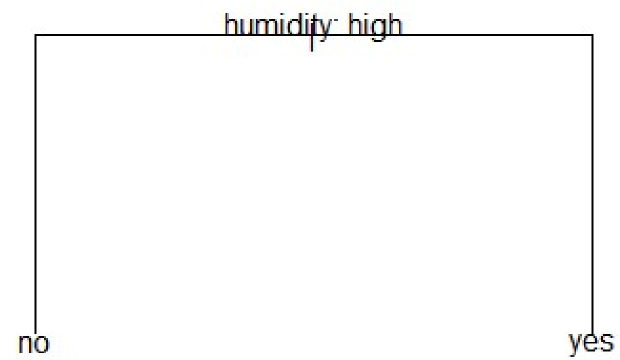
Tree construction in R

There are many packages in R to build classification tress: *tree*, *rpart*, *rpart2*, *C5.0*, etc.

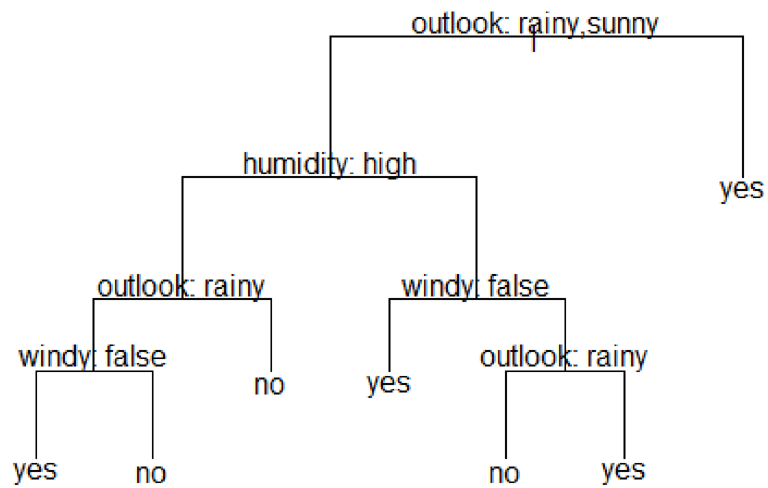
Use of *tree* and *rpart* (CART; based on the Gini index):

Example for playTennis (categorical attributes)

```
## tree package
library(tree)
# default parameters
t = tree(formula = play ~ ., data = tennis)
plot(t)
text(t, pretty = F)
```




```
# user-defined parameters
t = tree(formula = play ~ ., data = tennis,
minsize = 1)
plot(t)
text(t, pretty = F)
```



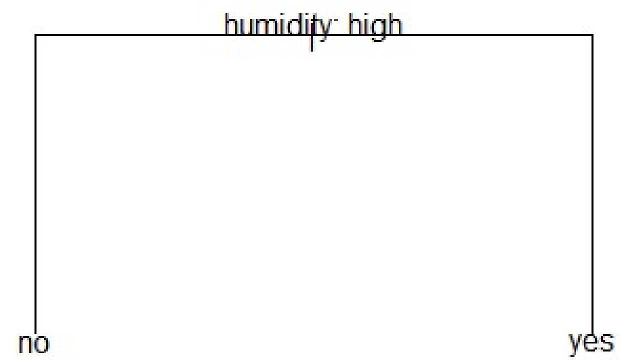
Tree construction in R

There are many packages in R to build classification tress: *tree*, *rpart*, *rpart2*, *C5.0*, etc.

Use of *tree* and *rpart* (CART; based on the Gini index):

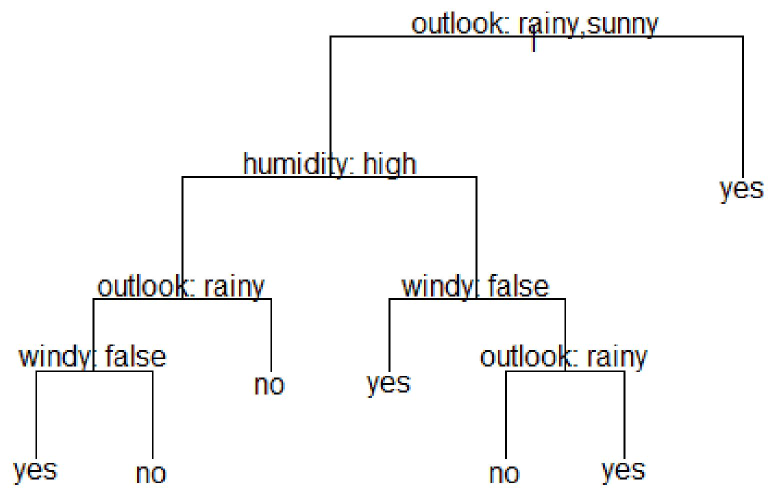
<u>*Example for playTennis (categorical attributes)*</u>

```
## tree package
library(tree)
# default parameters
t = tree(formula = play ~ ., data = tennis)
plot(t)
text(t, pretty = F)
```




```
# user-defined parameters
t = tree(formula = play ~ ., data = tennis,
minsize = 1)
plot(t)
text(t, pretty = F)
```

Documentation must be carefully read!!

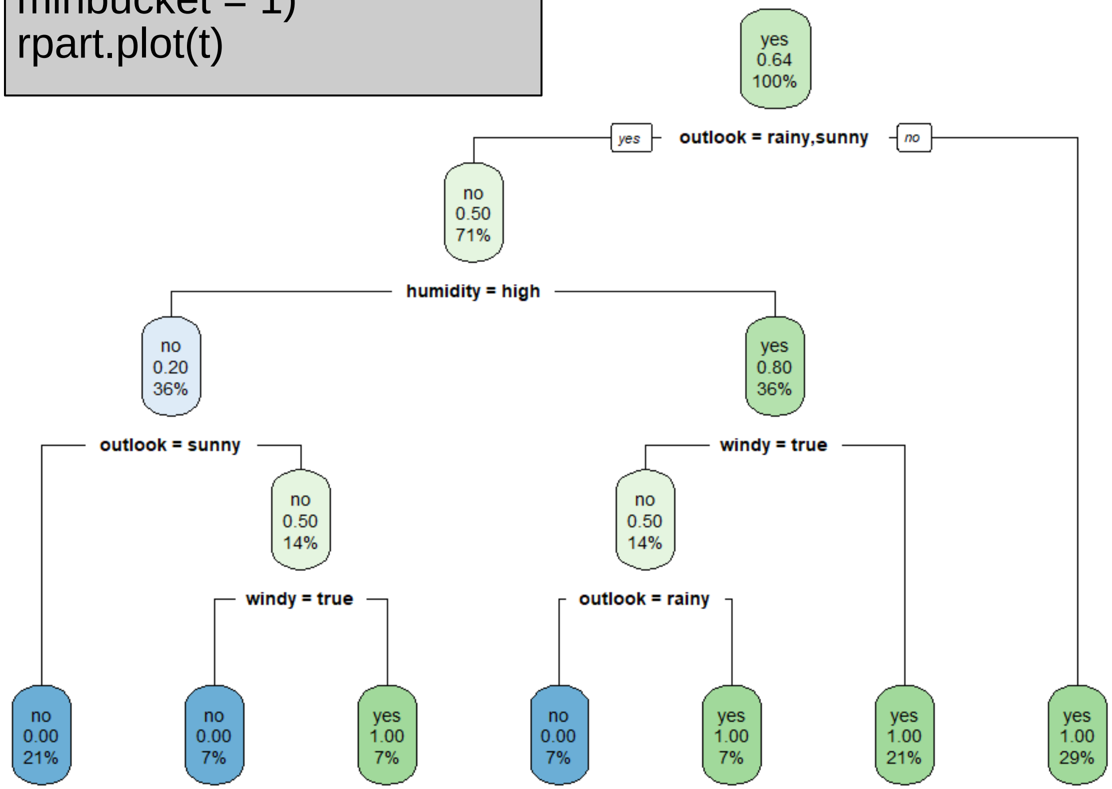


```
## rpart/rpart.plot packages
library(rpart)
# default parameters
t = rpart(formula = play ~ .,
data = tennis)
```

library(rpart.plot)

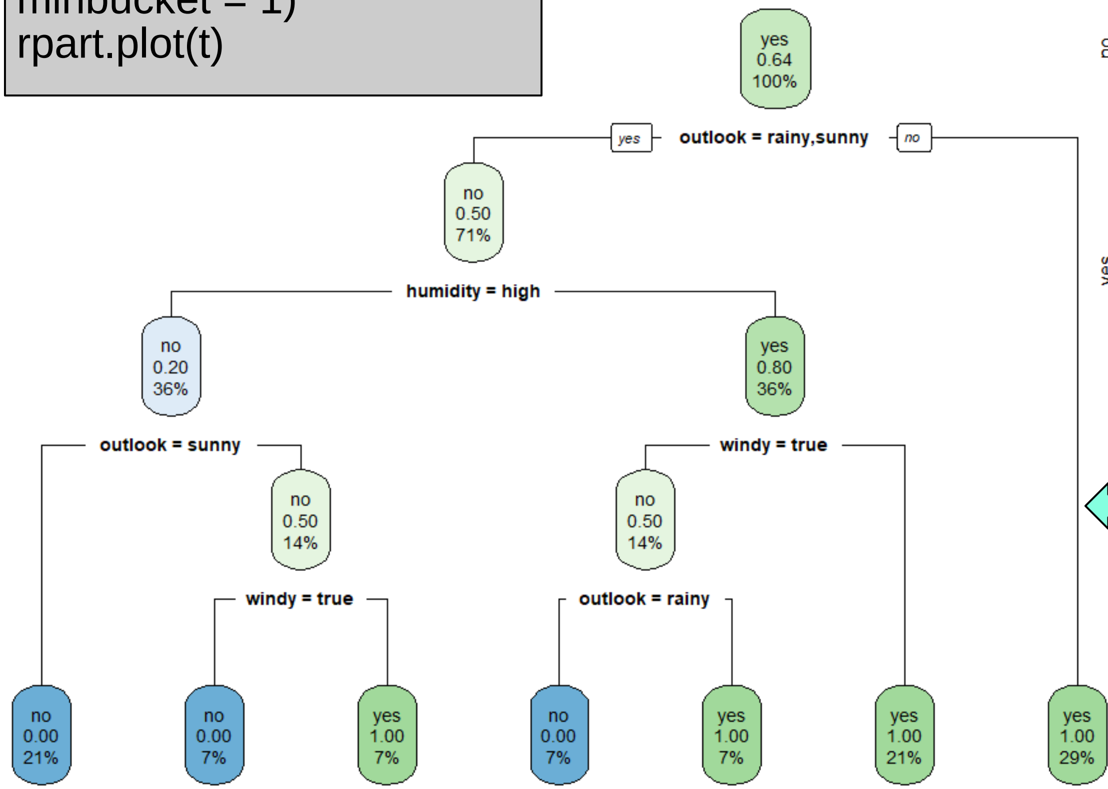
```r
## rpart/rpart.plot packages
library(rpart)
# default parameters
t = rpart(formula = play ~ .,
data = tennis)
library(rpart.plot)
rpart.plot(t)
```

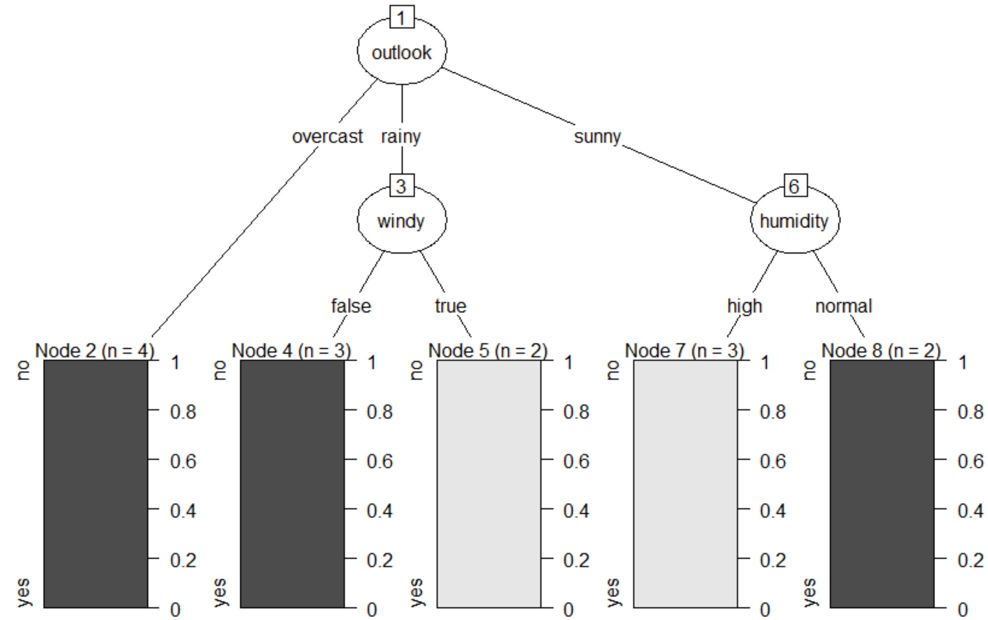

```r
# user-defined parameters
t = rpart(formula = play ~ .,
data = tennis, minsplit = 2,
minbucket = 1)
rpart.plot(t)
```



```
## rpart/rpart.plot packages
library(rpart)
# default parameters
t = rpart(formula = play ~ .,
data = tennis)
library(rpart.plot)
rpart.plot(t)
```


```
# user-defined parameters
t = rpart(formula = play ~ .,
data = tennis, minsplit = 2,
minbucket = 1)
rpart.plot(t)
```

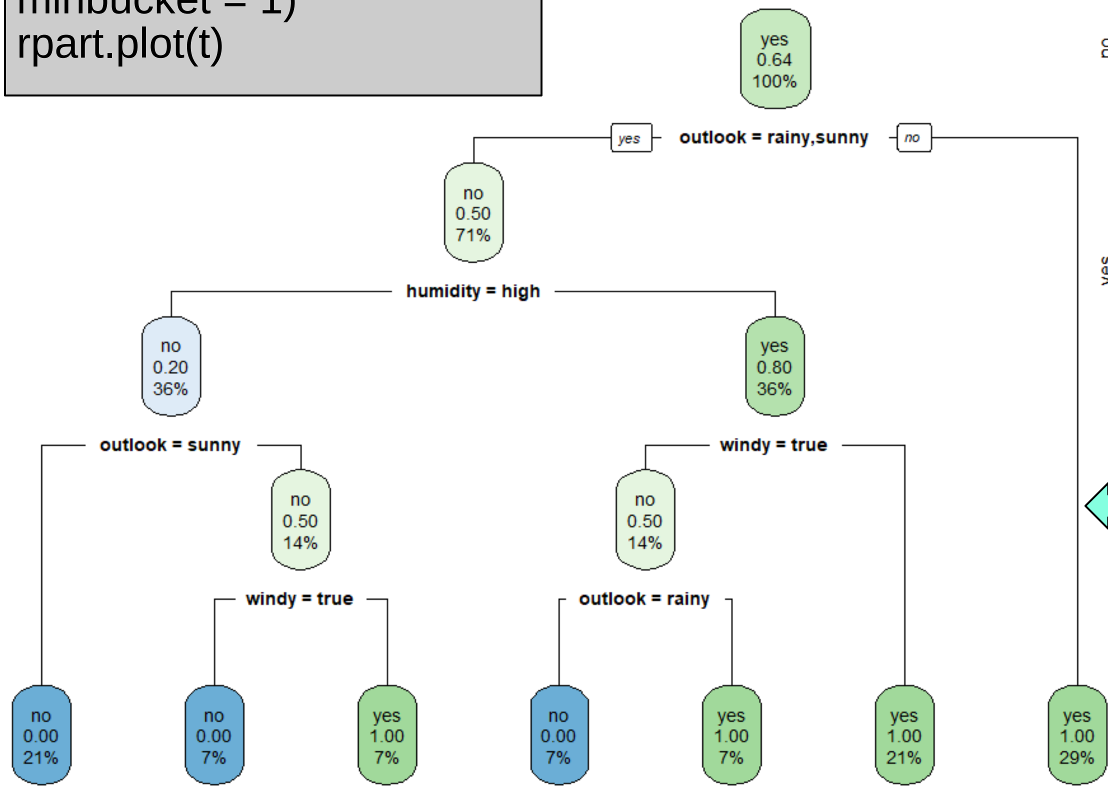


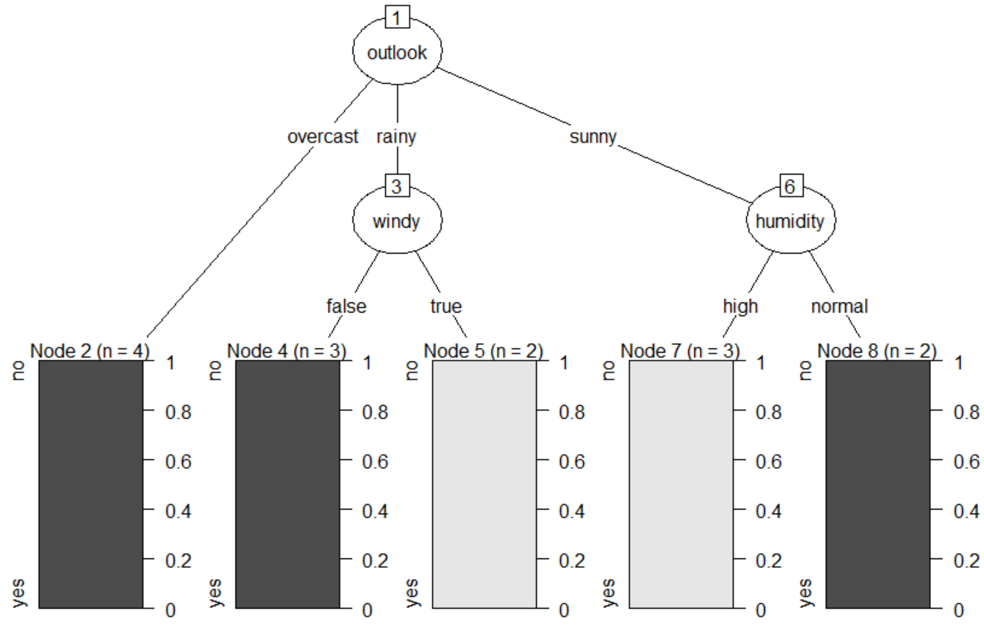


Compare with the result obtained with the *C50* package: **which are the differences?**

```r
## rpart/rpart.plot packages
library(rpart)
# default parameters
t = rpart(formula = play ~ .,
data = tennis)
library(rpart.plot)
rpart.plot(t)
```


```r
# user-defined parameters
t = rpart(formula = play ~ .,
data = tennis, minsplit = 2,
minbucket = 1)
rpart.plot(t)
```



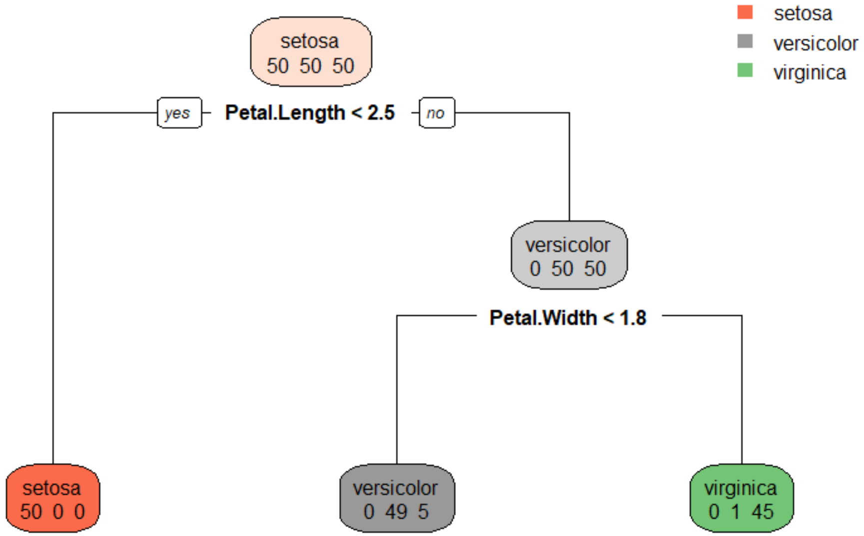


Compare with the result obtained with the *C50* package: **which are the differences?**

Documentation must be carefully read!!

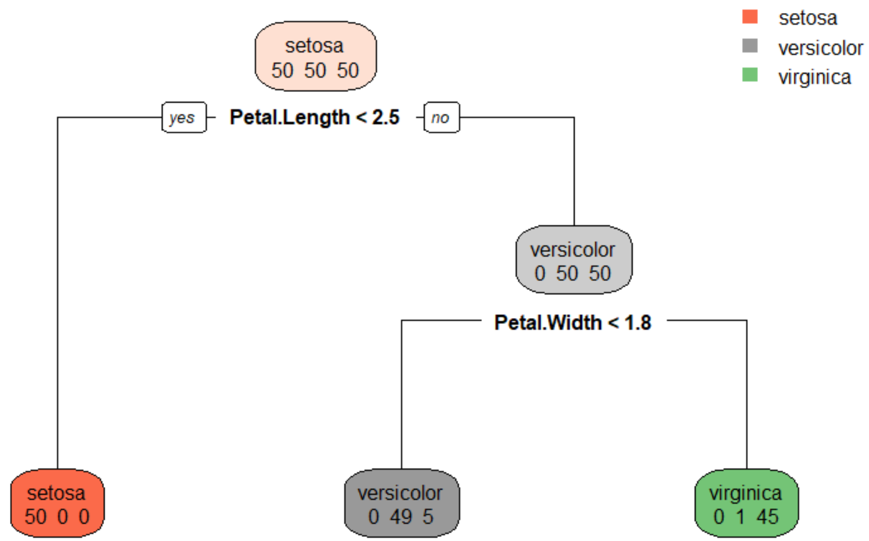
Example for iris (continuous attributes)

```
# default parameters
t = rpart(formula = Species ~ ., data = iris)
rpart.plot(t, extra = 1)
```

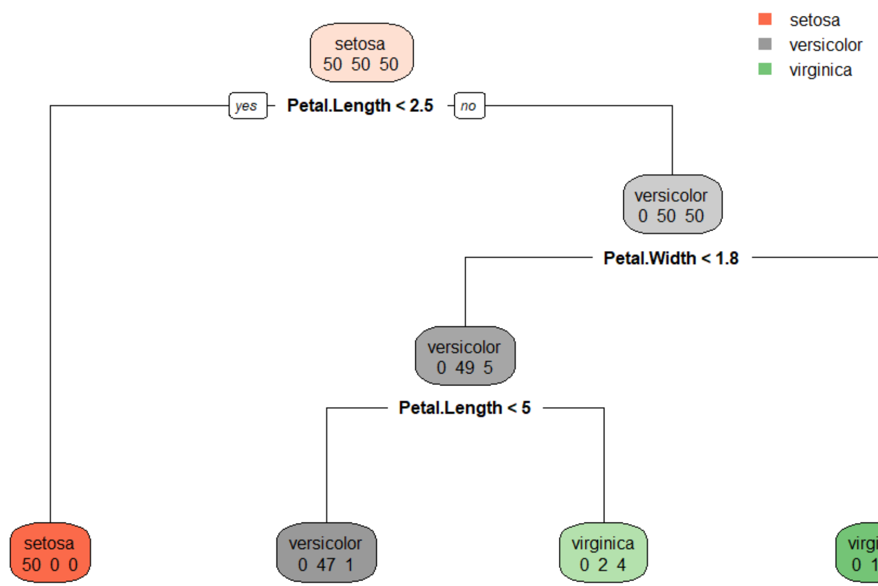


Example for iris (continuous attributes)

```
# default parameters
t = rpart(formula = Species ~ ., data = iris)
rpart.plot(t, extra = 1)
```

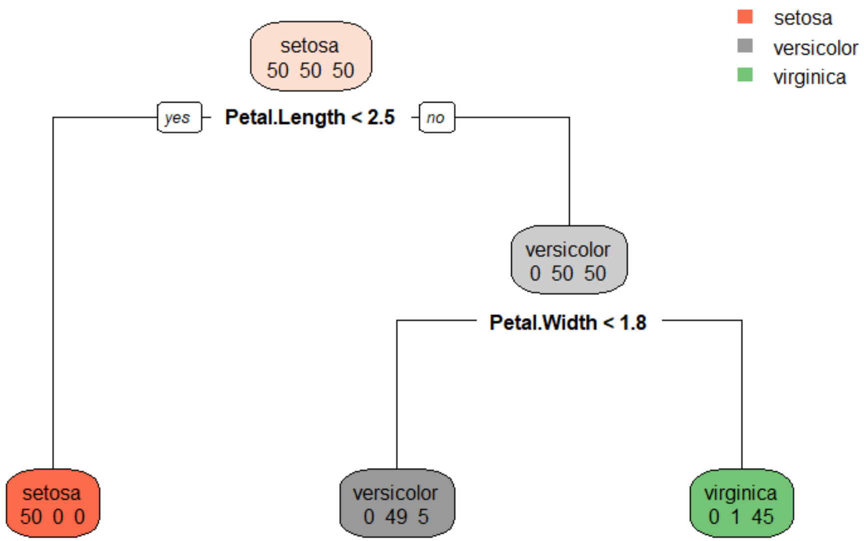



```
# user-defined parameters
t = rpart(formula = Species ~ ., data = iris,
minsplit = 2, minbucket = 1)
rpart.plot(t, extra = 1)
```

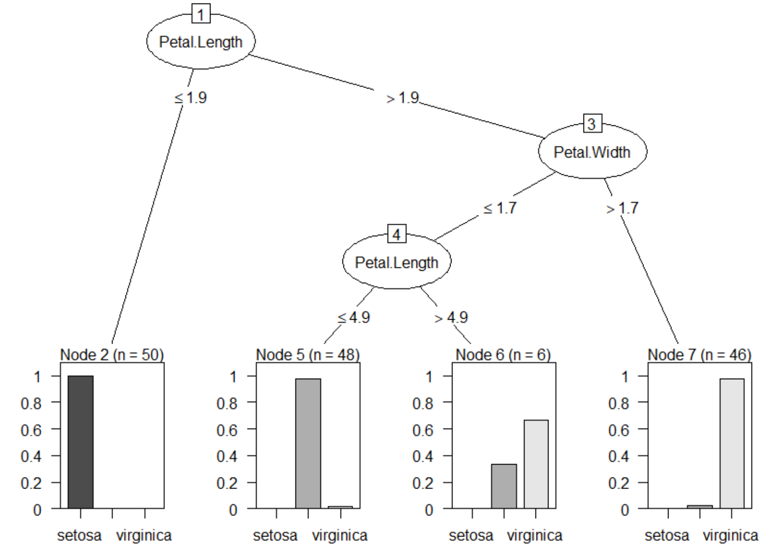


Example for iris (continuous attributes)

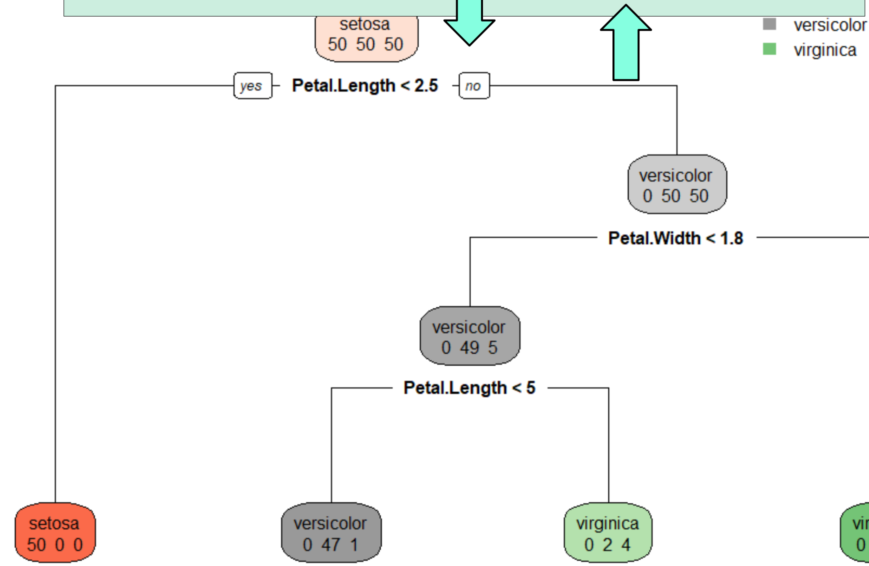
```
# default parameters
t = rpart(formula = Species ~ ., data = iris)
rpart.plot(t, extra = 1)
```




```
# user-defined parameters
t = rpart(formula = Species ~ ., data = iris,
minsplit = 2, minbucket = 1)
rpart.plot(t, extra = 1)
```

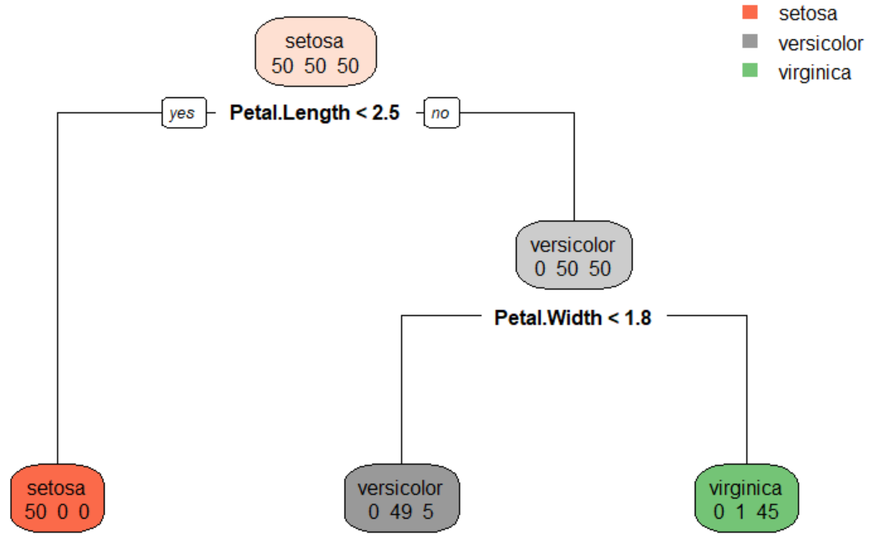


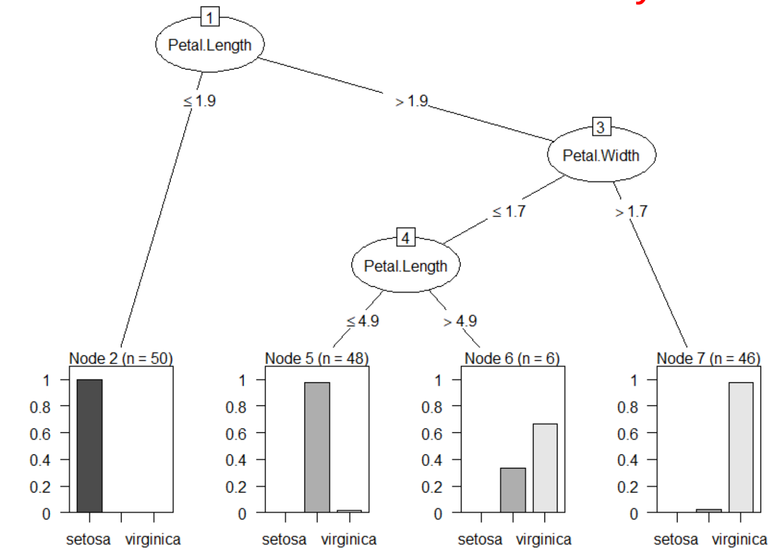
Compare with the result obtained with the *C50* package: **which are the differences?**



Documentation must be carefully read!!

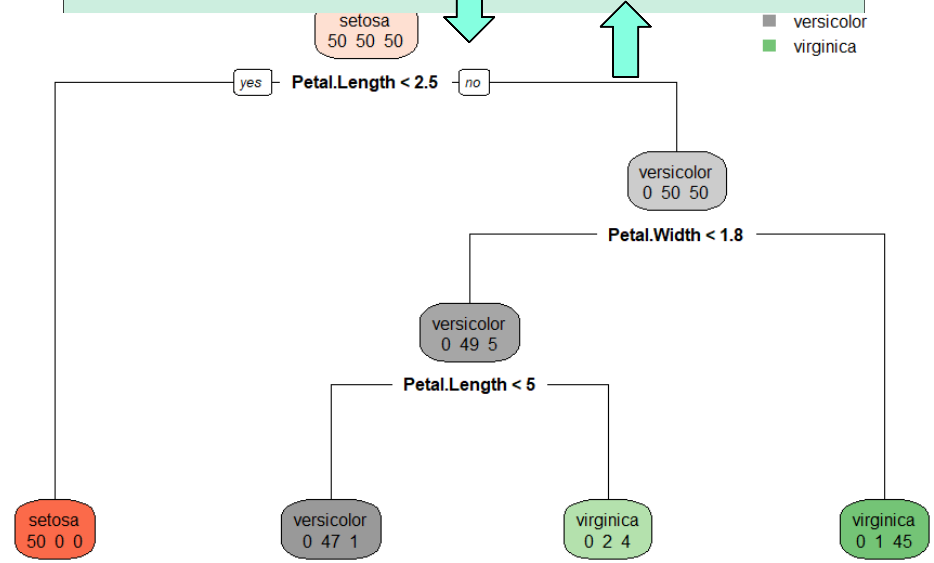
```
# default parameters
t = rpart(formula = Species ~ ., data = iris)
rpart.plot(t, extra = 1)
```





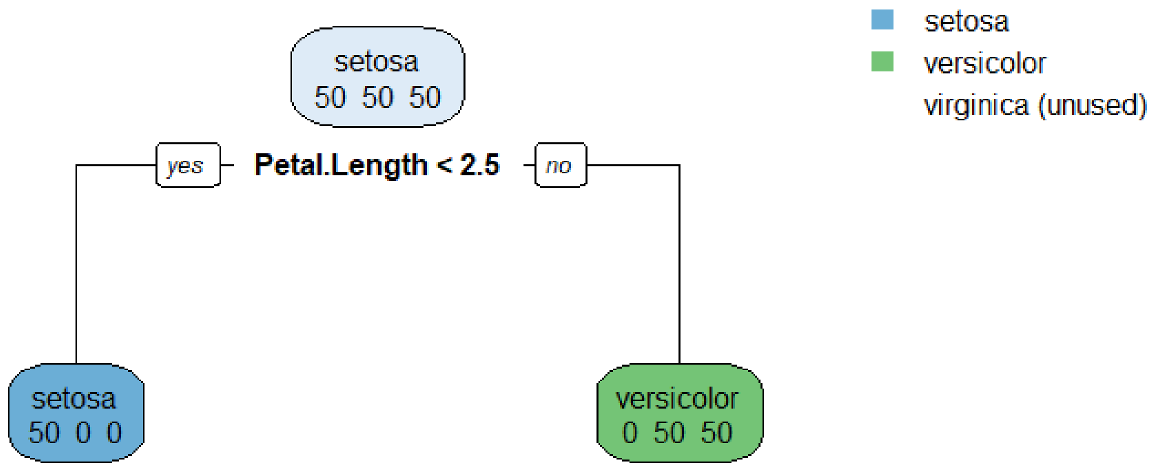

```
# user-defined parameters
t = rpart(formula = Species ~ ., data = iris,
minsplit = 2, minbucket = 1)
rpart.plot(t, extra = 1)
```

Compare with the result obtained with the *C50* package: **which are the differences?**

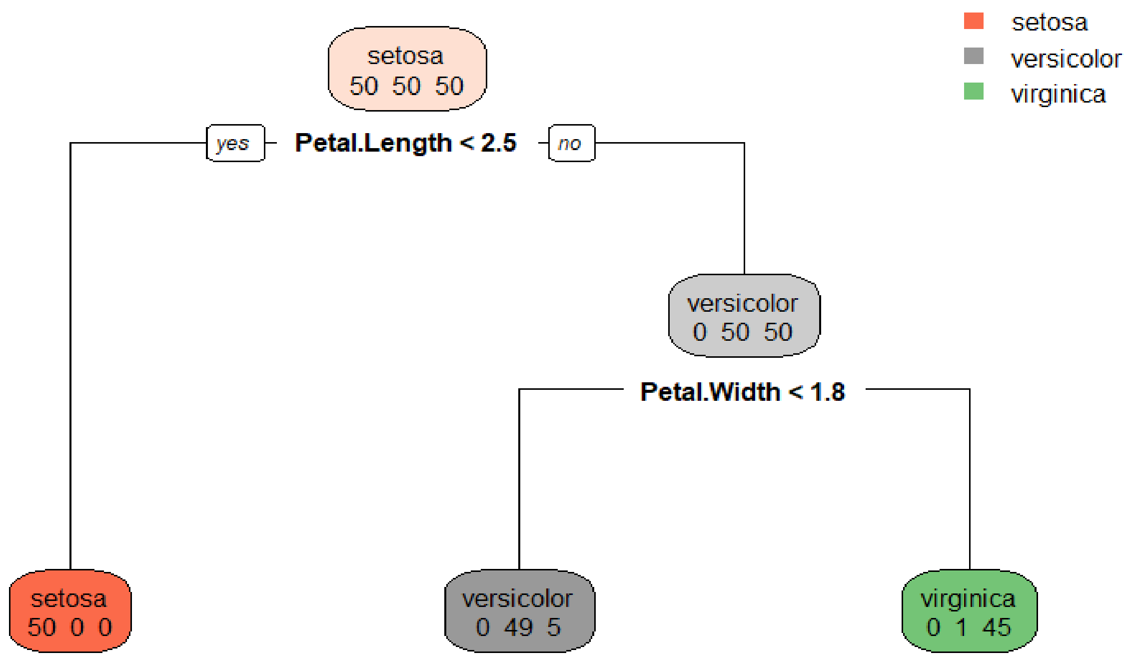


```
summary(t)
Call:
rpart(formula = Species ~ ., data = iris, minsplit = 2, minbucket = 1)
  n= 150

    CP nsplit rel error xerror       xstd
1 0.50      0      1.00   1.22 0.04772141
2 0.44      1      0.50   0.70 0.06110101
3 0.02      2      0.06   0.11 0.03192700
4 0.01      3      0.04   0.10 0.03055050

Variable importance
 Petal.Width Petal.Length Sepal.Length  Sepal.Width
          34           32           20           14

Node number 1: 150 observations,    complexity param=0.5
  predicted class=setosa      expected loss=0.6666667  P(node) =1
    class counts:    50    50    50
   probabilities: 0.333 0.333 0.333
  left son=2 (50 obs) right son=3 (100 obs)
  Primary splits:
      Petal.Length < 2.45 to the left,  improve=50.00000, (0 missing)
      Petal.Width  < 0.8  to the left,  improve=50.00000, (0 missing)
      Sepal.Length < 5.45 to the left,  improve=34.16405, (0 missing)
      Sepal.Width  < 3.35 to the right, improve=19.03851, (0 missing)
  Surrogate splits:
      Petal.Width  < 0.8  to the left,  agree=1.000, adj=1.00, (0 split)
      Sepal.Length < 5.45 to the left,  agree=0.920, adj=0.76, (0 split)
      Sepal.Width  < 3.35 to the right, agree=0.833, adj=0.50, (0 split)

Node number 2: 50 observations
  predicted class=setosa      expected loss=0  P(node) =0.3333333
    class counts:    50     0     0
   probabilities: 1.000 0.000 0.000

Node number 3: 100 observations,    complexity param=0.44
  predicted class=versicolor  expected loss=0.5  P(node) =0.6666667
    class counts:     0    50    50
   probabilities: 0.000 0.500 0.500
  left son=6 (54 obs) right son=7 (46 obs)
  Primary splits:
      Petal.Width  < 1.75 to the left,  improve=38.969400, (0 missing)
      Petal.Length < 4.75 to the left,  improve=37.353540, (0 missing)
      Sepal.Length < 6.15 to the left,  improve=10.686870, (0 missing)
      Sepal.Width  < 2.45 to the left,  improve= 3.555556, (0 missing)
  Surrogate splits:
      Petal.Length < 4.75 to the left,  agree=0.91, adj=0.804, (0 split)
      Sepal.Length < 6.15 to the left,  agree=0.73, adj=0.413, (0 split)
      Sepal.Width  < 2.95 to the left,  agree=0.67, adj=0.283, (0 split)

Node number 6: 54 observations,    complexity param=0.02
  predicted class=versicolor  expected loss=0.09259259  P(node) =0.36
    class counts:     0    49     5
   probabilities: 0.000 0.907 0.093
  left son=12 (48 obs) right son=13 (6 obs)
  Primary splits:
      Petal.Length < 4.95 to the left,  improve=4.4490740, (0 missing)
      Sepal.Length < 7.1  to the left,  improve=1.6778480, (0 missing)
      Petal.Width  < 1.35 to the left,  improve=0.9971510, (0 missing)
      Sepal.Width  < 2.65 to the right, improve=0.2500139, (0 missing)

Node number 7: 46 observations
  predicted class=virginica   expected loss=0.02173913  P(node) =0.3066667
    class counts:     0     1    45
   probabilities: 0.000 0.022 0.978

Node number 12: 48 observations
  predicted class=versicolor  expected loss=0.02083333  P(node) =0.32
    class counts:     0    47     1
   probabilities: 0.000 0.979 0.021

Node number 13: 6 observations
  predicted class=virginica   expected loss=0.3333333  P(node) =0.04
    class counts:     0     2     4
   probabilities: 0.000 0.333 0.667
```

```
# user-defined parameters: "maxpdepth"
t = rpart(formula = Species ~ ., data = iris,
maxdepth = 1)
rpart.plot(t, extra = 1)
```

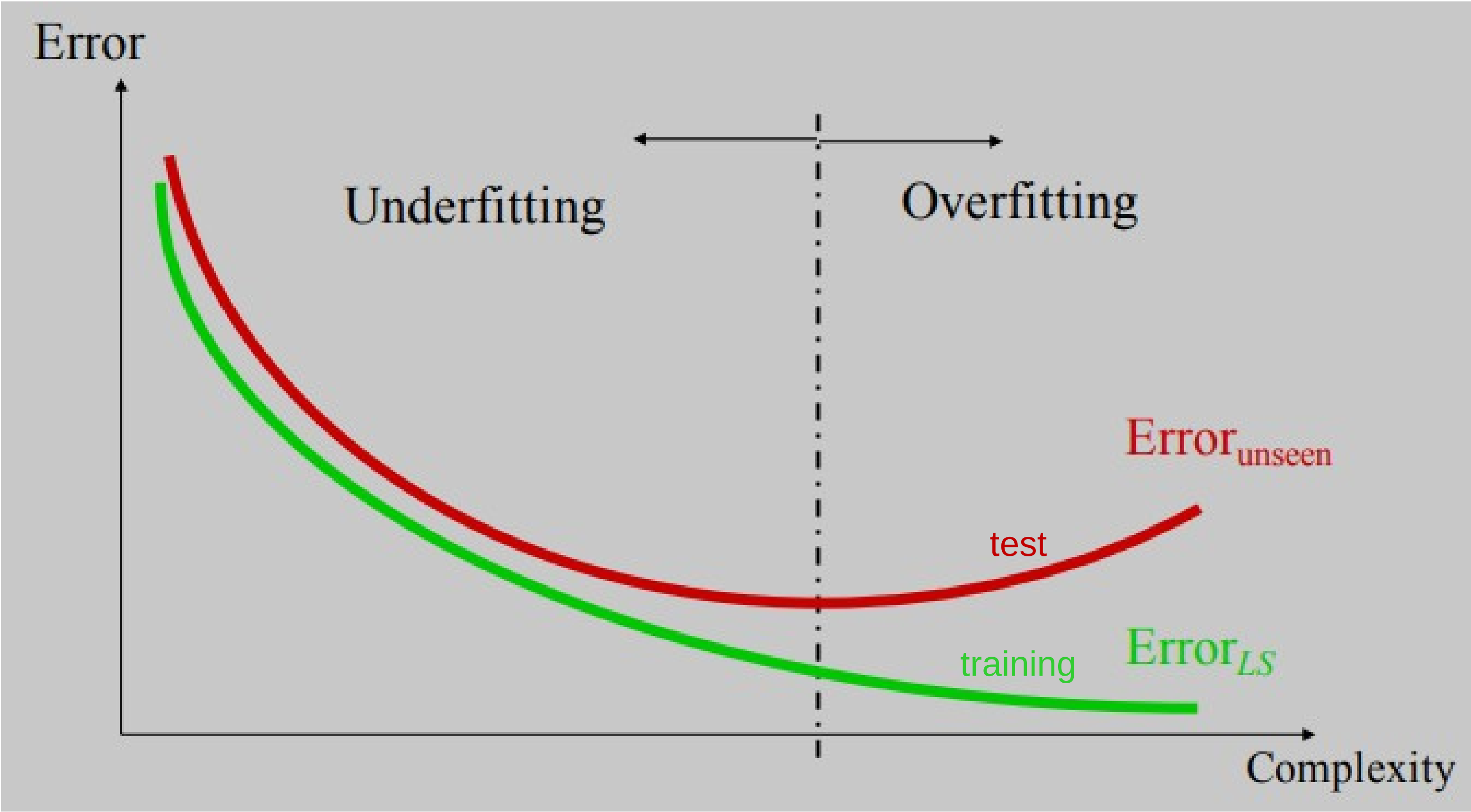



```
# user-defined parameters: "maxpdepth"
t = rpart(formula = Species ~ ., data = iris,
maxdepth = 2)
rpart.plot(t, extra = 1)
```



Overfitting

A large tree (i.e. with many terminal nodes) may tend to overfit the training data, leading to poor performance in the test set. Generally, we can improve this behavior by **pruning** the tree, i.e., cutting off some of the terminal nodes.



How can we avoid overfitting?

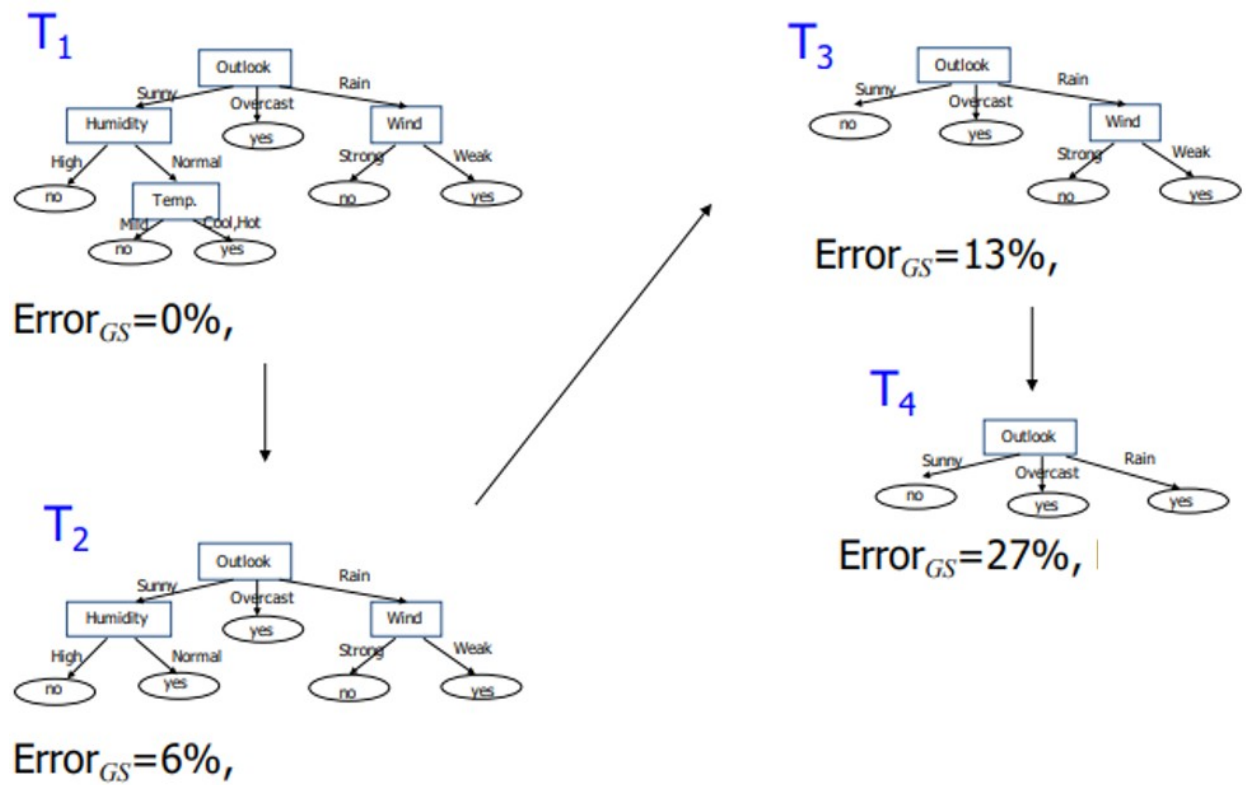
Pre-pruning: stop growing the tree before it reaches the point at which it perfectly classifies the learning sample. *Procedure:*
Stop splitting a node if:

- The number of objects is too small
- The impurity is low enough

This approach leads to small trees but can remove relevant splits

Post-pruning: allow the tree to overfit and then, one finalized, remove the less useful nodes. In general, this is preferred option. *Procedure:*
Compute a sequence of trees {T1, T2, …} where T1 is the complete tree. T2 is obtained by removing from T1 the node that less increases the error. Sometimes, this process is guided based on some **cost-complexity** criterion (e.g. in medicine)

The question is: **where to stop?** In practice, it is usual to split the learning dataset into two subsets: a **training sample** for growing the tree and a **test sample** for evaluating its generalization error (e.g. hold-out **cross-validation**).

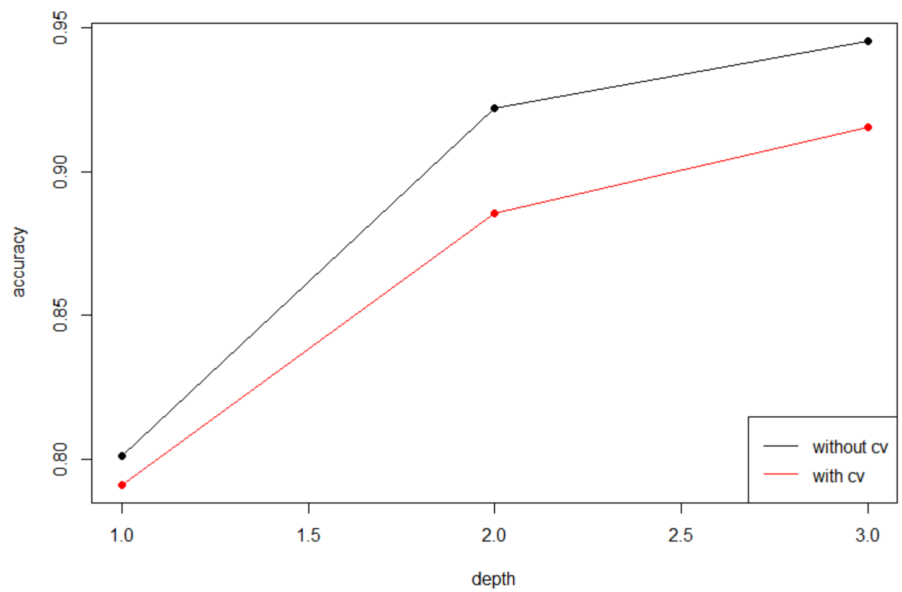


Post-pruning in R

Example for cars (dataset included in the caret package)

```
## exploring the dataset
library(caret)
data("cars")
summary(cars)
## convert continuous variable "Price" to categorical
cars$Price = as.factor(ifelse(cars$Price >= 22000, "E", "C"))

## 50% of dataset for training and the other 50% for test
n = dim(cars)[1]
set.seed(2)
indtrain = sample(1:n, round(0.5*n))
indtest = setdiff(1:n, indtrain)
dataset.train = cars[indtrain, ]
dataset.test = cars[indtest, ]

## performance without cross-validation, for increasingly complex configurations of the tree
acc.nocv = c()
for (md in 1:3) {  # maximum depth allowed for the tree
# learning the tree
t = rpart(formula = Price ~ ., data = dataset.train, maxdepth = md)
# applying learnt tree to predict
pred = predict(t, dataset.train, type = "class")
# performance evaluation
acc.nocv[md] = sum(diag(table(pred, dataset.train$Price))) / length(indtrain)
}
```

```
## performance with cross-validation, for increasingly complex configurations of the tree
acc.cv = c()
for (md in 1:3) {  # maximum depth allowed for the tree
# learning the tree
t = rpart(formula = Price ~ ., data = dataset.train, maxdepth = md)
# applying learnt tree to predict
pred = predict(t, dataset.test, type = "class")
# performance evaluation
acc.cv[md] = sum(diag(table(pred, dataset.test$Price))) / length(indtest)
}

## plotting results
matplot(cbind(acc.nocv, acc.cv), type = "o", pch = 19, lty = 1, col = c("black", "red"),
        xlab = "depth", ylab = "accuracy")
legend("bottomright", c("without cv", "with cv"), lty = 1, col = c("black", "red"))
grid()
```



<u>*Example for cars (using caret)*</u>

Caret tremendously simplifies the model fitting process, allowing for automatized cross-validation

```
## 50% of the dataset for cross-validation and
## 50% for test
indtrain = createDataPartition(y = cars$Price, p = 0.5,
list = FALSE)
dataset.train = cars[indtrain,]
dataset.test = cars[-indtrain,]
# 10 folds
trctrl = trainControl(method = "cv", number = 10)
## caret automatically tries different values of the
## method parameter (5 in this case, internally selected)
t = train(Price ~ ., data = dataset.train,
            method = "rpart2",
            trControl = trctrl,
            tuneLength = 5)
plot(t)

## prediction
pred = predict(t, newdata = dataset.test)
## evaluation
sum(diag(table(pred, dataset.test$Price))) / dim(dataset.test)[1]
```

To check the parameters handled by caret:
http://topepo.github.io/caret/index.html