



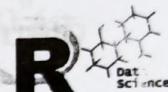
# R for Data Science

## Bài 19: *Decision tree*

Phòng LT & Mạng

[https://csc.edu.vn/lap-trinh-va-cSDL/R-Programming-Language-for-Data-Science\\_190](https://csc.edu.vn/lap-trinh-va-cSDL/R-Programming-Language-for-Data-Science_190)

2019



## Nội dung

### 1. Giới thiệu

### 2. Decision Tree



## Decision Tree



## Giới thiệu

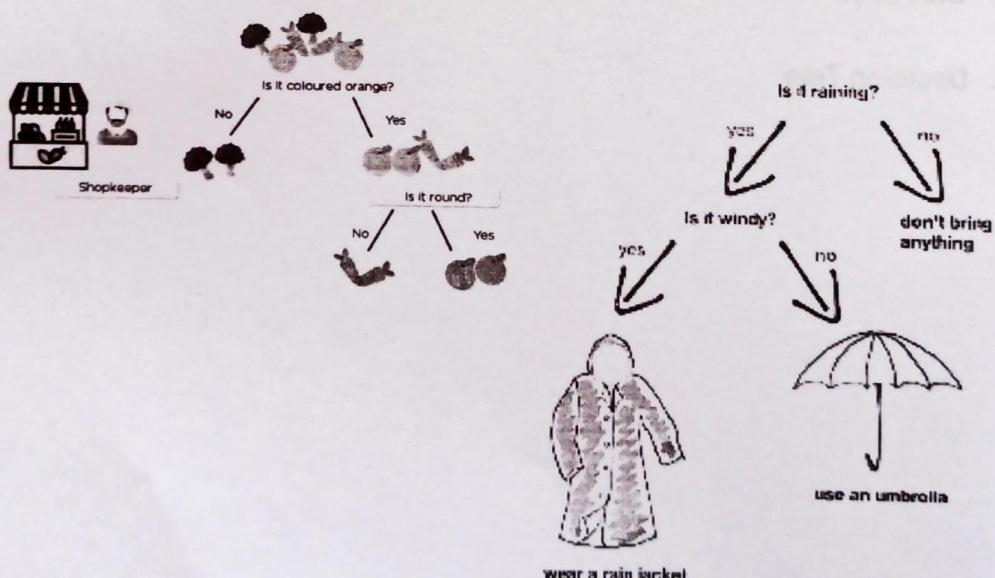
- Decision Tree là một thuật toán thuộc nhóm Supervised Learning được sử dụng cho cả classification và regression.
  - Là biểu đồ hiển thị các lựa chọn và kết quả dưới dạng cây. Các node trên biểu đồ đại diện cho một sự kiện hoặc lựa chọn và các cạnh của biểu đồ thể hiện các quy luật hoặc điều kiện quyết định.



R programming language for Data Science

3

## Decision Tree



R programming language for Data Science

12 of 12 pages

4

## □ Ý tưởng

- Tìm các tính năng chứa thông tin mô tả có chứa “thông tin” nhất về target feature và sau đó chia tập dữ liệu dọc theo các giá trị của các feature này sao cho các giá trị target feature cho các tập con (sub\_dataset) càng “thuần khiết” càng tốt.  
=> các feature mô tả dẫn đến target feature ‘thuần khiết’ nhất được gọi là có “thông tin” nhất”.

R programming language for Data Science

5

# Decision Tree

- Quá trình tìm kiếm các feature có thông tin nhất (“most informative”) được thực hiện cho đến khi chúng ta hoàn thành điều kiện dừng khi cuối cùng kết thúc ở nút lá (**leaf node**). Các nút lá chứa các thông tin dự đoán mà chúng ta sẽ thực hiện cho các thực thể mới được trình bày cho trained model.
- Điều này khả thi vì model đã học được cấu trúc cơ bản của dữ liệu huấn luyện (training data), và do đó có thể đưa ra một số giả định, đưa ra các dự đoán về giá trị target feature (class) của các thực thể chưa biết.

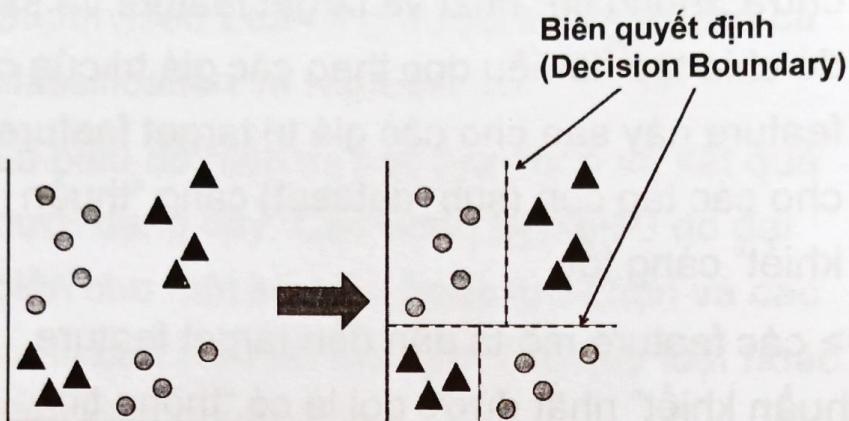
R programming language for Data Science

6

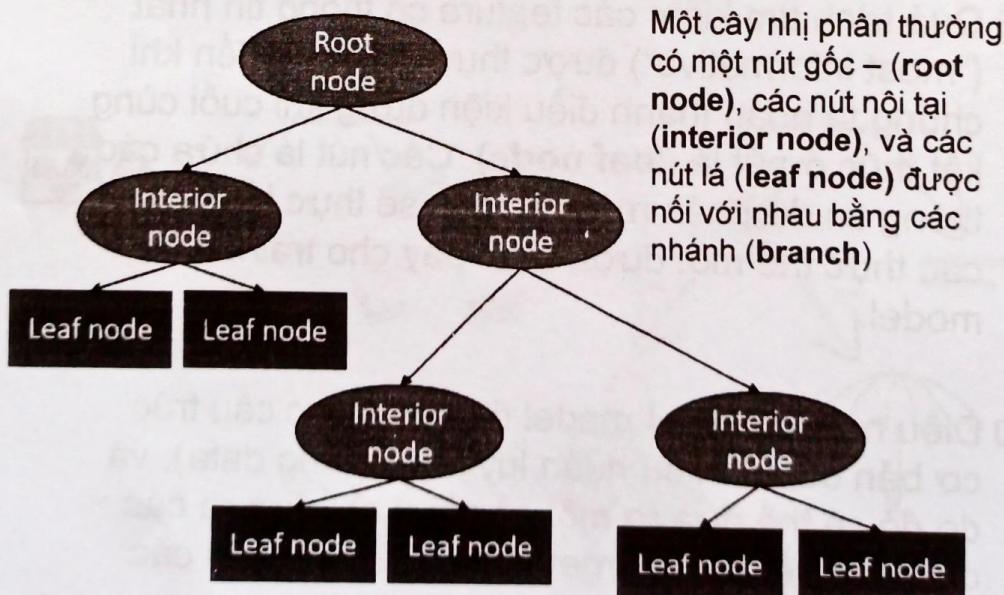
## Decision Tree

### □ Ý tưởng

- Chia dữ liệu vào những vùng “thuần nhất”



## Decision Tree



# Nội dung



## 1. Giới thiệu

## 2. Decision Tree



## Decision Tree



### □ Xây dựng cây quyết định

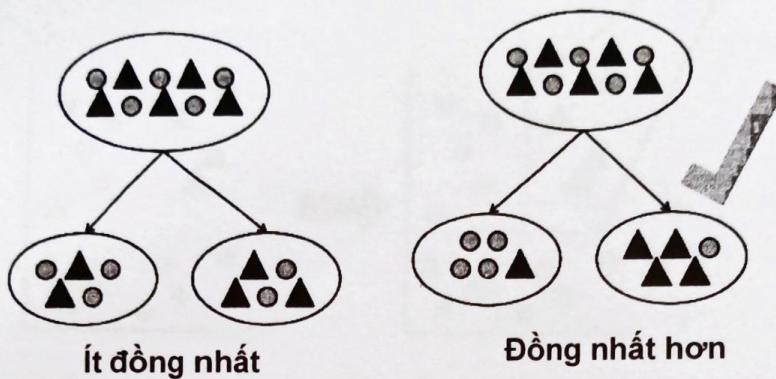
- Bắt đầu với tất cả các sample tại một node.
- Các sample phân vùng dựa trên input để tạo tập con thuần khiết nhất (purest subset)
- Lặp lại quá trình phân vùng dữ liệu vào các tập con thuần khiết hơn.



## Decision Tree

- Làm thế nào để xác định được phân chia tốt nhất?

- Mong muốn tập con càng đồng nhất càng tốt



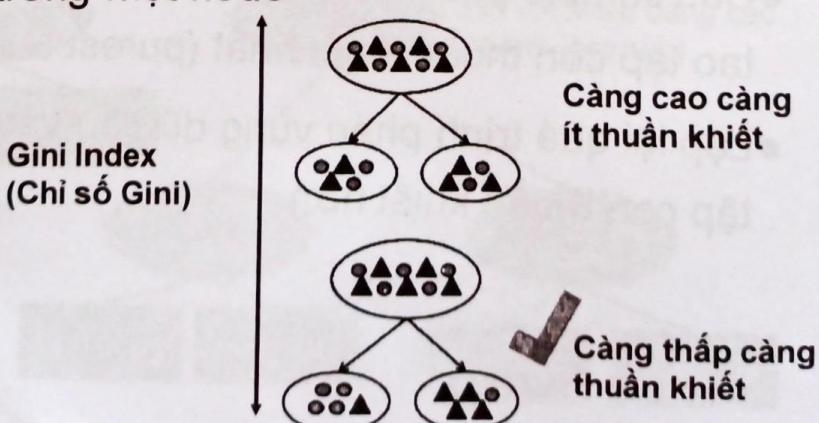
R programming language for Data Science

11

## Decision Tree

- Đo mức độ không đồng nhất

- Để so sánh các cách khác nhau khi cắt dữ liệu trong một node



R programming language for Data Science

12

# Decision Tree

## ❑ Gini

- Làm việc với categorical target variable (Ví dụ: “Success” hay “Failure”, “Pass” hay “Fail”)
- Thực hiện chia cây theo Binary splits (2 nhánh)
- Gini index càng thấp thì tính đồng nhất càng cao
- CART (Classification and Regression Tree) sử dụng Gini method để tạo binary splits.



# Thuật toán

## ❑ Gini

$$Gini = 1 - \sum_{i=1}^C (p_i)^2$$

- Tính gini cho từng thuộc tính, theo class kết quả.
- Thuộc tính nào trong nhóm các thuộc tính có gini bé nhất => chọn thuộc tính đó để chia nhánh cho cây.



## Thuật toán

● Ví dụ:

| Tid | Refund | Marital Status | Taxable Income | Evade |
|-----|--------|----------------|----------------|-------|
| 1   | Yes    | Single         | 125K           | No    |
| 2   | No     | Married        | 100K           | No    |
| 3   | No     | Single         | 70K            | No    |
| 4   | Yes    | Married        | 120K           | No    |
| 5   | No     | Divorced       | 95K            | Yes   |
| 6   | No     | Married        | 60K            | No    |
| 7   | Yes    | Divorced       | 220K           | No    |
| 8   | No     | Single         | 85K            | Yes   |
| 9   | No     | Married        | 75K            | No    |
| 10  | No     | Single         | 90K            | Yes   |

R programming language for Data Science

15

## Thuật toán

### □ Tính gini index cho Refund

● Refund: Yes = 3, No = 7

- Refund = Yes & Output = Yes: 0
- Refund = Yes & Output = No : 3/3 = 1

$$\Rightarrow Gini(0,3) = 1 - (0 + 1*1) = 0$$

- Refund = No & Output = Yes: 3/7

- Refund = No & Output = No : 4/7

$$\Rightarrow Gini(3,4) = 1 - (3/7*3/7 + 4/7*4/7) = 0.49$$

$$\Rightarrow Gini(\text{Target, Color}) = 3/10*0 + 7/10*0.49 = 0.34$$

| Refund | Evade |
|--------|-------|
| Yes    | No    |
| No     | No    |
| No     | No    |
| Yes    | No    |
| No     | Yes   |
| No     | No    |
| Yes    | No    |
| No     | Yes   |
| No     | No    |
| No     | Yes   |

R programming language for Data Science

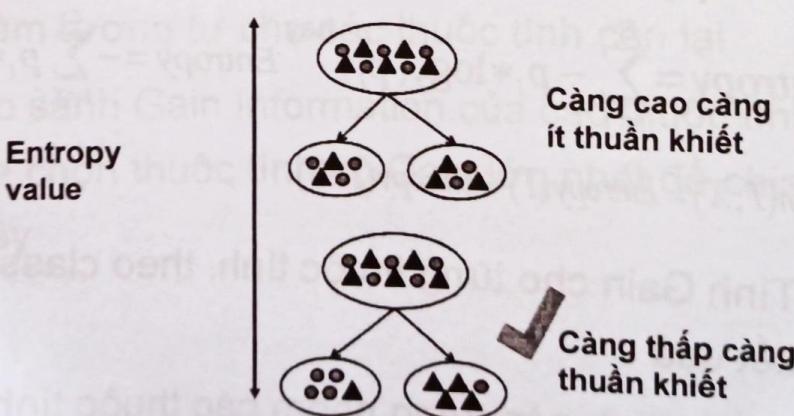
16

## ❑ Tính gini index cho thuộc tính

- Làm tương tự cho các thuộc tính còn lại
- So sánh gini của các thuộc tính => chọn thuộc tính có gini nhỏ nhất để chia cây



## ❑ Entropy và Information Gain



Entropy = 0: mẫu hoàn toàn thuần khiết, entropy = 1: mẫu không thuần khiết (trộn đều). Tùy vào số lượng các class trong dataset, entropy có thể lớn hơn 1, tuy nhiên ý nghĩa như nhau: entropy càng lớn thì càng ít thuần khiết.



## Thuật toán

### □ Information Gain

- Dựa vào việc giảm entropy sau khi data-set được phân chia dựa trên một thuộc tính.
- Việc xây dựng decision tree dựa vào việc tìm thuộc tính trả về information gain cao nhất (các nhánh đồng nhất nhất).



## Thuật toán

### □ Entropy & Gain

$$\text{Entropy} = \sum_{i=1}^C -p_i * \log_2(p_i) \quad \text{hay} \quad \text{Entropy} = -\sum_{i=1}^C p_i * \log_2(p_i)$$

$$\text{Gain}(T, X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$

- Tính Gain cho từng thuộc tính, theo class kết quả.
- Thuộc tính nào trong nhóm các thuộc tính có Gain lớn nhất => chọn thuộc tính đó để chia nhánh cho cây.



# Thuật toán

## ☐ Tính Gain cho Refund

- Refund: Yes = 3, No = 7

- Refund = Yes & Output = Yes: 0
- Refund = Yes & Output = No : 3/3 = 1
- ⇒  $\text{Entropy}(0,3) = -1 * (0 * \log_2(0) + 1 * \log_2(1))$
- Refund = No & Output = Yes: 3/7
- Refund = No & Output = No : 4/7
- ⇒  $\text{Entropy}(3,4) = -1 * (3/7 * \log_2(3/7) + 4/7 * \log_2(4/7))$
- ⇒  $\text{Entropy}(\text{Target}, \text{Refund}) = 3/10 * \text{Entropy}(0,3) + 7/10 * \text{Entropy}(3,4)$
- ⇒  $\text{Gain} = \text{Entropy}(\text{Target}) - \text{Entropy}(\text{Target}, \text{Refund})$  (Với  
 $\text{Entropy}(\text{Target} (3, 7) = -1 * (p_{\text{Yes}} * \log_2(p_{\text{Yes}}) + p_{\text{No}} * \log_2(p_{\text{No}}))$

 R programming language for Data Science

21

# Thuật toán

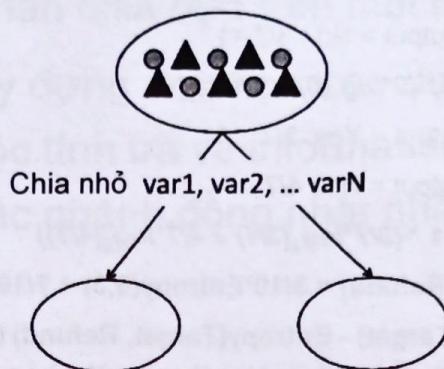
## ☐ Tính Entropy cho thuộc tính

- Làm tương tự cho các thuộc tính còn lại
- So sánh Gain Information của các thuộc tính  
 => chọn thuộc tính có Gain lớn nhất để chia cây

## Decision Tree

### ❑ Variable nào sẽ được chia nhỏ?

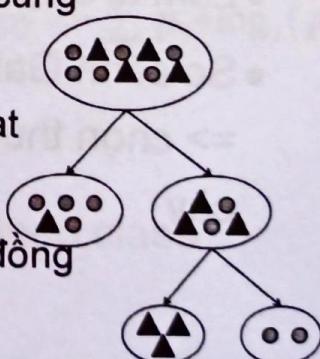
- Chia nhỏ trên tất cả các biến được kiểm tra.



## Decision Tree

### ❑ Khi nào dừng chia nhỏ node?

- Khi tất cả ( $X\%$ ) của các sample có cùng class label
- Số lượng các sample trong node đạt đến mức tối thiểu
- Thay đổi trong đo lường độ không đồng nhất nhỏ hơn ngưỡng (threshold)
- Đạt được độ sâu cây tối đa
- ...



## □ Ưu điểm

- Dễ hiểu, dễ mô hình
- Không cần chuẩn hóa tính năng
- Có thể áp dụng trong cả Classification và Regression
- Có thể mô hình các quan hệ phi tuyến (non-linear relationship)
- Có thể mô hình các tương tác giữa các tính năng mô tả khác nhau



# Decision Tree

## □ Khuyết điểm

- Nếu tính năng liên tục (continuous feature) được sử dụng thì cây có thể trở nên rất lớn và ít diễn giải
- Những thay đổi nhỏ trong dữ liệu có thể dẫn đến cây hoàn toàn khác (có thể dùng random forest để khắc phục)
- Nếu số lượng đặc điểm tương đối lớn mà số lượng thực thể lại nhỏ có thể dẫn đến không phù hợp dữ liệu



## Decision Tree

### ❑ Các bước thực hiện

- Tìm hiểu về dataset (Thu thập các số liệu quan sát, tiền xử lý, trực quan hóa dữ liệu)
- Tạo training data và test data
- Xây dựng mô hình (model) sử dụng rpart(formula, data = trainData, method = 'class')
- In kết quả của model
- Sử dụng mô hình để dự đoán cho test data dùng predict(mylogit, newdata = testData, type = "response")
- Tính toán độ chính xác của model
- Dự đoán kết quả cho các sample mới theo predict()



## Decision Tree

- Tìm hiểu về dataset (Thu thập các số liệu quan sát, tiền xử lý, trực quan hóa dữ liệu)

```
library(rpart)
print(summary(iris))
print(head(iris))
```

| Sepal.Length  | Sepal.Width   | Petal.Length  | Petal.Width   | Species       |
|---------------|---------------|---------------|---------------|---------------|
| Min. :4.300   | Min. :2.000   | Min. :1.000   | Min. :0.100   | setosa :50    |
| 1st Qu.:5.100 | 1st Qu.:2.800 | 1st Qu.:1.600 | 1st Qu.:0.300 | versicolor:50 |
| Median :5.800 | Median :3.000 | Median :4.350 | Median :1.300 | virginica :50 |
| Mean :5.843   | Mean :3.057   | Mean :3.758   | Mean :1.199   |               |
| 3rd Qu.:6.400 | 3rd Qu.:3.300 | 3rd Qu.:5.100 | 3rd Qu.:1.800 |               |
| Max. :7.900   | Max. :4.400   | Max. :6.900   | Max. :2.500   |               |
| Sepal.Length  | Sepal.Width   | Petal.Length  | Petal.Width   | Species       |
| 1             | 5.1           | 3.5           | 1.4           | 0.2 setosa    |
| 2             | 4.9           | 3.0           | 1.4           | 0.2 setosa    |
| 3             | 4.7           | 3.2           | 1.3           | 0.2 setosa    |
| 4             | 4.6           | 3.1           | 1.5           | 0.2 setosa    |
| 5             | 5.0           | 3.6           | 1.4           | 0.2 setosa    |
| 6             | 5.4           | 3.9           | 1.7           | 0.4 setosa    |



# Decision Tree



## • Tạo training data và test data

```
# Create the training and test data
set.seed(42)
trainingRowIndex <- sample(1:nrow(iris), 0.7 * nrow(iris))
print("Selected training row indexes:")
print(trainingRowIndex)
trainingData <- iris[trainingRowIndex, ] # training data
testData <- iris[-trainingRowIndex, ] # test data
print("Rows of training data and test data:")
print(nrow(trainingData))
print(nrow(testData))

[1] "Selected training row indexes:"
[1] 138 140 43 123 94 76 107 20 146 100 65 141 129 35 63 127 132 16 136 74 118 18 135 121 11 149 49 1
12 55 102
[31] 89 97 46 81 1 96 116 24 128 68 42 48 5 105 130 101 93 66 139 134 34 137 40 77 4 72 64 1
33 25 47
[61] 61 88 67 50 131 17 23 69 57 143 115 12 85 37 104 54 114 28 38 86 41 126 92 44 52 119 15
6 144 19
[91] 70 71 13 75 103 60 150 73 39 32 51 83 124 90 120
[1] "Rows of training data and test data:"
[1] 105
[1] 45
```

R programming language for Data Science

29

# Decision Tree



## • Classification

- Xây dựng mô hình (model) sử dụng rpart(formula, data = trainData, method = 'class')
- In kết quả model

```
# Build model
iris.tree <- rpart(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
                     data = trainingData, method="class")
print(iris.tree)

n= 105

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

1) root 105 68 virginica (0.31428571 0.33333333 0.35238095)
  2) Petal.Width< 0.8 33 0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Width>=0.8 72 35 virginica (0.00000000 0.48611111 0.51388889) *
  6) Petal.Width< 1.65 38 4 versicolor (0.00000000 0.89473684 0.10526316) *
  7) Petal.Width>=1.65 34 1 virginica (0.00000000 0.02941176 0.97058824) *
```

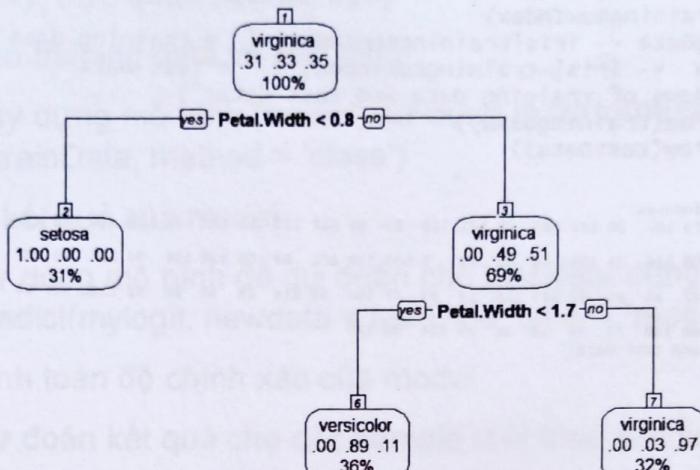
R programming language for Data Science

30

## Decision Tree

- Trực quan hóa kết quả

```
# draw tree
library(rpart.plot)
prp(iris.tree,type=2,extra="auto",nn = TRUE,branch=1,varlen=0,yesno=2)
```



R programming language for Data Science

31

## Decision Tree

- Dự đoán kết quả cho test data
- Tính toán độ chính xác của model

```
#test model
pred_new = predict(iris.tree, testData, type = "class")

# accuracy
accuracy <- table(pred_new, testData$Species)
accuracy = sum(diag(accuracy))/sum(accuracy)
print(paste("Accuracy s1:", accuracy))
```

[1] "Accuracy s1: 0.977777777777778"

# Decision Tree

- Dự đoán kết quả cho các sample mới theo predict()

```
# prediction new values
newCase <- iris[c(1,10, 100, 140),]
newCase$Species <- NULL
print(newCase)

print("New predictions:")
pred_new = predict(iris.tree, newCase, type = "class")
print(pred_new)

  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1       3.5        1.4       0.2
10         4.9       3.1        1.5       0.1
100        5.7       2.8        4.1       1.3
140        6.9       3.1        5.4       2.1
[1] "New predictions:"
     1      10     100     140
setosa setosa versicolor virginica
Levels: setosa versicolor virginica
```



# Decision Tree

## • Regression

- Xây dựng mô hình (model) sử dụng rpart(formula, data = trainData)
- In kết quả model

```
# fit model
tree.gre <- rpart(Petal.Length ~ Petal.Width, data=training_data)

tree.gre

n= 105

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

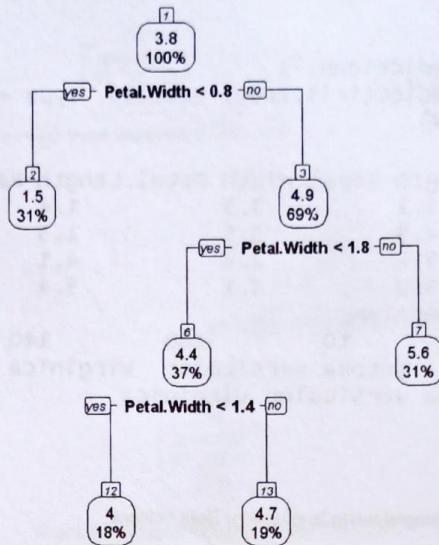
 1) root 105 319.5585000 3.829524
 2) Petal.Width< 0.8 33  0.9024242 1.451515 *
 3) Petal.Width>=0.8 72  46.5127800 4.919444
 6) Petal.Width< 1.75 39  10.0466700 4.366667 *
 12) Petal.Width< 1.35 19  1.8094740 4.005263 *
 13) Petal.Width>=1.35 20  3.3980000 4.710000 *
 7) Petal.Width>=1.75 33  10.4654500 5.572727 *
```



## Decision Tree

- Trực quan hóa kết quả

```
prp(tree.gre, type = 3, extra = "auto", nn = TRUE, branch = 1,
varlen=0, yesno = 2)
```



35

## Decision Tree

- Dự đoán kết quả cho test data

```
newdf <- data.frame(testing_data['Petal.Width'])
```

```
# make predictions
predictions <- predict(tree.gre, newdf)
```

```
# summarize accuracy
mse <- mean((testing_data$Petal.Length - predictions)^2)
print(mse)
```

```
[1] 0.160362
```



# Decision Tree



- Dự đoán kết quả cho các sample mới theo predict()

```
now <- data.frame(Petal.Width = c(0.25,1.25,2.25))
```

```
predict_new = predict(tree.gre, now)  
predict_new
```

```
1 1.45151515151515  
2 4.00526315789474  
3 5.57272727272727
```



# Chapter 19: Decision Tree

## Exercise 1: Classification Animal

Cho dữ liệu zoo.data.txt.

Thông tin các cột dữ liệu: Data Information: Bộ dữ liệu chứa 17 thuộc tính kiểu Boolean.  
Thuộc tính "type" là class attribute:

Class# – Set of animals: =====

- =====
1. (41) aardvark, antelope, bear, boar, buffalo, calf, cavy, cheetah, deer, dolphin, elephant, fruitbat, giraffe, girl, goat, gorilla, hamster, hare, leopard, lion, lynx, mink, mole, mongoose, opossum, oryx, platypus, polecat, pony, porpoise, puma, pussycat, raccoon, reindeer, seal, sealion, squirrel, vampire, vole, wallaby, wolf
  2. (20) chicken, crow, dove, duck, flamingo, gull, hawk, kiwi, lark, ostrich, parakeet, penguin, pheasant, rhea, skimmer, skua, sparrow, swan, vulture, wren
  3. (5) pitviper, seasnake, slowworm, tortoise, tuatara
  4. (13) bass, carp, catfish, chub, dogfish, haddock, herring, pike, piranha, seahorse, sole, stingray, tuna
  5. (4) frog, frog, newt, toad
  6. (8) flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp
  7. (10) clam, crab, crayfish, lobster, octopus, scorpion, seawasp, slug, starfish, worm

Thuộc tính:

1. animal name: Unique for each instance
2. hair: Boolean
3. feathers: Boolean
4. eggs: Boolean
5. milk: Boolean
6. airborne: Boolean
7. aquatic: Boolean
8. predator: Boolean
9. toothed: Boolean
10. backbone: Boolean
11. breathes: Boolean
12. venomous: Boolean
13. fins: Boolean
14. legs: Numeric (set of values: {0,2,4,5,6,8})
15. tail: Boolean
16. domestic: Boolean
17. catsize: Boolean
18. type: Numeric (integer values in range [1,7])

**Yêu cầu: Hãy áp dụng Decision Tree để dự đoán loại của animal dựa trên các thông tin được cung cấp:**

- Đọc dữ liệu và gán cho biến data.
- In thông tin head, tail, str, summary
- Chuẩn hóa dữ liệu nếu cần
- Tạo train:test từ dữ liệu data với tỉ lệ 75:25
- Áp dụng decision tree
- Tìm kết quả
- Tính toán độ chính xác
- Vẽ hình => xem kết quả
- Với các thông tin: c(1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 4, 0, 0, 1), c(1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 6, 0, 0, 0), thì mẫu này thuộc loại nào?

```
In [1]: library(rpart)
data <- read.csv("zoo.data.txt", header = FALSE)
print(head(data))
print(paste("Is dataframe?", is.data.frame(data)))
```

|   | V1       | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | V11 | V12 | V13 | V14 | V15 | V16 | V17 | V18 |
|---|----------|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1 | aardvark | 1  | 0  | 0  | 1  | 0  | 0  | 1  | 1  | 1   | 1   | 0   | 0   | 4   | 0   | 0   | 1   | 1   |
| 2 | antelope | 1  | 0  | 0  | 1  | 0  | 0  | 0  | 1  | 1   | 1   | 0   | 0   | 4   | 1   | 0   | 1   | 1   |
| 3 | bass     | 0  | 0  | 1  | 0  | 0  | 1  | 1  | 1  | 1   | 0   | 0   | 1   | 0   | 1   | 0   | 0   | 4   |
| 4 | bear     | 1  | 0  | 0  | 1  | 0  | 0  | 1  | 1  | 1   | 1   | 0   | 0   | 4   | 0   | 0   | 1   | 1   |
| 5 | boar     | 1  | 0  | 0  | 1  | 0  | 0  | 1  | 1  | 1   | 1   | 0   | 0   | 4   | 1   | 0   | 1   | 1   |
| 6 | buffalo  | 1  | 0  | 0  | 1  | 0  | 0  | 0  | 1  | 1   | 1   | 0   | 0   | 4   | 1   | 0   | 1   | 1   |

[1] "Is dataframe? TRUE"

```
In [2]: # tail(data)
```

```
In [3]: print(paste("cols:", ncol(data)))
print(paste("rows:", nrow(data)))

[1] "cols: 18"
[1] "rows: 101"
```

In [4]: `summary(data)`

|          | V1      | V2             | V3             | V4              |
|----------|---------|----------------|----------------|-----------------|
| frog     | : 2     | Min. :0.0000   | Min. :0.000    | Min. :0.0000    |
| aardvark | : 1     | 1st Qu.:0.0000 | 1st Qu.:0.000  | 1st Qu.:0.0000  |
| antelope | : 1     | Median :0.0000 | Median :0.000  | Median :1.0000  |
| bass     | : 1     | Mean :0.4257   | Mean :0.198    | Mean :0.5842    |
| bear     | : 1     | 3rd Qu.:1.0000 | 3rd Qu.:0.000  | 3rd Qu.:1.0000  |
| boar     | : 1     | Max. :1.0000   | Max. :1.000    | Max. :1.0000    |
| (Other)  | : 94    |                |                |                 |
|          | V5      | V6             | V7             | V8              |
| Min.     | :0.0000 | Min. :0.0000   | Min. :0.0000   | Min. :0.0000    |
| 1st Qu.  | :0.0000 | 1st Qu.:0.0000 | 1st Qu.:0.0000 | 1st Qu.:0.0000  |
| Median   | :0.0000 | Median :0.0000 | Median :0.0000 | Median :1.0000  |
| Mean     | :0.4059 | Mean :0.2376   | Mean :0.3564   | Mean :0.5545    |
| 3rd Qu.  | :1.0000 | 3rd Qu.:0.0000 | 3rd Qu.:1.0000 | 3rd Qu.:1.0000  |
| Max.     | :1.0000 | Max. :1.0000   | Max. :1.0000   | Max. :1.0000    |
|          | V9      | V10            | V11            | V12             |
| Min.     | :0.000  | Min. :0.0000   | Min. :0.0000   | Min. :0.00000   |
| 1st Qu.  | :0.000  | 1st Qu.:1.0000 | 1st Qu.:1.0000 | 1st Qu.:0.00000 |
| Median   | :1.000  | Median :1.0000 | Median :1.0000 | Median :0.00000 |
| Mean     | :0.604  | Mean :0.8218   | Mean :0.7921   | Mean :0.07921   |
| 3rd Qu.  | :1.000  | 3rd Qu.:1.0000 | 3rd Qu.:1.0000 | 3rd Qu.:0.00000 |
| Max.     | :1.000  | Max. :1.0000   | Max. :1.0000   | Max. :1.00000   |
|          | V13     | V14            | V15            | V16             |
| Min.     | :0.0000 | Min. :0.000    | Min. :0.0000   | Min. :0.0000    |
| 1st Qu.  | :0.0000 | 1st Qu.:2.000  | 1st Qu.:0.0000 | 1st Qu.:0.0000  |
| Median   | :0.0000 | Median :4.000  | Median :1.0000 | Median :0.0000  |
| Mean     | :0.1683 | Mean :2.842    | Mean :0.7426   | Mean :0.1287    |
| 3rd Qu.  | :0.0000 | 3rd Qu.:4.000  | 3rd Qu.:1.0000 | 3rd Qu.:0.0000  |
| Max.     | :1.0000 | Max. :8.000    | Max. :1.0000   | Max. :1.0000    |
|          | V17     | V18            |                |                 |
| Min.     | :0.0000 | Min. :1.000    |                |                 |
| 1st Qu.  | :0.0000 | 1st Qu.:1.000  |                |                 |
| Median   | :0.0000 | Median :2.000  |                |                 |
| Mean     | :0.4356 | Mean :2.832    |                |                 |
| 3rd Qu.  | :1.0000 | 3rd Qu.:4.000  |                |                 |
| Max.     | :1.0000 | Max. :7.000    |                |                 |

In [5]: str(data)

```
'data.frame': 101 obs. of 18 variables:  
 $ V1 : Factor w/ 100 levels "aardvark","antelope",...: 1 2 3 4 5 6 7 8 9 10 ...  
 $ V2 : int 1 1 0 1 1 1 1 0 0 1 ...  
 $ V3 : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ V4 : int 0 0 1 0 0 0 0 1 1 0 ...  
 $ V5 : int 1 1 0 1 1 1 1 0 0 1 ...  
 $ V6 : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ V7 : int 0 0 1 0 0 0 0 1 1 0 ...  
 $ V8 : int 1 0 1 1 1 0 0 0 1 0 ...  
 $ V9 : int 1 1 1 1 1 1 1 1 1 1 ...  
 $ V10: int 1 1 1 1 1 1 1 1 1 1 ...  
 $ V11: int 1 1 0 1 1 1 1 0 0 1 ...  
 $ V12: int 0 0 0 0 0 0 0 0 0 0 ...  
 $ V13: int 0 0 1 0 0 0 0 1 1 0 ...  
 $ V14: int 4 4 0 4 4 4 4 0 0 4 ...  
 $ V15: int 0 1 1 0 1 1 1 1 1 0 ...  
 $ V16: int 0 0 0 0 0 0 1 1 0 1 ...  
 $ V17: int 1 1 0 1 1 1 1 0 0 0 ...  
 $ V18: int 1 1 4 1 1 1 1 4 4 1 ...
```

In [6]: # Column 18: type

```
data <- subset(data, select=-V1)  
print(head(data))
```

|   | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | V11 | V12 | V13 | V14 | V15 | V16 | V17 | V18 |   |
|---|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| 1 | 1  | 1  | 0  | 0  | 1  | 0  | 0  | 1  | 1   | 1   | 1   | 0   | 0   | 4   | 0   | 0   | 1   | 1 |
| 2 | 1  | 0  | 0  | 1  | 0  | 0  | 0  | 1  | 1   | 1   | 0   | 0   | 0   | 4   | 1   | 0   | 1   | 1 |
| 3 | 0  | 0  | 1  | 0  | 0  | 1  | 1  | 1  | 1   | 0   | 0   | 1   | 0   | 1   | 0   | 0   | 4   |   |
| 4 | 1  | 0  | 0  | 1  | 0  | 0  | 1  | 1  | 1   | 1   | 0   | 0   | 0   | 4   | 0   | 0   | 1   | 1 |
| 5 | 1  | 0  | 0  | 1  | 0  | 0  | 1  | 1  | 1   | 1   | 0   | 0   | 0   | 4   | 1   | 0   | 1   | 1 |
| 6 | 1  | 0  | 0  | 1  | 0  | 0  | 0  | 1  | 1   | 1   | 0   | 0   | 0   | 4   | 1   | 0   | 1   | 1 |



```
In [7]: # Create the training and test data
set.seed(42)
trainingRowIndex <- sample(1:nrow(data), 0.75*nrow(data))
print("Selected training row indexes:")
print(trainingRowIndex)
trainingData <- data[trainingRowIndex, ] # training data
testData <- data[-trainingRowIndex, ] # test data
print("Rows of training data and test data:")
print(nrow(trainingData))
print(nrow(testData))

[1] "Selected training row indexes:"
[1]  93  94  29  82  63  50  70  13  62  65  42  92  84  23  41  81  89  10  4
0
[20]  46  74  12  79  86   7  83  30  68  33  61  53  57  27  47   1  55  67  1
4
[39]  58  38  24  69   3  95  25  54  49  35  52  73  18  51  20 101   2  56  3
1
[58]   8  80  22  28  76  75  91  32  77  85  99  88  44  78   5  36  64   6
[1] "Rows of training data and test data:"
[1] 75
[1] 26
```

```
In [8]: # Build model
#use: control = list(maxdepth = 15)
data.tree <- rpart(V18 ~ V2+V3+V4+V5+V6+V7+V8+V9+V10+V11+V12+V13+V14+V15+V16+V17,
                    data = trainingData,
                    method="class",
                    minbucket=1)

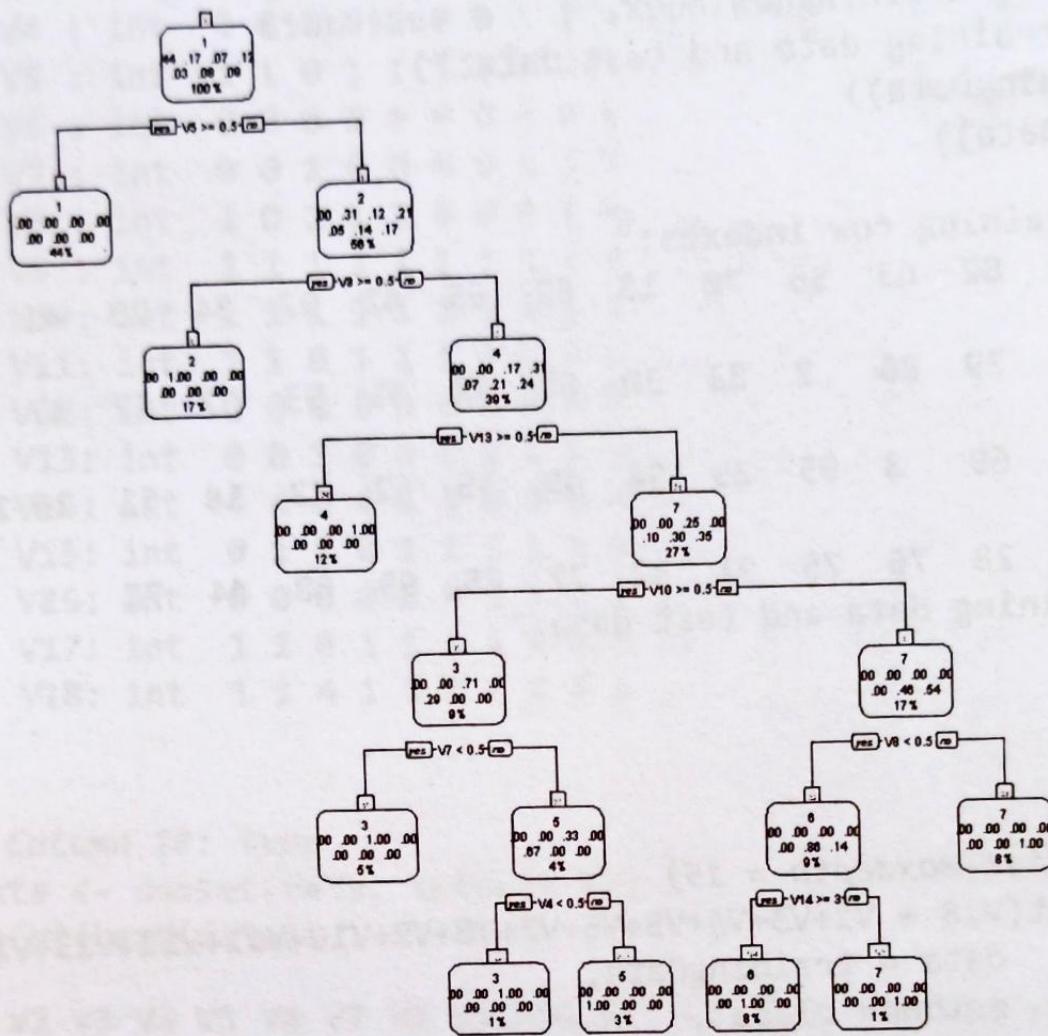
print(data.tree)

n= 75

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

  1) root 75 42 1 (0.44 0.17 0.067 0.12 0.027 0.08 0.093)
  2) V5>=0.5 33  0 1 (1 0 0 0 0 0) *
  3) V5< 0.5 42 29 2 (0 0.31 0.12 0.21 0.048 0.14 0.17)
     6) V3>=0.5 13  0 2 (0 1 0 0 0 0) *
     7) V3< 0.5 29 20 4 (0 0 0.17 0.31 0.069 0.21 0.24)
    14) V13>=0.5 9  0 4 (0 0 0 1 0 0 0) *
    15) V13< 0.5 20 13 7 (0 0 0.25 0 0.1 0.3 0.35)
     30) V10>=0.5 7  2 3 (0 0 0.71 0 0.29 0 0)
        60) V7< 0.5 4  0 3 (0 0 1 0 0 0 0) *
        61) V7>=0.5 3  1 5 (0 0 0.33 0 0.67 0 0)
          122) V4< 0.5 1  0 3 (0 0 1 0 0 0 0) *
          123) V4>=0.5 2  0 5 (0 0 0 0 1 0 0) *
     31) V10< 0.5 13  6 7 (0 0 0 0 0 0.46 0.54)
     62) V8< 0.5 7  1 6 (0 0 0 0 0 0.86 0.14)
        124) V14>=3 6  0 6 (0 0 0 0 0 1 0) *
        125) V14< 3 1  0 7 (0 0 0 0 0 0 1) *
     63) V8>=0.5 6  0 7 (0 0 0 0 0 0 1) *
```

```
In [9]: # draw tree
library(rpart.plot)
prp(data.tree,type=2,extra="auto",nn = TRUE,branch=1,varlen=0,yesno=2)
```



```
In [10]: #test model
pred_new = predict(data.tree, testData, type = "class")
```

```
In [11]: print("Predict vs Actual:")
result <- data.frame(Predict = pred_new, Actual = testData$V18)
print(result)

[1] "Predict vs Actual:"
   Predict Actual
 4        1      1
 9        4      4
11       1      1
15       7      7
16       7      7
17       2      2
19       4      4
21       2      2
26       5      5
34       2      2
37       1      1
39       4      4
43       7      6
45       1      1
48       1      1
59       2      2
60       2      2
66       1      1
71       1      1
72       2      2
87       4      4
90       5      5
96       2      2
97       1      1
98       6      6
100      7      7
```

```
In [12]: # SOLUTION 2
misClasificError <- mean(pred_new != testData$V18)
print(paste('Accuracy s2: ', 1-misClasificError))

[1] "Accuracy s2:  0.961538461538462"
```

```
In [13]: # prediction new values
newCase <- data[c(1,10, 100),]
print(newCase$V18)
newCase$V18 <- NULL
print(newCase)

[1] 1 1 7
   V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17
 1  1  0  0  1  0  0  1  1  1  1  0  0  4  0  0  0  1
10  1  0  0  1  0  0  0  1  1  1  0  0  4  0  1  0
100 0  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0  0
```

```
In [14]: print("New predictions:")
pred_new = predict(data.tree, newCase, type = "class")
print(pred_new)
```

```
[1] "New predictions:"
  1 10 100
  1   1   7
Levels: 1 2 3 4 5 6 7
```

```
In [15]: newdata = data.frame(V2 = c(1, 1),
                             V3 = c(0, 0),
                             V4 = c(0, 1),
                             V5 = c(1, 0),
                             V6 = c(0, 1),
                             V7 = c(0, 0),
                             V8 = c(1, 0),
                             V9 = c(1, 0),
                             V10 = c(1, 0),
                             V11 = c(0, 1),
                             V12 = c(0, 1),
                             V13 = c(0, 0),
                             V14 = c(4, 6),
                             V15 = c(0, 0),
                             V16 = c(0, 0),
                             V17 = c(1, 0))
```

```
print("New predictions:")
pred_new = predict(data.tree, newdata, type = "class")
print(pred_new)
```

```
[1] "New predictions:"
  1 2
  1 6
Levels: 1 2 3 4 5 6 7
```

# Chapter 19: Decision Tree

## Exercise 2: Baseball

**Yêu cầu: Áp dụng Decision Tree để dự đoán cân nặng dựa trên chiều cao**

- Cho dữ liệu baseball\_2D.txt
- Tái sử dụng bài Baseball ở phần Linear Regression, sau đó áp dụng Decision Tree để dự đoán cân nặng dựa trên chiều cao
- Cho chiều cao lần lượt là c(1.775, 1.825, 1.925), hỏi cân nặng bằng bao nhiêu?
- Chú ý: khi build model Decision Tree cho Regression phải sử dụng method là “anova”, khi predict thì chọn type là “vector”

```
In [1]: # predict weight based on height
# open and read csv file
library(rpart)
data <- read.csv("baseball.csv")
print(head(data))
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

|   | Name            | Team | Position       | Height | Weight | Age   | PosCategory |
|---|-----------------|------|----------------|--------|--------|-------|-------------|
| 1 | Adam_Donachie   | BAL  | Catcher        | 74     | 180    | 22.99 | Catcher     |
| 2 | Paul_Bako       | BAL  | Catcher        | 74     | 215    | 34.69 | Catcher     |
| 3 | Ramon_Hernandez | BAL  | Catcher        | 72     | 210    | 30.78 | Catcher     |
| 4 | Kevin_Millar    | BAL  | First_Baseman  | 72     | 210    | 35.43 | Infielder   |
| 5 | Chris_Gomez     | BAL  | First_Baseman  | 73     | 188    | 35.71 | Infielder   |
| 6 | Brian_Roberts   | BAL  | Second_Baseman | 69     | 176    | 29.39 | Infielder   |

```
[1] TRUE
[1] 7
[1] 1015
```

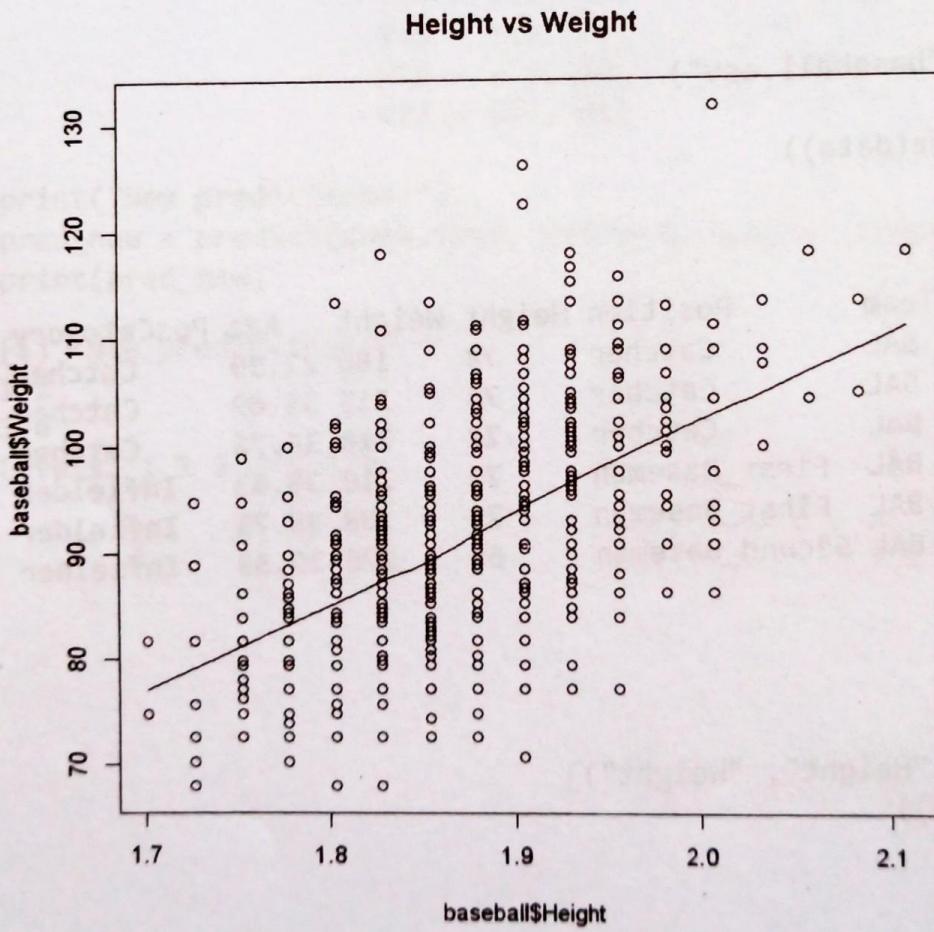
```
In [2]: baseball <- data[c("Height", "Weight")]
print(head(baseball))
```

|   | Height | Weight |
|---|--------|--------|
| 1 | 74     | 180    |
| 2 | 74     | 215    |
| 3 | 72     | 210    |
| 4 | 72     | 210    |
| 5 | 73     | 188    |
| 6 | 69     | 176    |

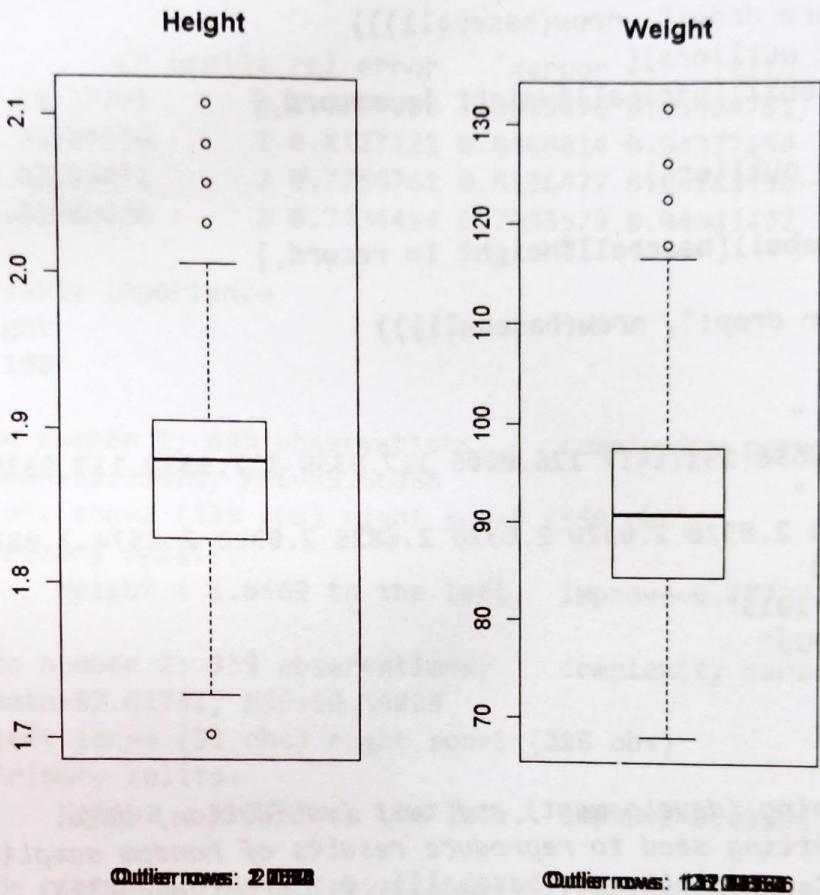
```
In [3]: baseball["Height"] <- baseball["Height"] * 0.0254  
baseball["Weight"] <- baseball["Weight"] * 0.453592  
  
print("After changing data:")  
print(head(baseball))
```

```
[1] "After changing data:"  
    Height   Weight  
1 1.8796 81.64656  
2 1.8796 97.52228  
3 1.8288 95.25432  
4 1.8288 95.25432  
5 1.8542 85.27530  
6 1.7526 79.83219
```

```
In [4]: scatter.smooth(x=baseball$Height,  
                      y=baseball$Weight,  
                      main="Height vs Weight")
```



```
In [5]: # BoxPlot to Check for outliers  
par(mfrow=c(1, 2)) # divide graph area in 2 columns  
boxplot(baseball$Height, main="Height",  
        sub=paste("Outlier rows: ",  
                  boxplot.stats(baseball$Height)$out))  
boxplot(baseball$Weight, main="Weight",  
        sub=paste("Outlier rows: ",  
                  boxplot.stats(baseball$Weight)$out))
```



```
In [6]: # calculate correlation between Width and Length  
print(cor(baseball$Height, baseball$Weight))
```

```
[1] 0.5315393
```



```
In [7]: wt_outliers <- c(boxplot.stats(baseball$Weight)$out)
print("wt_outliers: ")
print(wt_outliers)

ht_outliers <- c(boxplot.stats(baseball$Height)$out)
print("ht_outliers: ")
print(ht_outliers)

#drop rows have outliers
print(paste("Before drop:", nrow(baseball)))
for (record in wt_outliers){
  baseball <- baseball[baseball$Weight != record,]
}
for (record in ht_outliers)
{
  baseball <- baseball[baseball$Height != record,]
}
print(paste("After drop:", nrow(baseball)))

[1] "wt_outliers: "
[1] 117.9339 122.4698 131.5417 126.0986 117.9339 117.9339 117.9339
[1] "ht_outliers: "
[1] 2.0574 2.0320 2.0320 2.0320 2.0320 2.0828 2.0320 2.0574 2.0828 2.1082
[11] 1.7018 1.7018
[1] "Before drop: 1015"
[1] "After drop: 998"
```

```
In [8]: # Create the training (development) and test (validation) data.
set.seed(42) # setting seed to reproduce results of random sampling
trainingRowIndex <- sample(1:nrow(baseball), 0.7*nrow(baseball))
print("Selected training row indexes:")
# print(trainingRowIndex)
trainingData <- baseball[trainingRowIndex, ] # training data
testData <- baseball[-trainingRowIndex, ] # test data
print("Rows of training data and test data:")
print(nrow(trainingData))
print(nrow(testData))

[1] "Selected training row indexes:"
[1] "Rows of training data and test data:"
[1] 698
[1] 300
```

```
In [9]: # Use Decision Tree
# Build model
baseball.tree <- rpart(Weight~Height,
                        data = trainingData,
                        method="anova")
print(summary(baseball.tree))

Call:
rpart(formula = Weight ~ Height, data = trainingData, method = "anova")
n= 698

          CP nsplit rel error      xerror      xstd
1 0.18727791    0 1.0000000 1.0015896 0.04984781
2 0.03704596    1 0.8127221 0.8408814 0.04377180
3 0.03223072    2 0.7756761 0.8136477 0.04243592
4 0.01000000    3 0.7434454 0.7693579 0.04083492

Variable importance
Height
100

Node number 1: 698 observations,      complexity param=0.1872779
mean=91.09076, MSE=83.66086
left son=2 (339 obs) right son=3 (359 obs)
Primary splits:
Height < 1.8669 to the left,  improve=0.1872779, (0 missing)

Node number 2: 339 observations,      complexity param=0.03223072
mean=87.01741, MSE=60.56898
left son=4 (53 obs) right son=5 (286 obs)
Primary splits:
Height < 1.7907 to the left,  improve=0.09166379, (0 missing)

Node number 3: 359 observations,      complexity param=0.03704596
mean=94.93718, MSE=75.00348
left son=6 (230 obs) right son=7 (129 obs)
Primary splits:
Height < 1.9177 to the left,  improve=0.08034201, (0 missing)

Node number 4: 53 observations
mean=81.54386, MSE=39.88859

Node number 5: 286 observations
mean=88.03174, MSE=57.82052

Node number 6: 230 observations
mean=93.09877, MSE=65.82509

Node number 7: 129 observations
mean=98.21497, MSE=74.59822

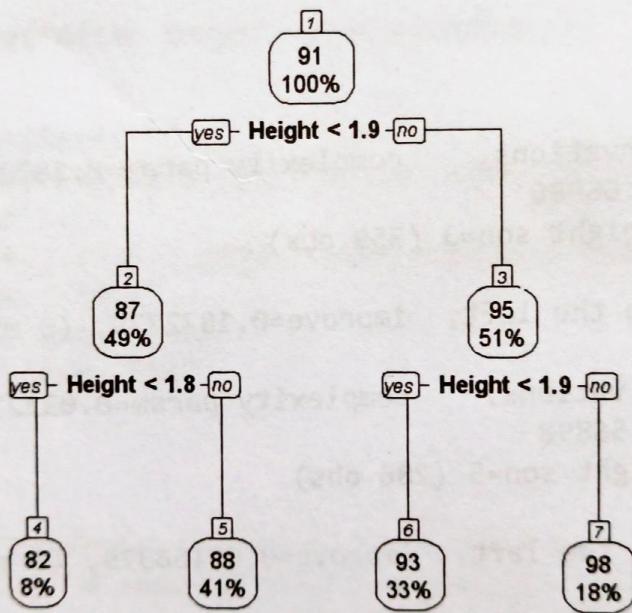
n= 698

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

1) root 698 58395.280 91.09076
```

- 2) Height < 1.8669 339 20532.890 87.01741  
 4) Height < 1.7907 53 2114.095 81.54386 \*  
 5) Height >= 1.7907 286 16536.670 88.03174 \*  
 3) Height >= 1.8669 359 26926.250 94.93718  
 6) Height < 1.9177 230 15139.770 93.09877 \*  
 7) Height >= 1.9177 129 9623.170 98.21497 \*

```
In [10]: # draw tree
library(rpart.plot)
prp(baseball.tree, type=2, extra="auto",
nn = TRUE, branch=1, varlen=0, yesno=2)
```



```
In [11]: #test model
pred_new = predict(baseball.tree, testData, type = "vector")

# mean square error of testData
mse_test = mean((testData$Weight - pred_new)^2)
print(paste("mse in test: ", mse_test))

[1] "mse in test: 59.6827254267076"
```

```
In [12]: # new predictions
x <- c(1.775, 1.825, 1.925)
y1 <- predict(baseball.tree, data.frame(Height = x), type = "vector")
print("Solution 2 - results:")
print(y1)

[1] "Solution 2 - results:"
     1      2      3
81.54386 88.03174 98.21497
```