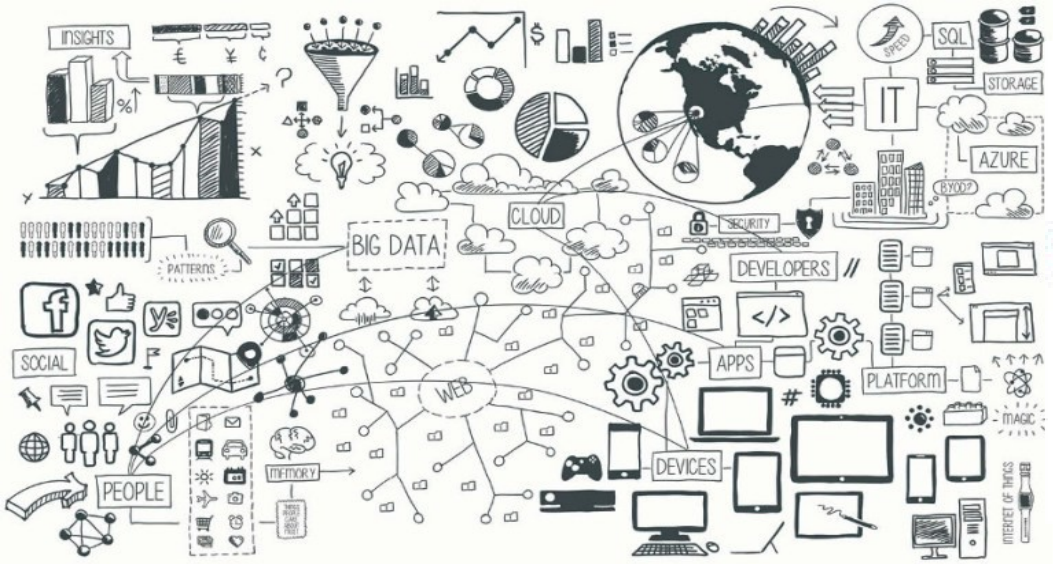
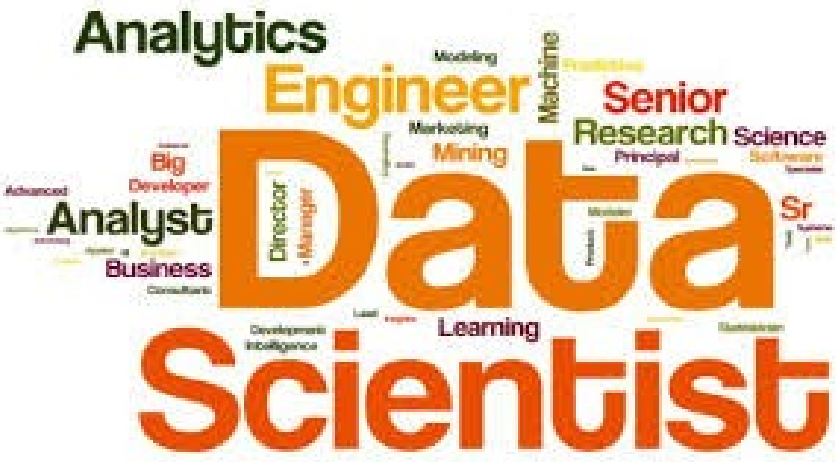
Data Mining (Minería de Datos)

Decision Trees: Regression

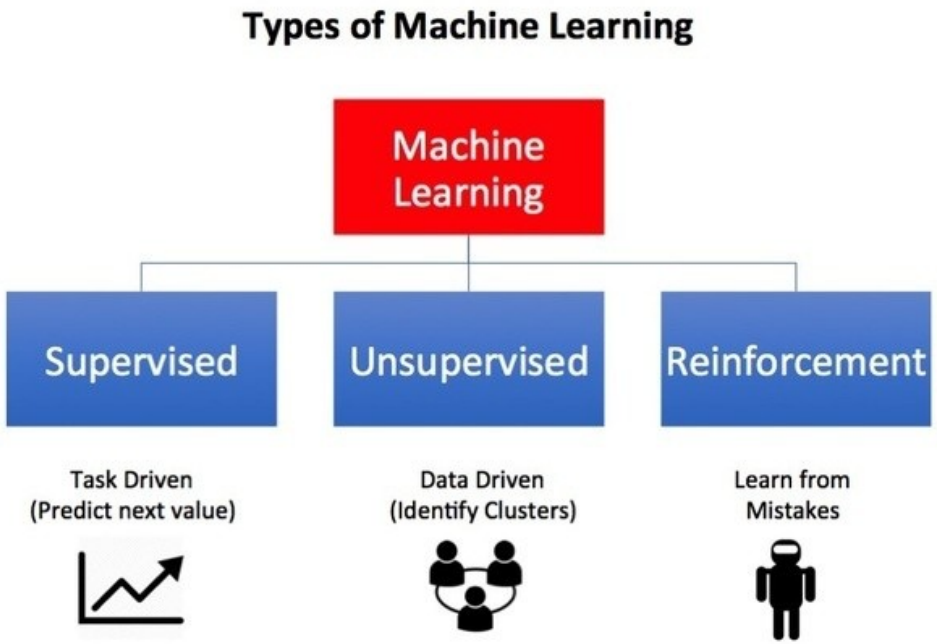




Sixto Herrera

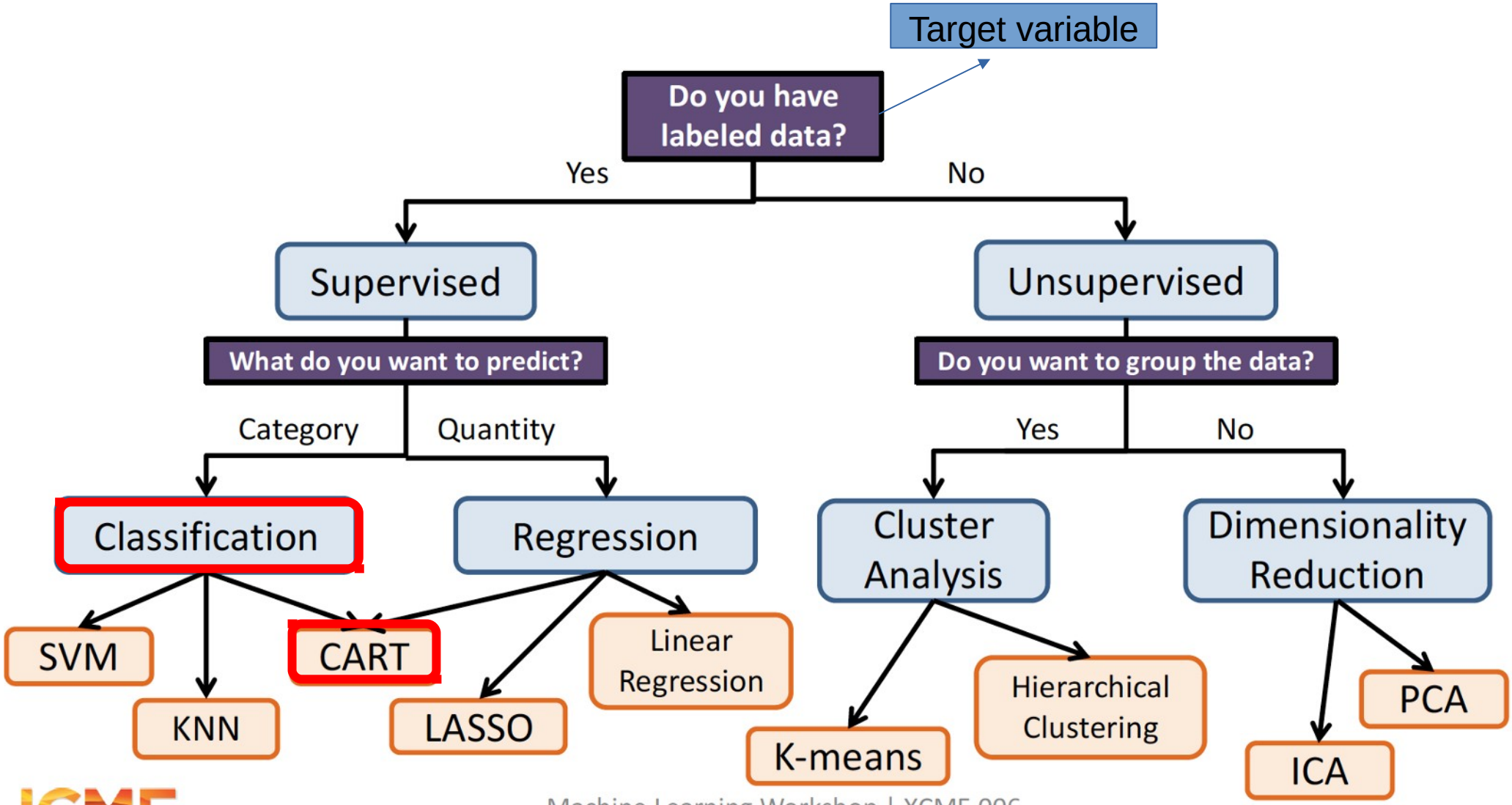
Grupo de Meteorología

Univ. de Cantabria – CSIC
MACC / IFCA



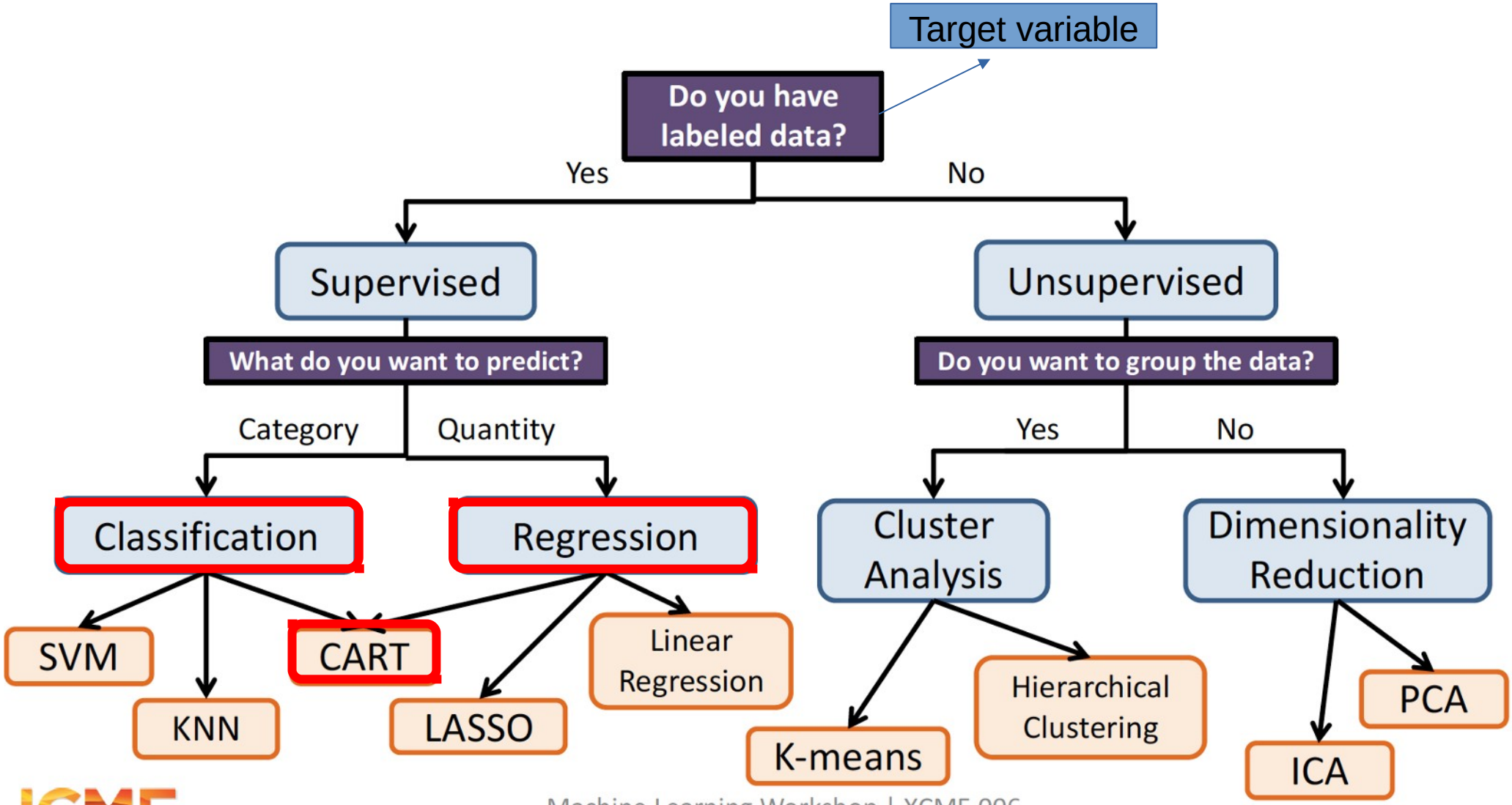
NOTA: Las líneas de código de R en esta presentación se muestran sobre un fondo gris

Oct	29	Aplazada (sesión de refuezo)
	31	Presentación, introducción y perspectiva histórica
Nov	5	Paradigmas, problemas canonicos y data challenges
	7	Reglas de asociación
	12	Práctica: Reglas de asociación
	14	Evaluación, sobrejuste y crossvalidacion
	19	Práctica: Cross-validación
	21	Árboles de clasificacion y decision
	26	Practica: Árboles de clasificación
		T01. Datos discretos
	28	Técnicas de vecinos cercano (k-NN)
Dic	3	Práctica: Vecinos cercanos
	5	Reducción de dimensión no lineal
	10	Práctica: Reducción de dimensión no lineal
		T02. Clasificación
	12	Árboles de clasificación y regresion (CART)
	17	Práctica: Árboles de clasificación y regresion (CART)
	19	Ensembles: Bagging and Boosting
Ene	7	Práctica Random Forests
	9	Práctica Gradient boosting
		T03. Prediccion
	14	Técnicas de agrupamiento
	16	Practica: Técnicas de agrupamiento
	21a	Practica: El paquete CARET
	21b	Examen



ICME

Machine Learning Workshop | XCME 006


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
ICA
PCA

Classification Trees

Aim:

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

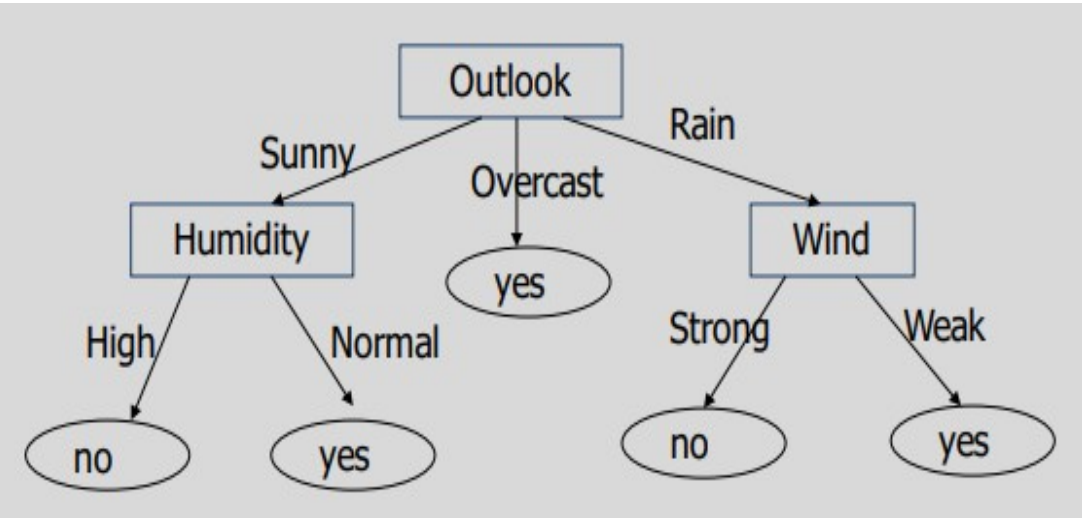
Structure:

- 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

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



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

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

Regression Trees

Aim:
To predict a **continuous** target variable 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

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**.

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

Many variants for attribute selection:

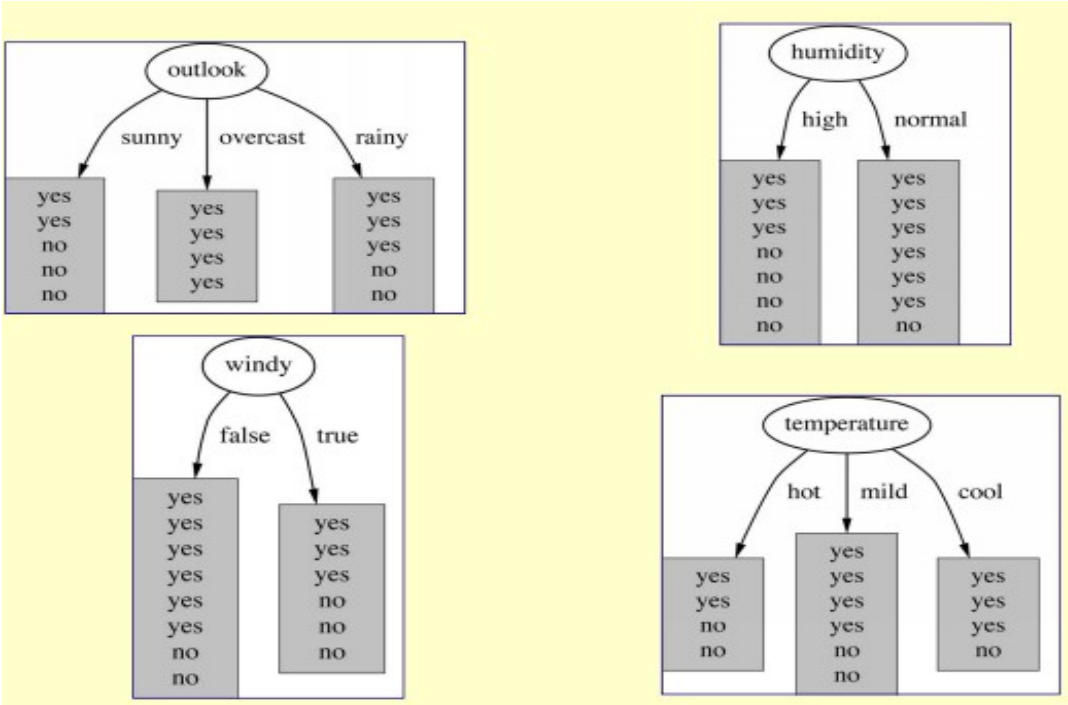
- from machine learning: **ID3** (**I**terative **D**ichotomizer), **C4.5** and **C5.0** (Quinlan 86, 93)
- from statistics: **CART** (**C**lassification **A**nd **R**egression **T**rees) (Breiman et al. 84)
- from pattern recognition: **CHAID** (**CH**i-squared **A**utomated **I**nteraction **D**etection)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

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

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**.

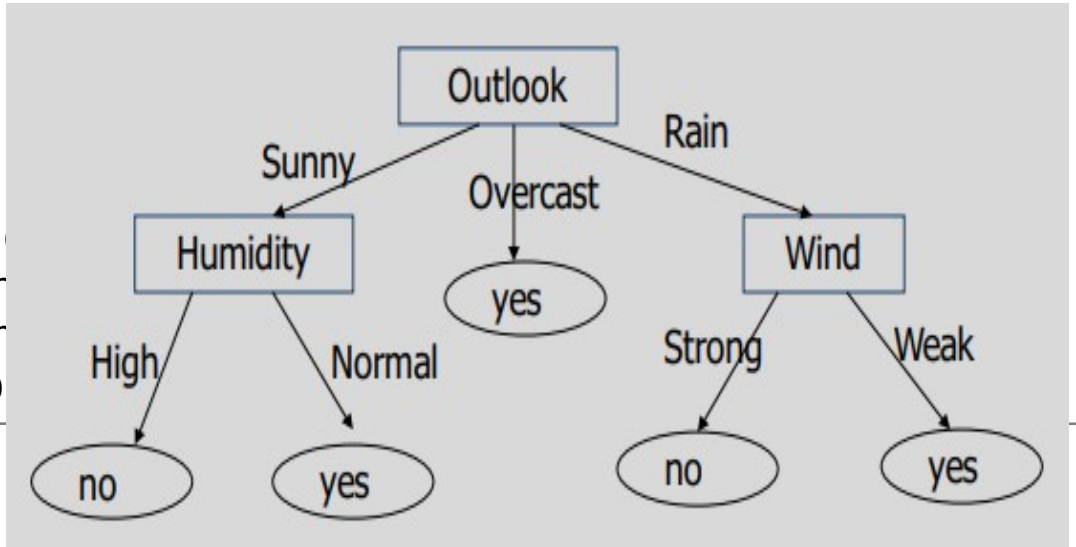
Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes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



Many variants for attribute selection:

- from machine learning: **ID3** (**I**terative **D**ichotomizer),
- from statistics: **CART** (**C**lassification **A**nd **R**egressior
- from pattern recognition: **CHAID** (**CH**i-squared **A**utor

Their main difference is the criterion followed to



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**.

ID3 → Information Gain (IG) & Entropy (H)

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

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

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)

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**.

ID3 → Information Gain (IG) & Entropy (H)

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

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

C4.5 → Information Gain (IG) & Gain Ratio (GR)

$$GR = -\frac{IG}{Info}$$

$$Info = -\sum_{i} \frac{|p_i|}{N}\, log_2 \frac{|p_i|}{N}$$

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)

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**.

ID3 → Information Gain (IG) & Entropy (H)

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

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

C4.5 → Information Gain (IG) & Gain Ratio (GR)

$$GR = -\frac{IG}{Info}$$

$$Info = -\sum_{i} \frac{|p_i|}{N}\, log_2 \frac{|p_i|}{N}$$

What is the measure considered for Regression Trees?

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)

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**.

ID3 → Information Gain (IG) & Entropy (H)

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

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

C4.5 → Information Gain (IG) & Gain Ratio (GR)

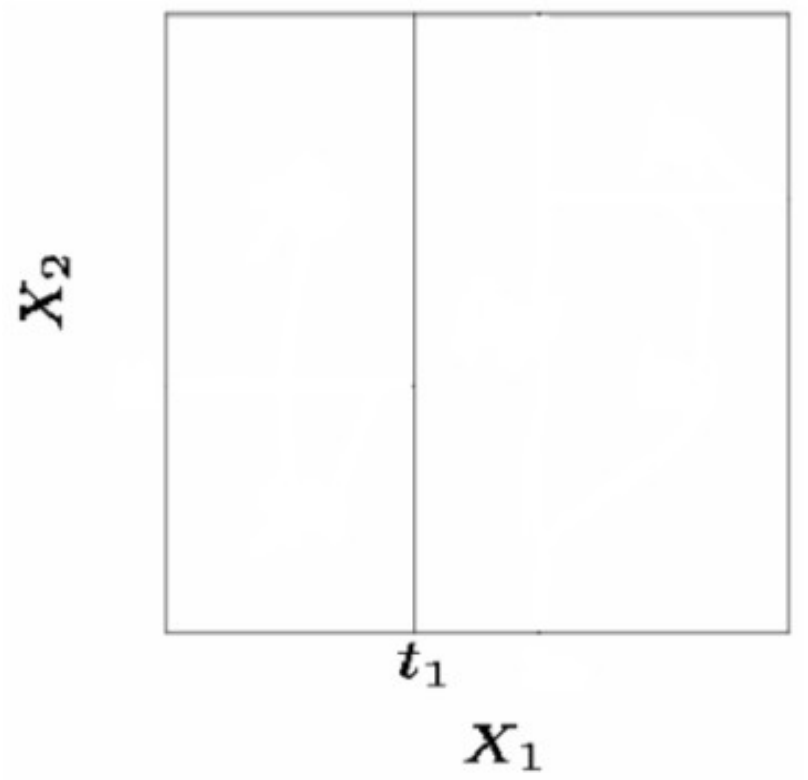
$$GR = -\frac{IG}{Info} \qquad Info = -\sum_{i} \frac{|p_i|}{N}\, log_2 \frac{|p_i|}{N}$$

What is the measure considered for Regression Trees?

Residual Sum of Squares (RSS) ⟶ $$\sum_{j=1}^{J}\sum_{i\in R_j} (y_i - \hat{y}_{R_j})^2,$$

Generally we create the partitions by iteratively splitting one of the X variables into two regions.

First split on: $X_1 = t_1$



$$R_1(j,s) = \{X | X_j < s\} \text{ and } R_2(j,s) = \{X | X_j \geq s\},$$

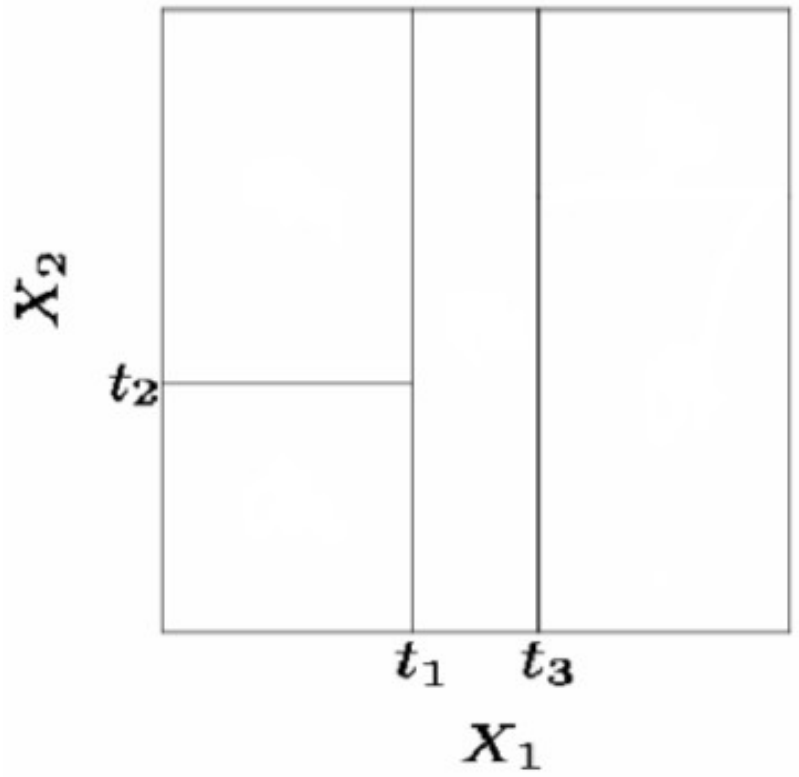
$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

Generally we create the partitions by iteratively splitting one of the ***X*** variables into two regions.

First split on: $X_1 = t_1$

If $X_1 < t_1$ split on: $X_2 = t_2$

If $X_1 >= t_1$ split on: $X_1 = t_3$



$$R_1(j,s) = \{X | X_j < s\} \text{ and } R_2(j,s) = \{X | X_j \geq s\},$$

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

Generally we create the partitions by iteratively splitting one of the ***X*** variables into two regions.

First split on: $X_1 = t_1$

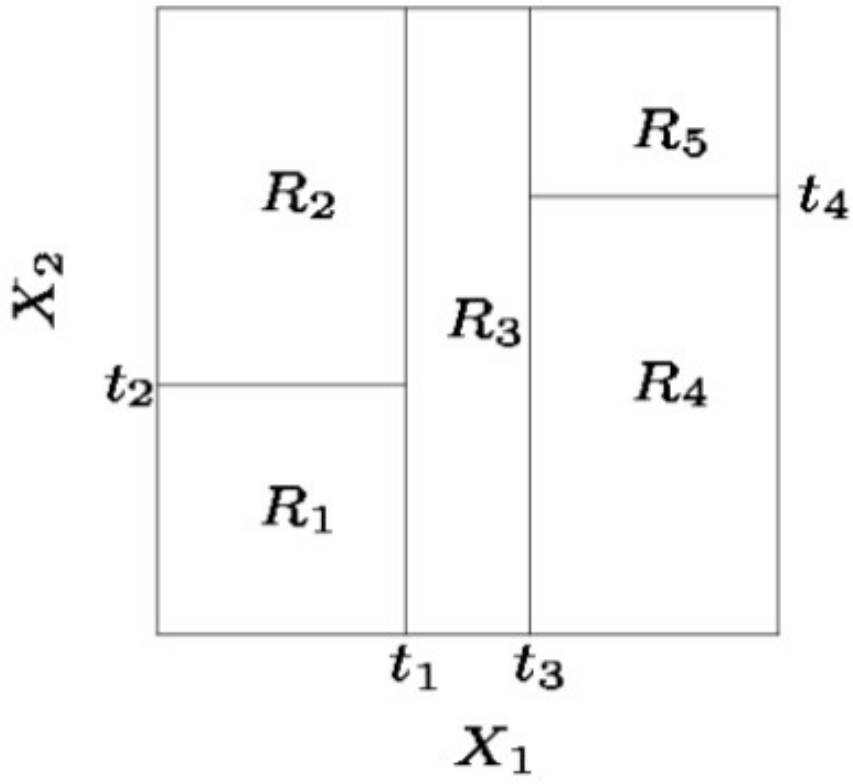
If $X_1 < t_1$ split on: $X_2 = t_2$

If $X_1 >= t_1$ split on: $X_1 = t_3$

If $X_1 >= t_3$ split on: $X_2 = t_4$

…

until the tree correctly predicts the sample ...



$$R_1(j,s) = \{X | X_j < s\} \text{ and } R_2(j,s) = \{X | X_j \geq s\},$$

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

Generally we create the partitions by iteratively splitting one of the X variables into two regions.

First split on: $X_1=t_1$

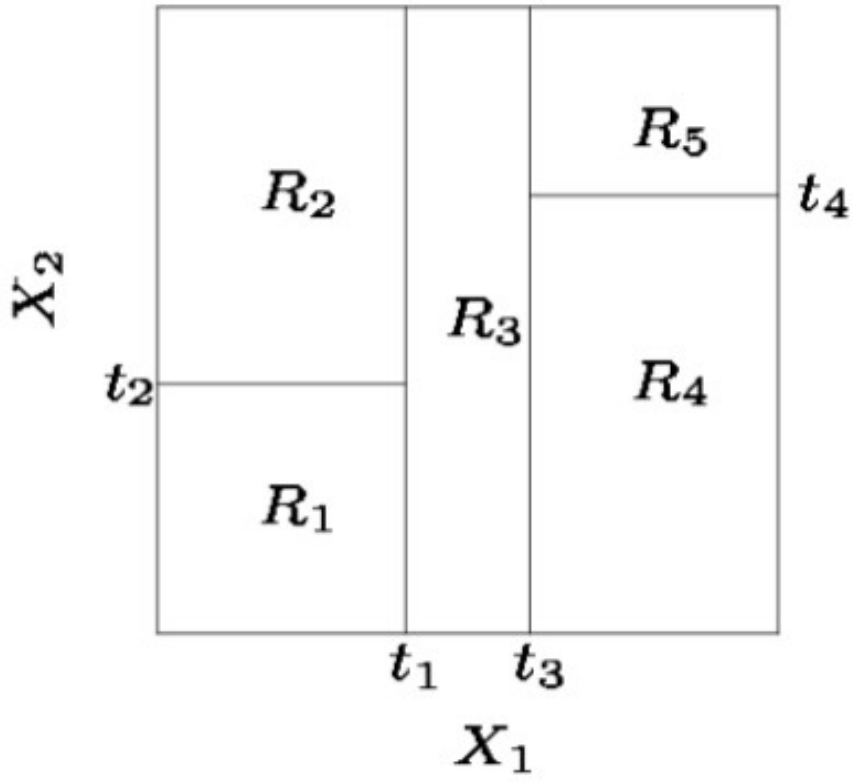
If $X_1<t_1$ split on: $X_2=t_2$

If $X_1>=t_1$ split on: $X_1=t_3$

If $X_1>=t_3$ split on: $X_2=t_4$

…

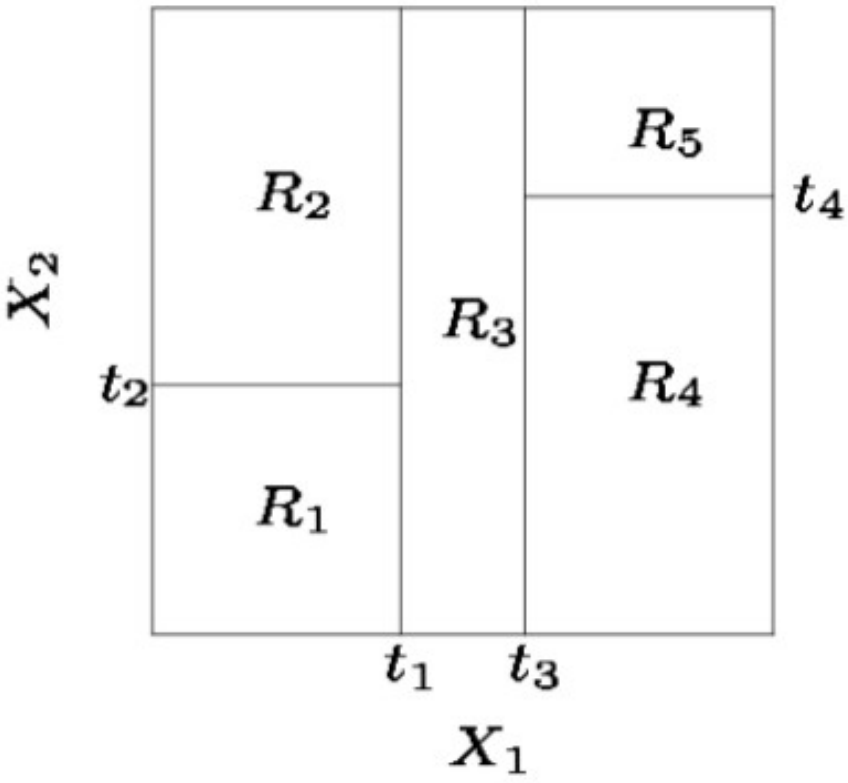
until the tree correctly predicts the sample or some stop criteria has been reached:

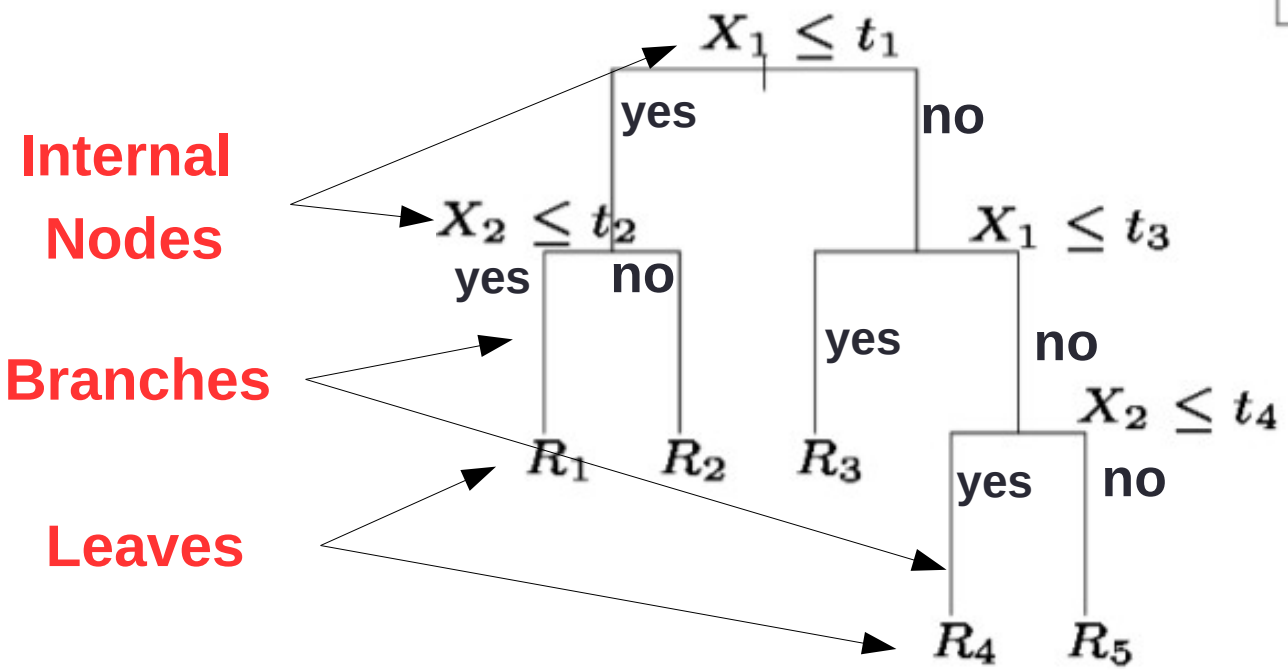


- Reduction cost too small vs Complexity
- Max. depth
- Purity
- Sub-sample too small

Types of trees:

- **Classification:** discrete outputs
  - CLS, ID3, C4.5, ID4, ….
- **Regression:** continuous outputs
  - **CART**, **M5**, **M5'**, ….





```
library(MASS)
? Boston
```

Housing Values in Suburbs of Boston

Description:

The Boston data frame has 506 rows and 14 columns.

Format:

This data frame contains the following columns:

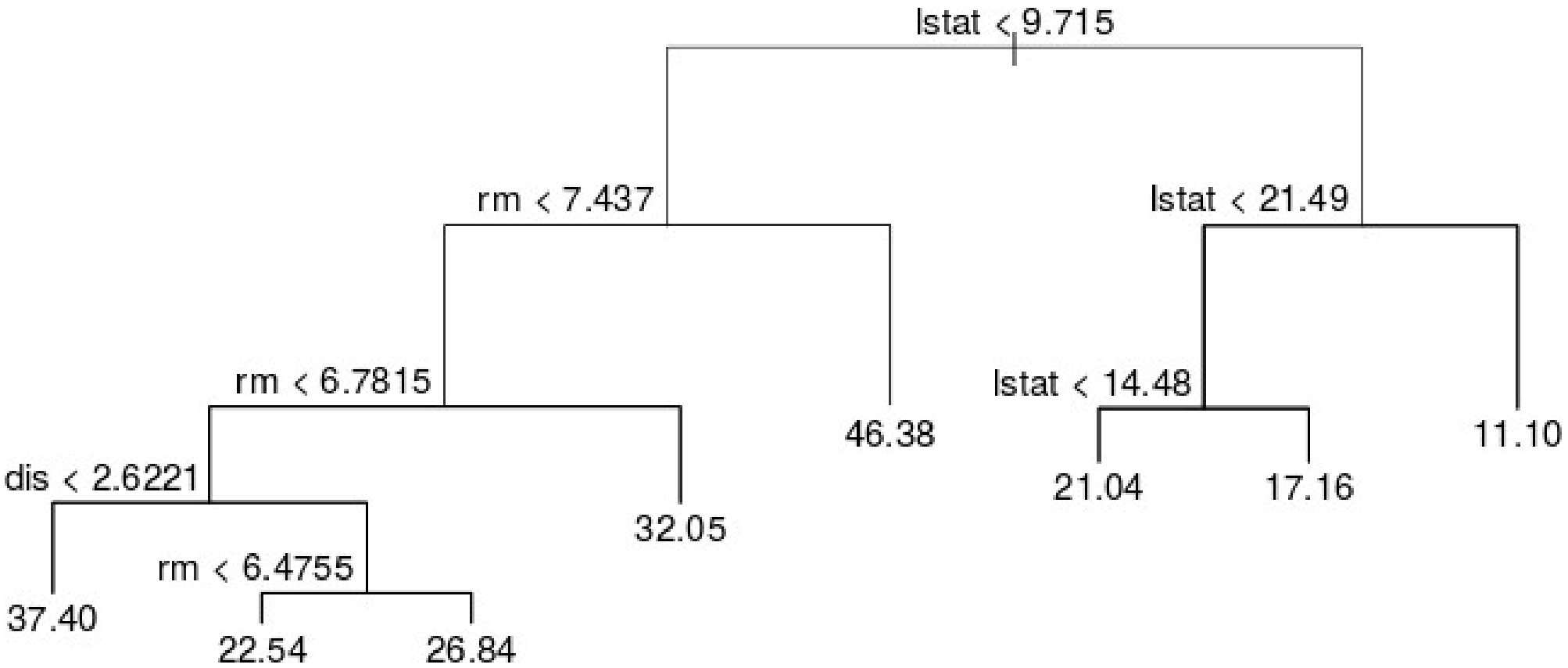
crim - per capita crime rate by town.
zn - proportion of residential land zoned for lots over 25000 sq.ft.
indus - proportion of non-retail business acres per town.
chas - Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
nox - nitrogen oxides concentration (parts per 10 million).
rm - average number of rooms per dwelling.
age - proportion of owner-occupied units built prior to 1940.
dis - weighted mean of distances to five Boston employment centres.
rad - index of accessibility to radial highways.
tax - full-value property-tax rate per 10000 $.
ptratio - pupil-teacher ratio by town.
black - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
lstat - lower status of the population (percent).
medv - median value of owner-occupied homes in 1000s $.

```r
library(tree)
set.seed(1)
indTrain <- sample(1:nrow(Boston), nrow(Boston)/2) ## Splitting the sample
tree.boston <- tree(medv~., Boston, subset = indTrain) ## Adjusting the regression tree.
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8
## Residual mean deviance:  12.65 = 3099 / 245
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -14.10000  -2.04200  -0.05357   0.00000   1.96000  12.60000
```

```
library(tree)
set.seed(1)
indTrain <- sample(1:nrow(Boston), nrow(Boston)/2) ## Splitting the sample
tree.boston <- tree(medv~., Boston, subset = indTrain) ## Adjusting the regression tree.
summary(tree.boston)

plot(tree.boston)
text(tree.boston, pretty = 0)
```



```
library(tree)
set.seed(1)
indTrain <- sample(1:nrow(Boston), nrow(Boston)/2) ## Splitting the sample
tree.boston <- tree(medv~., Boston, subset = indTrain) ## Adjusting the regression tree.
summary(tree.boston)
? tree.control
```

tree.control

Use:

tree.control(nobs, mincut = 5, minsize = 10, mindev = 0.01)

Arguments:

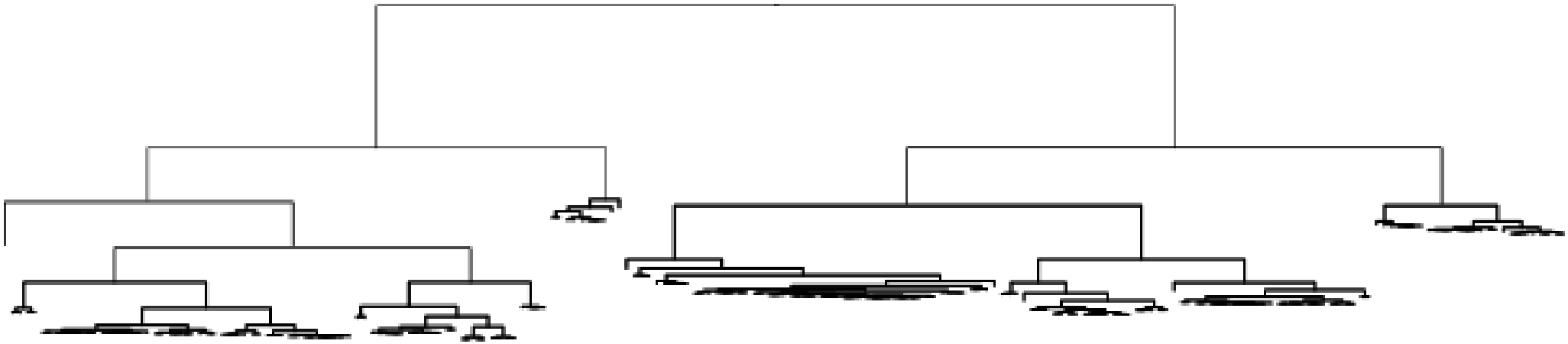
nobs - the number of observations in the training set.

mincut - the minimum number of observations to include in either child node.

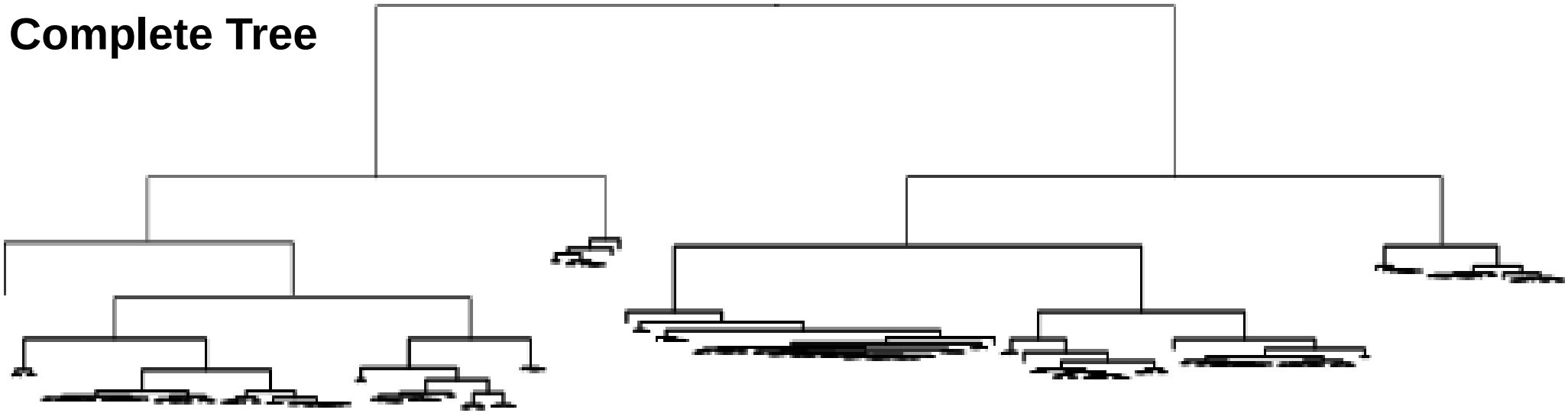
minsize - the smallest allowed node size.

mindev - the within-node deviance must be at least this times that of the root node for the node to be split.

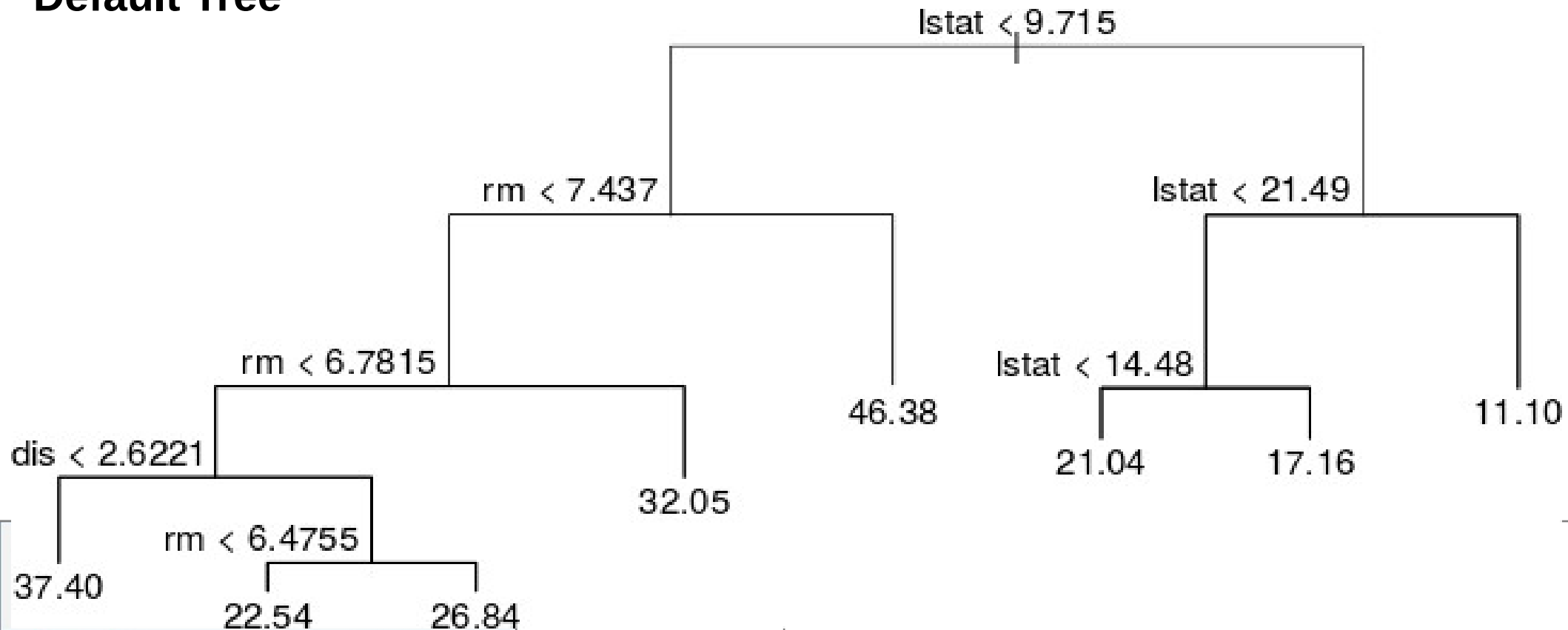
```
library(tree)
set.seed(1)
indTrain <- sample(1:nrow(Boston), nrow(Boston)/2) ## Splitting the sample
## Building the deepest tree.
deepTree.boston <- tree(medv~., Boston, subset = indTrain, split = "deviance", control =
tree.control(length(indTrain), mincut = 1, minsize = 2, mindev = 0))
plot(deepTree.boston)
```



Complete Tree

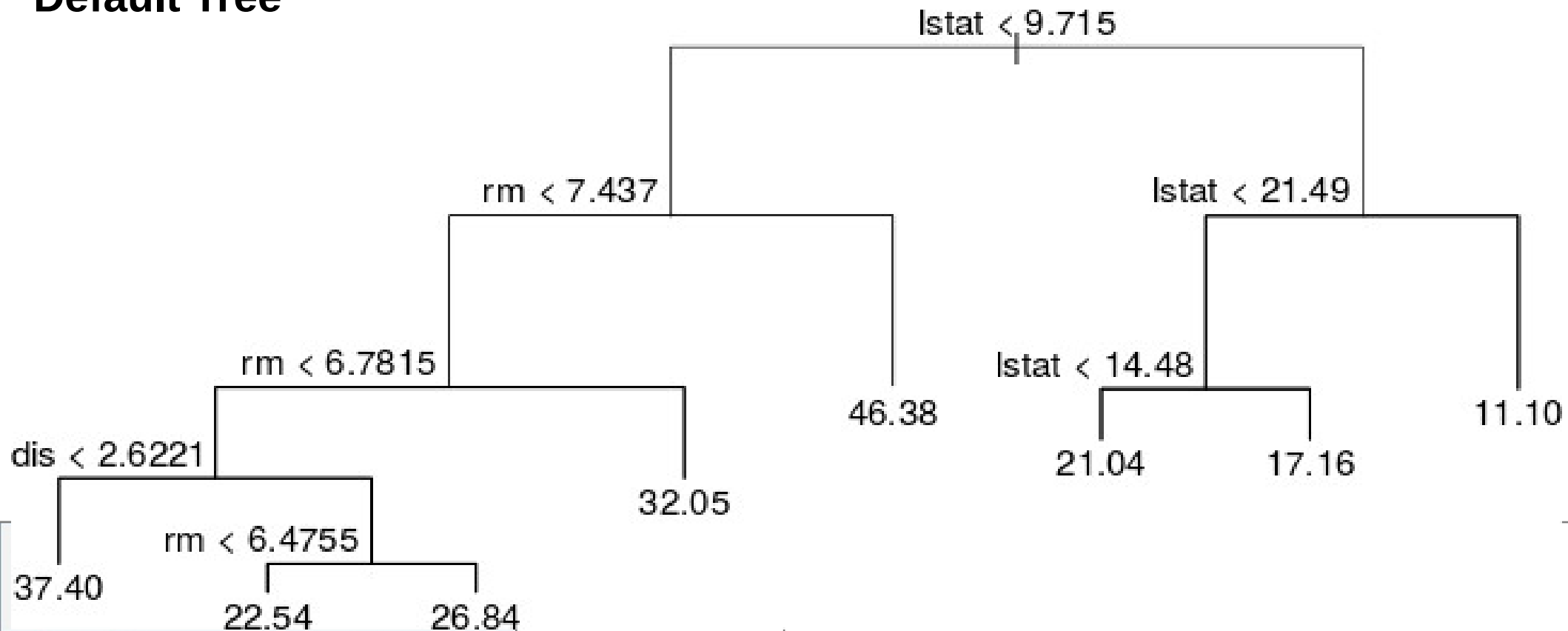


Default Tree



```
library(tree)
set.seed(1)
indTest <-  setdiff(1:nrow(Boston), indTrain) ## Splitting the sample
## Verification (Default Tree):
train.boston <- predict(tree.boston, newdata = Boston[indTrain,])
test.boston <- predict(tree.boston, newdata = Boston[indTest,])
## RMSE:
sqrt(mean((Boston$medv[indTrain]-train.boston)^2, na.rm = TRUE))
## [1]  3.499638
sqrt(mean((Boston$medv[indTest]-test.boston)^2, na.rm = TRUE))
## [1] 5.004557
```

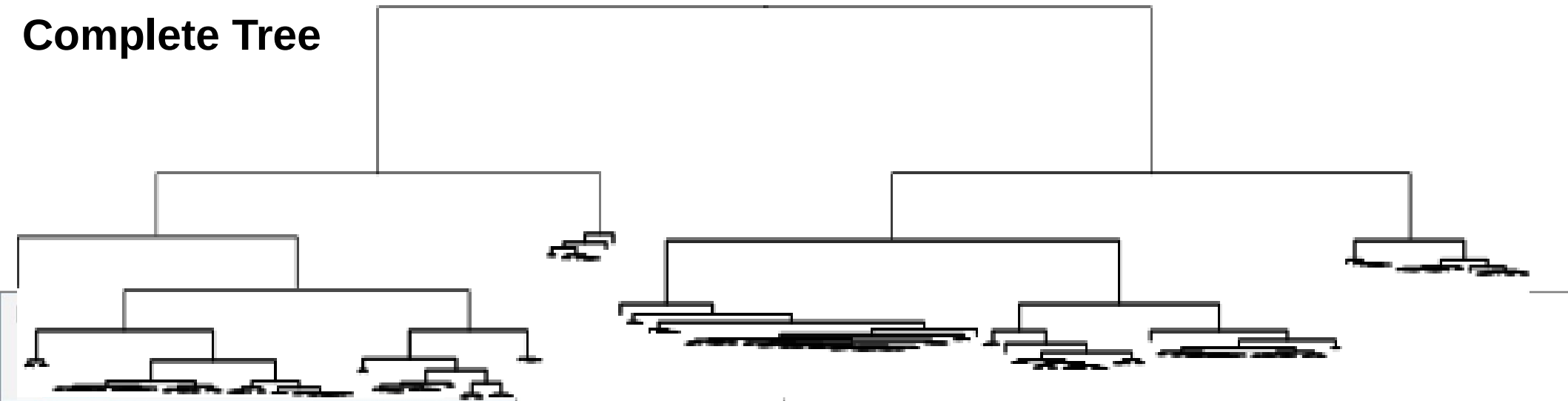
Default Tree



```r
library(tree)
set.seed(1)
indTest <-  setdiff(1:nrow(Boston), indTrain) ## Splitting the sample
## Verification (Default Tree):
train.boston <- predict(tree.boston, newdata = Boston[indTrain,])
test.boston <- predict(tree.boston, newdata = Boston[indTest,])
## RMSE:
sqrt(mean((Boston$medv[indTrain]-train.boston)^2, na.rm = TRUE))
## [1]  3.499638
sqrt(mean((Boston$medv[indTest]-test.boston)^2, na.rm = TRUE))
## [1] 5.004557

## Verification (Deepest Tree):
train1.boston <- predict(deepTree.boston, newdata = Boston[indTrain,])
test1.boston <- predict(deepTree.boston, newdata = Boston[indTest,])
## RMSE:
sqrt(mean((Boston$medv[indTrain]-train1.boston)^2, na.rm = TRUE))
## [1] 0.03976214
sqrt(mean((Boston$medv[indTest]-test1.boston)^2, na.rm = TRUE))
## [1] 4.465788
```

Complete Tree

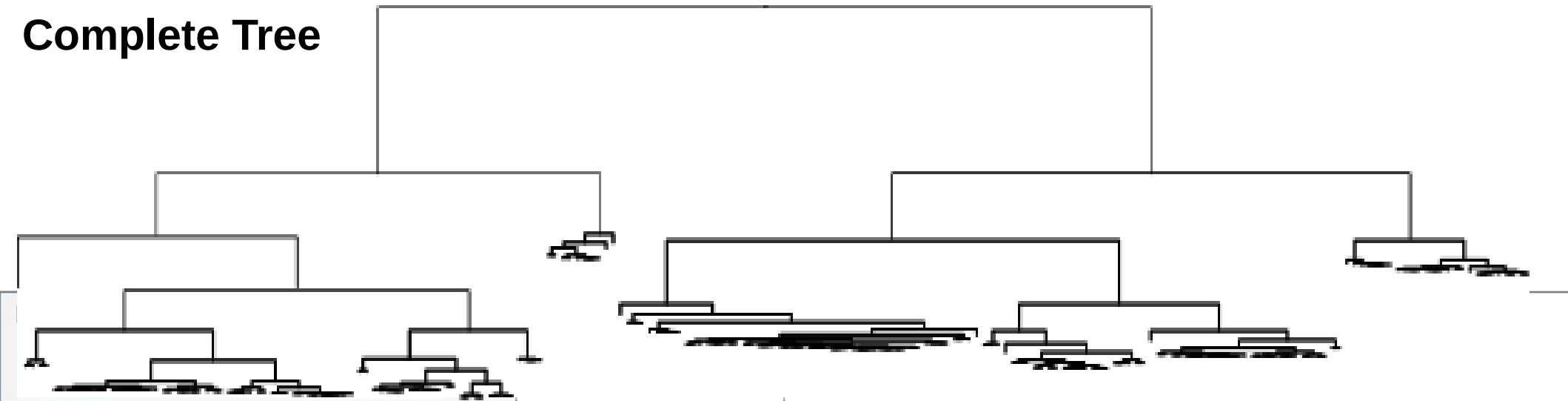


```
library(tree)
set.seed(1)
indTest <-  setdiff(1:nrow(Boston), indTrain) ## Splitting the sample
## Verification (Default Tree):
train.boston <- predict(tree.boston, newdata = Boston[indTrain,])
test.boston <- predict(tree.boston, newdata = Boston[indTest,])
## RMSE:
sqrt(mean((Boston$medv[indTrain]-train.boston)^2, na.rm = TRUE))
## [1]  3.499638
sqrt(mean((Boston$medv[indTest]-test.boston)^2, na.rm = TRUE))
## [1] 5.004557

## Verification (Deepest Tree):
train1.boston <- predict(deepTree.boston, newdata = Boston[indTrain,])
test1.boston <- predict(deepTree.boston, newdata = Boston[indTest,])
## RMSE:
sqrt(mean((Boston$medv[indTrain]-train1.boston)^2, na.rm = TRUE))
## [1] 0.03976214
sqrt(mean((Boston$medv[indTest]-test1.boston)^2, na.rm = TRUE))
## [1] 4.465788
```

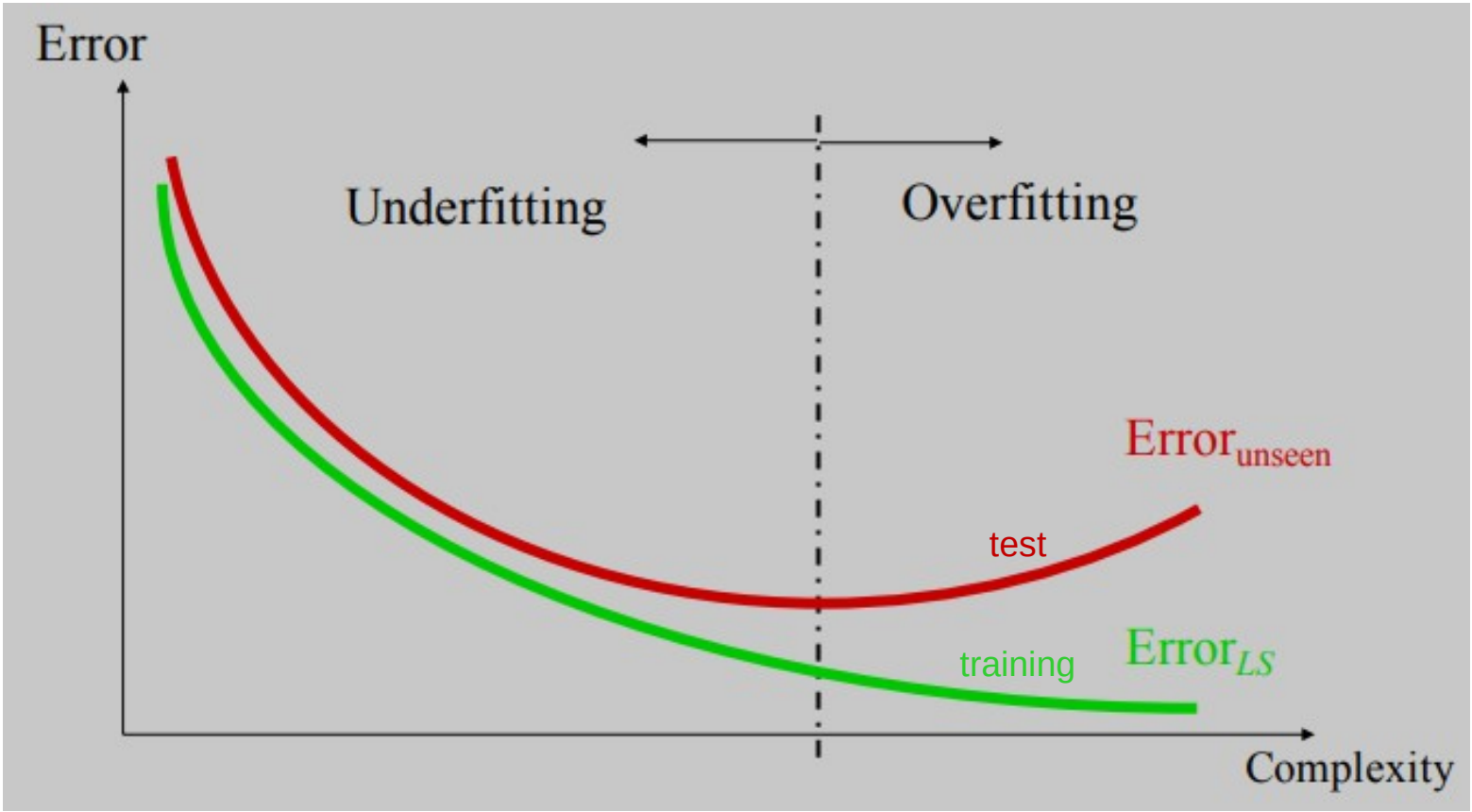
Overfitting

Complete Tree



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.



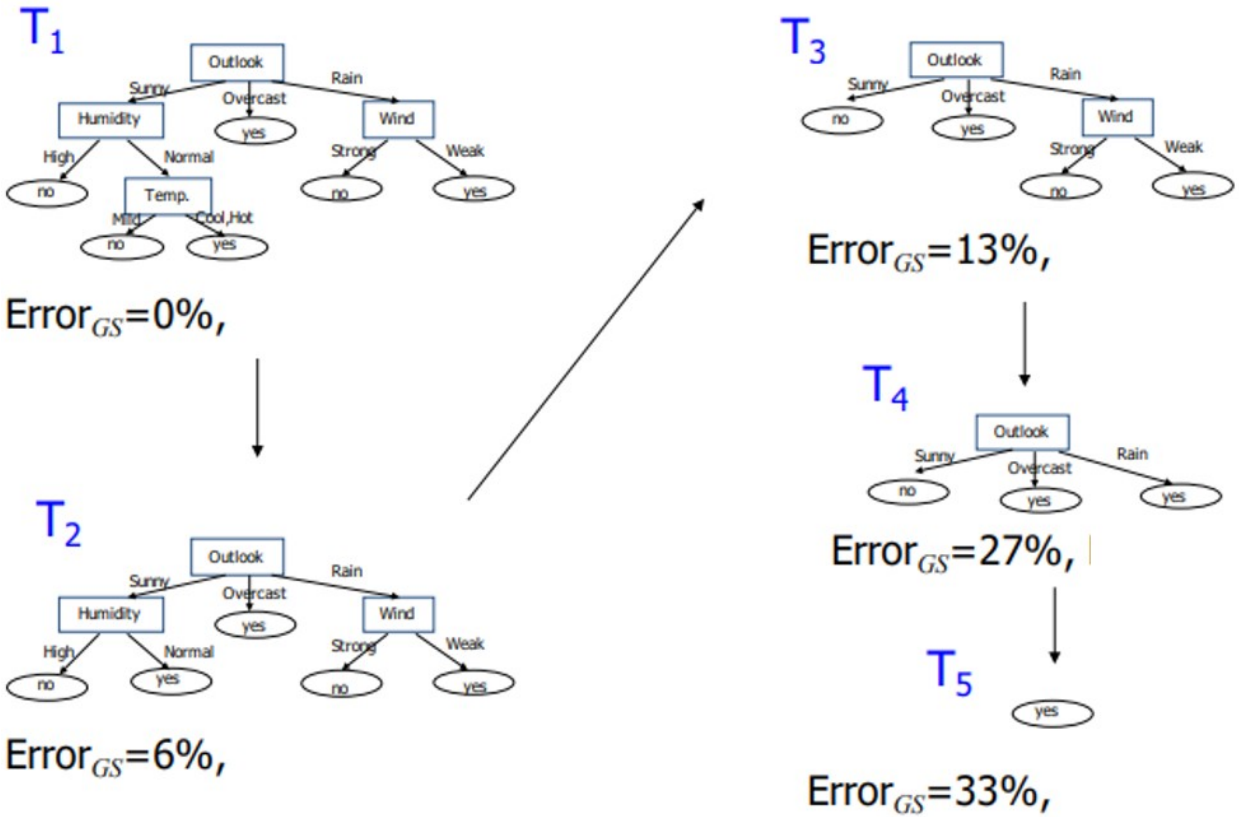
How can we avoid overfitting?

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**).



```
library(tree)
set.seed(1)
indTrain <- sample(1:nrow(Boston), nrow(Boston)/2) ## Splitting the sample
tree.boston <- tree(medv~., Boston, subset = indTrain) ## Adjusting the regression tree.
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8
## Residual mean deviance:  12.65 = 3099 / 245
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd
## -14.10000  -2.04200  -0.05357   0.00000   1.9
```

Regression trees: CART

- **Output** corresponds to the response **mean** value.
- Valid for any kind of attributes

Residual Sum of Squares (RSS) →

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

```r
library(tree)
set.seed(1)
indTrain <- sample(1:nrow(Boston), nrow(Boston)/2) ## Splitting the sample
tree.boston <- tree(medv~., Boston, subset = indTrain) ## Adjusting the regression tree.
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8
## Residual mean deviance:  12.65 = 3099 / 245
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd
## -14.10000  -2.04200  -0.05357   0.00000   1.9
```

Regression trees: CART

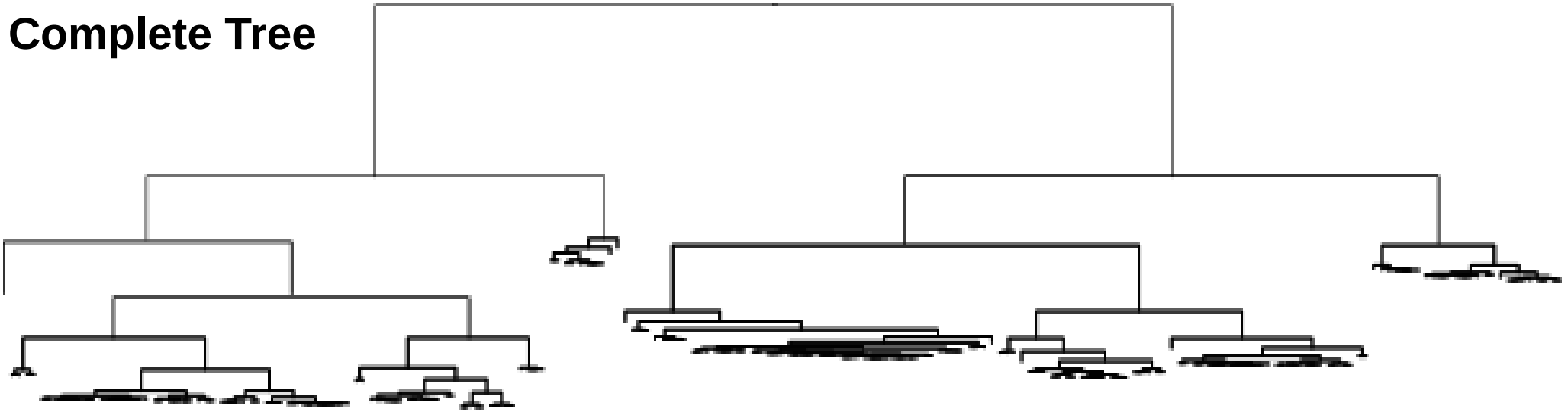
- **Output** corresponds to the response **mean** value.
- Valid for any kind of attributes

Residual Sum of Squares (RSS) →

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

We want to select a subtree that leads to the lowest test error rate

Complete Tree



Given a subtree, we can estimate the test error rate

↓

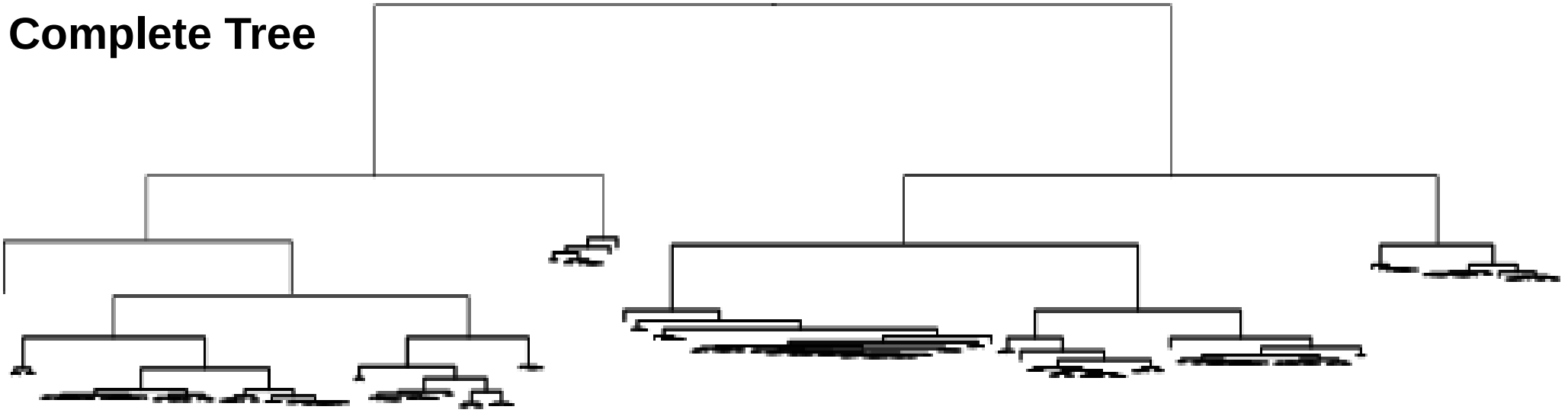
Choose the subtree with the lowest test error rate

Residual Sum of Squares (RSS) →

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

We want to select a subtree that leads to the lowest test error rate

Complete Tree



Given a subtree, we can estimate the test error rate

Computationally expensive ¡¡¡

Choose the subtree with the lowest test error rate

Residual Sum of Squares (RSS) $\longrightarrow$ $$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

We want to select a subtree that leads to the lowest test error rate

Cost complexity Pruning → Regularization

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Residual Sum of Squares (RSS) ⟶ $$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

We want to select a subtree that leads to the lowest test error rate

Cost complexity Pruning → Regularization

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Number of terminal nodes

Controls a trade-off between the subtree's complexity and its fit to the training data

Residual Sum of Squares (RSS) → $$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

We want to select a subtree that leads to the lowest test error rate

Cost complexity Pruning → Regularization

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Number of terminal nodes

Controls a trade-off between the subtree's complexity and its fit to the training data

For each value of the regularization parameter exists a subtree minimizing the expression above.

↓

Minimization problem ↔ Lagrange Multipliers

Residual Sum of Squares (RSS)

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

We want to select a subtree that leads to the lowest test error rate

- **Pros:**
  - Trees are very easy to explain to people (probably even easier than linear regression)
  - Trees can be plotted graphically, and are easily interpreted even by non-expert
  - Trees can easily handle qualitative predictors without the need to create dummy variables.
  - They work fine on both classification and regression problems
- **Cons:**
  - Trees don’t have the same prediction accuracy as some of the more complicated approaches that we examine in this course.
  - Trees tend to overfit.

```r
library(tree)
set.seed(1)
indTrain <- sample(1:nrow(Boston), nrow(Boston)/2) ## Splitting the sample
tree.boston <- tree(medv~., Boston, subset = indTrain) ## Adjusting the regression tree.
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8
## Residual mean deviance:  12.65 = 3099 / 245
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd
## -14.10000  -2.04200  -0.05357   0.00000   1.9
```

Model trees: M5, M5', ...

- Output corresponds to a linear regression model of the instances that reach the leaf.

Residual Sum of Squares (RSS) →

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

```
library(tree)
set.seed(1)
indTrain <- sample(1:nrow(Boston), nrow(Boston)/2) ## Splitting the sample
tree.boston <- tree(medv~., Boston, subset = indTrain) ## Adjusting the regression tree.
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8
## Residual mean deviance:  12.65 = 3099 / 245
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd
## -14.10000  -2.04200  -0.05357   0.00000   1.9
```

Model trees: M5, M5', ...

- Output corresponds to a linear regression model of the instances that reach the leaf.

M5 is included in the caret R-package (Cubist-implementation)

```
library(caret)
if (!require(Cubist)) install.packages("Cubist")
? models
autoTree <- train(form = medv~., data= Boston, subset = indTrain, method = "cubist")
summary(autoTree)
```

Rules ↔ Decision Trees

- **Pros:**
  - Trees are very easy to explain to people (probably even easier than linear regression)
  - Trees can be plotted graphically, and are easily interpreted even by non-expert
  - Trees can easily handle qualitative predictors without the need to create dummy variables.
  - They work fine on both classification and regression problems
- **Cons:**
  - Trees don't have the same prediction accuracy as some of the more complicated approaches that we examine in this course.

    By aggregating many decision trees, the predictive performance of trees can be substantially improved.

Bagging, Random Forest, Boosting

- **Which model is better?**
  - If the relationship between the predictors and response is linear, then classical linear models such as linear regression would outperform regression trees
  - On the other hand, if the relationship between the predictors is non-linear, then decision trees would outperform classical approaches
  - **Categorical** variables can be used for trees.
  - Trees are more easily **interpretable** than linear models.
  - **Visualization.** The graphical representation of the trees is very useful.

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j, \qquad f(X) = \sum_{m=1}^{M} c_m \cdot 1_{(X \in R_m)}$$

- **Which model is better?**
  - If the relationship between the predictors and response is linear, then classical linear models such as linear regression would outperform regression trees
  - On the other hand, if the relationship between the predictors is non-linear, then decision trees would outperform classical approaches
  - **Categorical** variables can be used for trees.
  - Trees are more easily **interpretables** than linear models.
  - **Visualization.** The graphical representation of the trees is very useful.
  - **Top row:** the true decision boundary is linear
    - **Left:** linear model (good)
    - **Right**: decision tree
  - **Bottom row:** the true decision boundary is non-linear
    - **Left:** linear model
    - **Right**: decision tree (good)