Tree-Based Methods

國立陽明交通大學
生醫光電所 吳育德

A Concise Introduction to Machine Learning, 2020 Anita C. Faul, Chapter 5
Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, An Introduction to Statistical Learning, Chapter 8

Tree-Based Methods

- We describe **tree-based** methods for regression and classification.
- These involve **stratifying** or **segmenting** the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as **decision tree** methods.

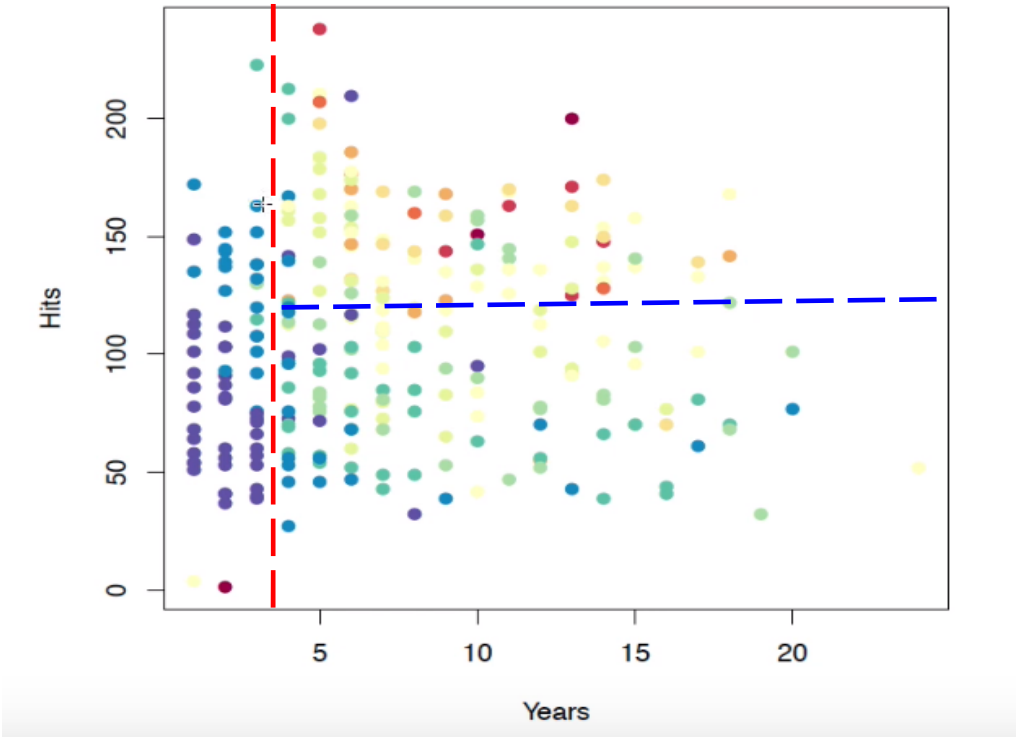
Pros and Cons

- Tree-based methods are simple and useful for interpretation.
- However, they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss **bagging**, **random forests**, and **boosting**.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss in interpretation.

How would you stratify it?

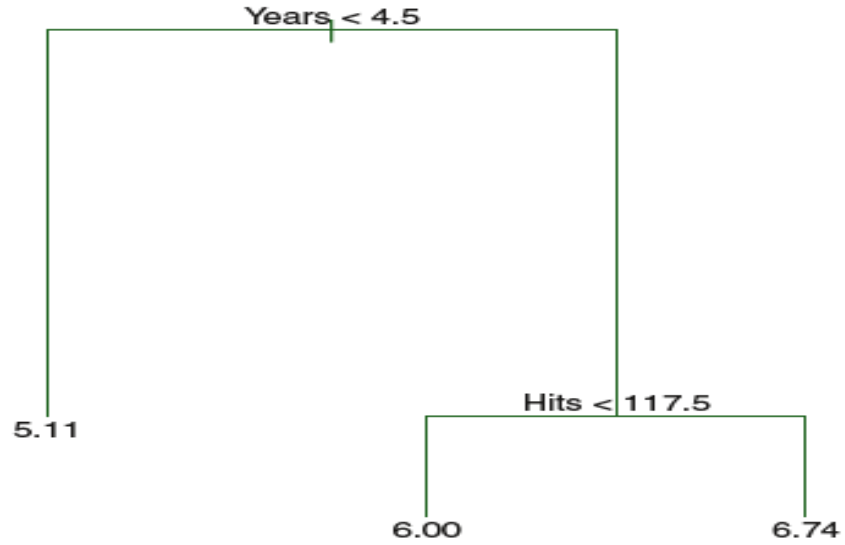
- **Baseball salary data :**

  Salary is color-coded from low (blue, green) to high (yellow, red).



The Basics of Decision Trees

- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. (**Hitters data**)



Regression Trees

Decision Tree for these data

- A regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.
- The split at the top of the tree results in two large branches. The left-hand branch corresponds to **Years < 4.5**, and the right-hand branch corresponds to **Years >= 4.5**.
- The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.

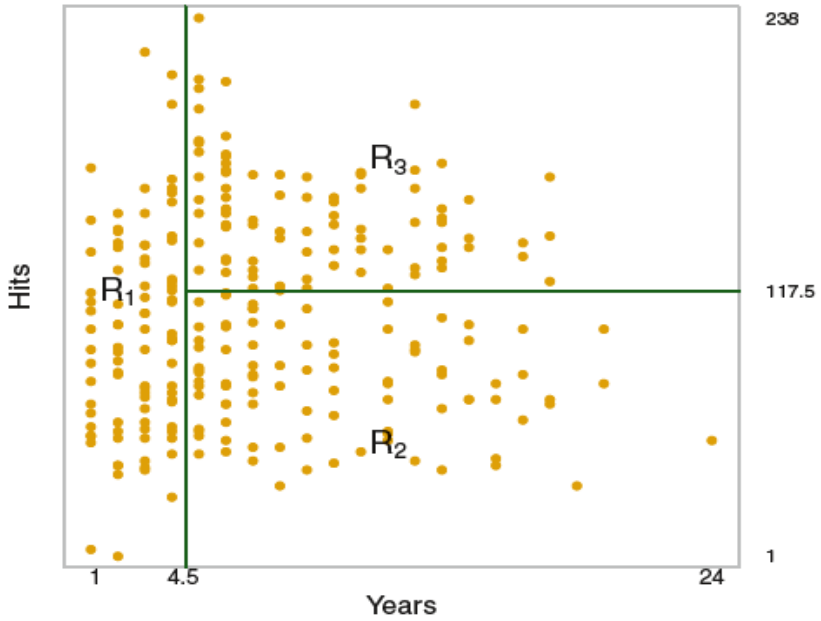
Predicting Baseball Players' Salaries Using Regression Trees

- We use the **Hitters data** set to predict a baseball player's Salary based on **Years** and **Hits**.
- These three regions can be written as

$R_1 = \{X | Years < 4.5\}$,

$R_2 = \{X | Years \geq 4.5, Hits < 117.5\}$,

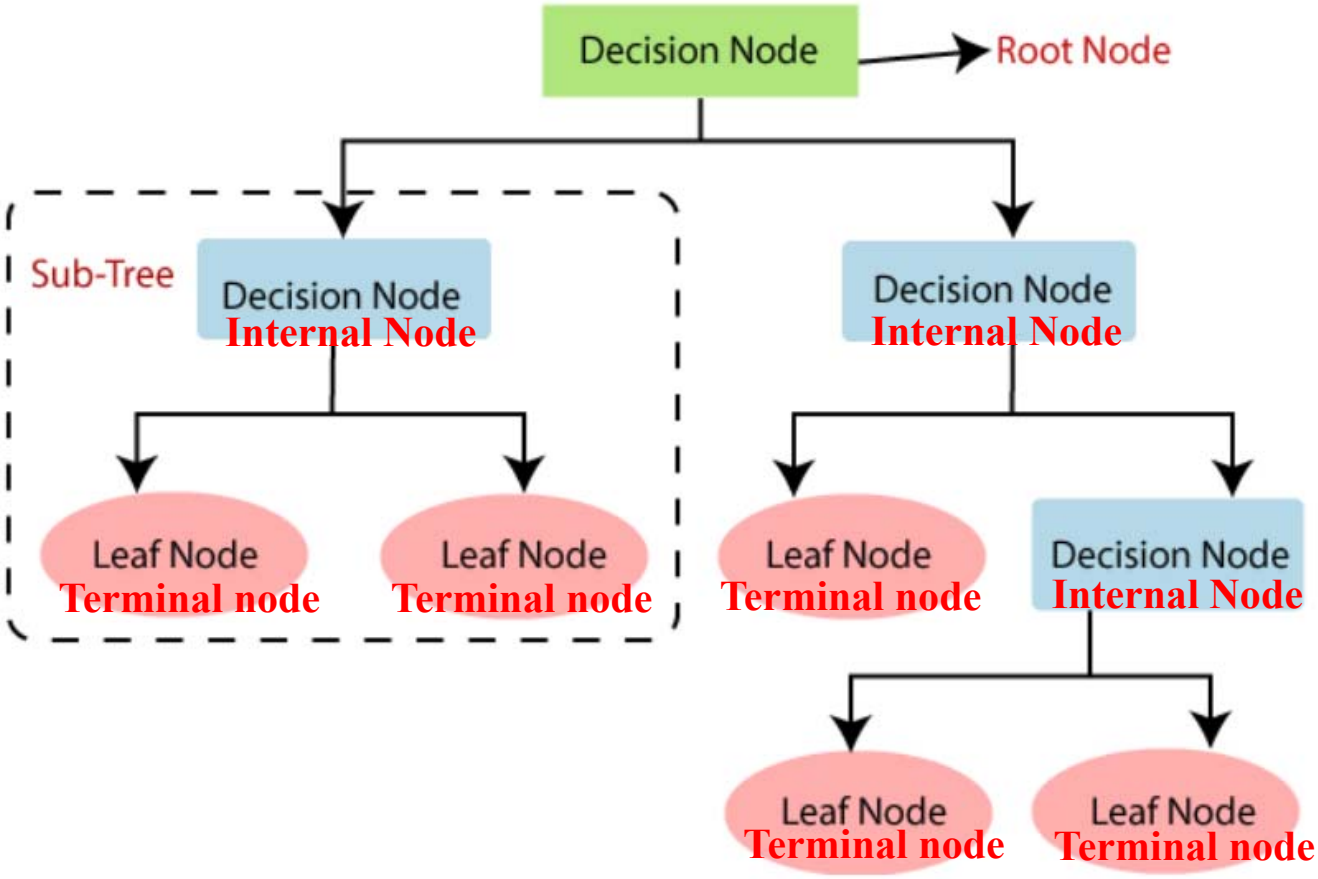
$R_3 = \{X | Years \geq 4.5, Hits \geq 117.5\}$.



Terminology for Trees

- The regions R_1, R_2, and R_3 are known as **leaf (terminal) nodes**
- Decision trees are typically drawn **upside down**, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as **decision (internal) nodes**
- In the hitters tree, the two **decision nodes** are indicated by the text **Years** <4.5 and **Hits** < 117.5.

Decision Tree



https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

Details of the tree-building process

1. We divide the predictor space—that is, the set of possible values for $X_1, X_2, \ldots, X_p$—into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j .

For instance, suppose in Step 1 we obtain two regions, R_1 and R_2, and that the response means of the training observations in R_1 and R_2 are 10 and 20. Then for a given observation $X = x$, if $x \in R_1$ we will predict 10, and if $x \in R_2$ we will predict 20.

Details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into **high-dimensional rectangles**, or **boxes**, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the residual sum of squares (RSS)

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

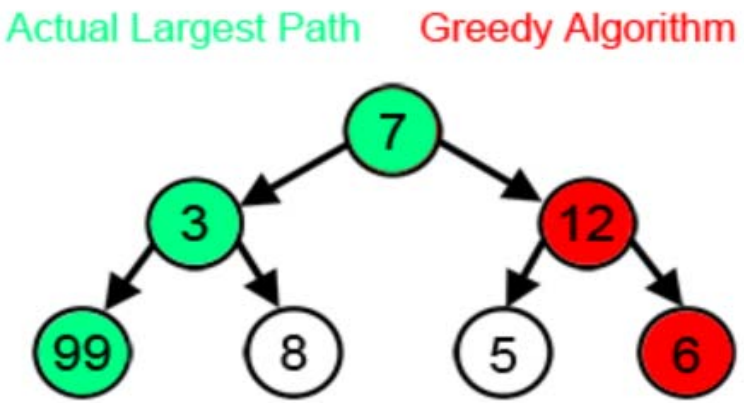
where y_i is the training observations and $\hat{y}_{R_j}$ is the mean response (prediction) for the training observations within the jth box.

Details of the tree-building process

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a **top-down**, **greedy** approach that is known as **recursive binary splitting**.
- The approach is **top-down** because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is **greedy** because at each step of the tree-building process, the **best** split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Greedy Algorithm

- A greedy algorithm is an intuitive algorithm that is used in optimization problems. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem.
- However, in many problems, a greedy strategy does not produce an optimal solution. For example, the greedy algorithm seeks to find the path with the largest sum. The greedy algorithm fails to find the largest sum, because it makes decisions based only on the information it has at any one step, without regard to the overall problem.



Details of the tree-building process

- We first select the predictor X_j and the cutpoint such that splitting the predictor space into the regions $R_1(j,s) = \{X | X_j < s\}$ and $R_2(j,s) = \{X | X_j \geq s\}$ and j and s that minimize the RSS

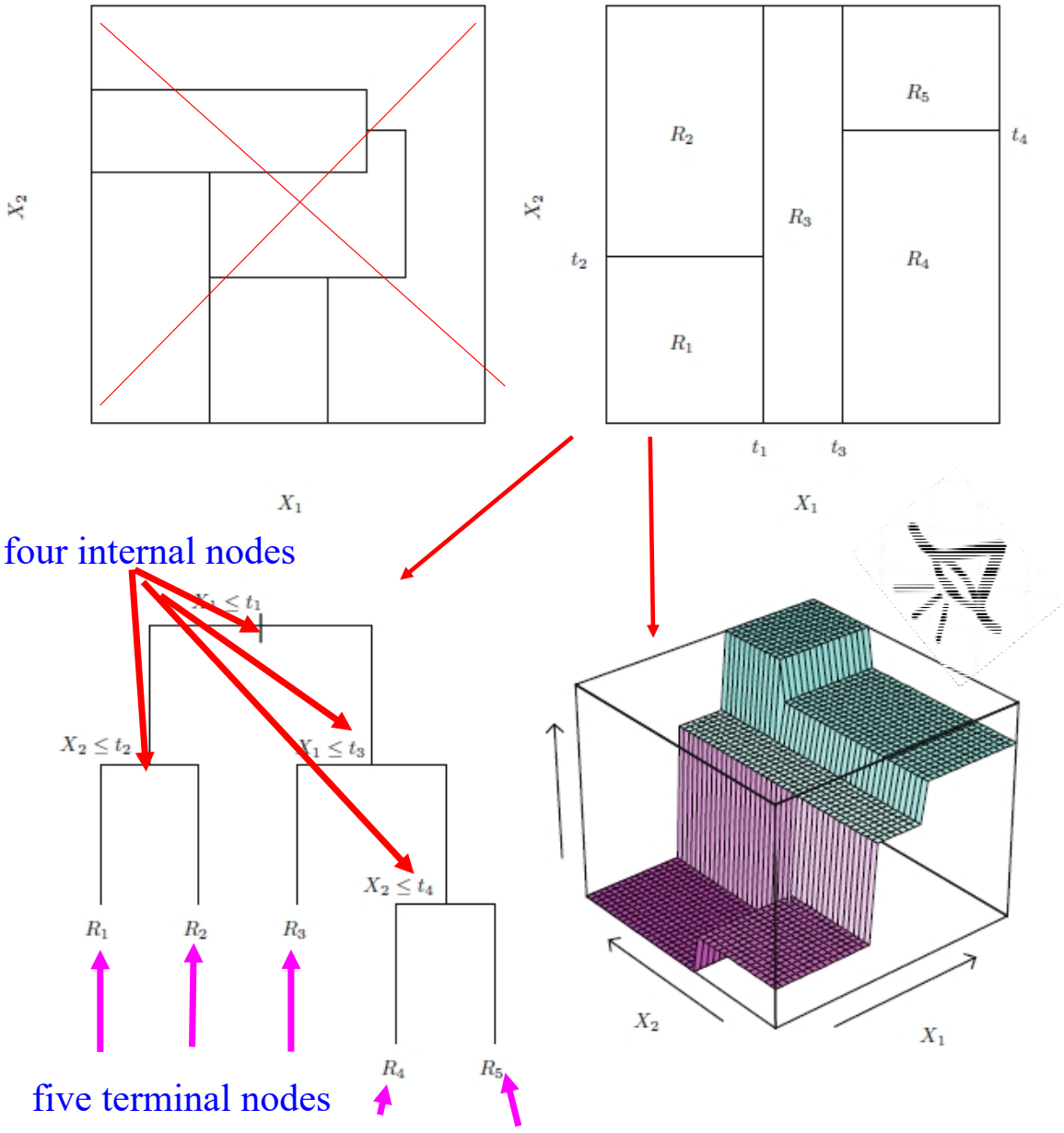
$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

where $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean responses for training observations in $R_1(j,s)$ and $R_2(j,s)$

- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- A five-region example of this approach is shown in next page.



Top Left: A partition of 2D feature space that could not result from recursive binary splitting.

Top Right: The output of recursive binary splitting on a 2D example.

Bottom Left: A tree corresponding to the partition in the top right panel.

Bottom Right: A perspective plot of the prediction surface corresponding to that tree.

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to **overfit** the data, leading to poor test set performance. **WHY?**
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to build the tree as the decrease in RSS due to each split exceeds some threshold.
- This strategy will result in smaller trees, but is too **short-sighted** since a seemingly worthless split early on in the tree might be followed by a very good split—that is, a split that leads to a large reduction in RSS later on.

Pruning a tree

- A better strategy is to grow a very large tree T_0, and then **prune** it back in order to obtain a **subtree**.
- **Cost complexity pruning**— also known as **weakest link pruning**— is used to do this.
- We consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

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

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: tree algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each k = 1, . . .,K:

   3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the Kth fold.

   3.2 Evaluate the mean squared prediction error on the data in the left-out Kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

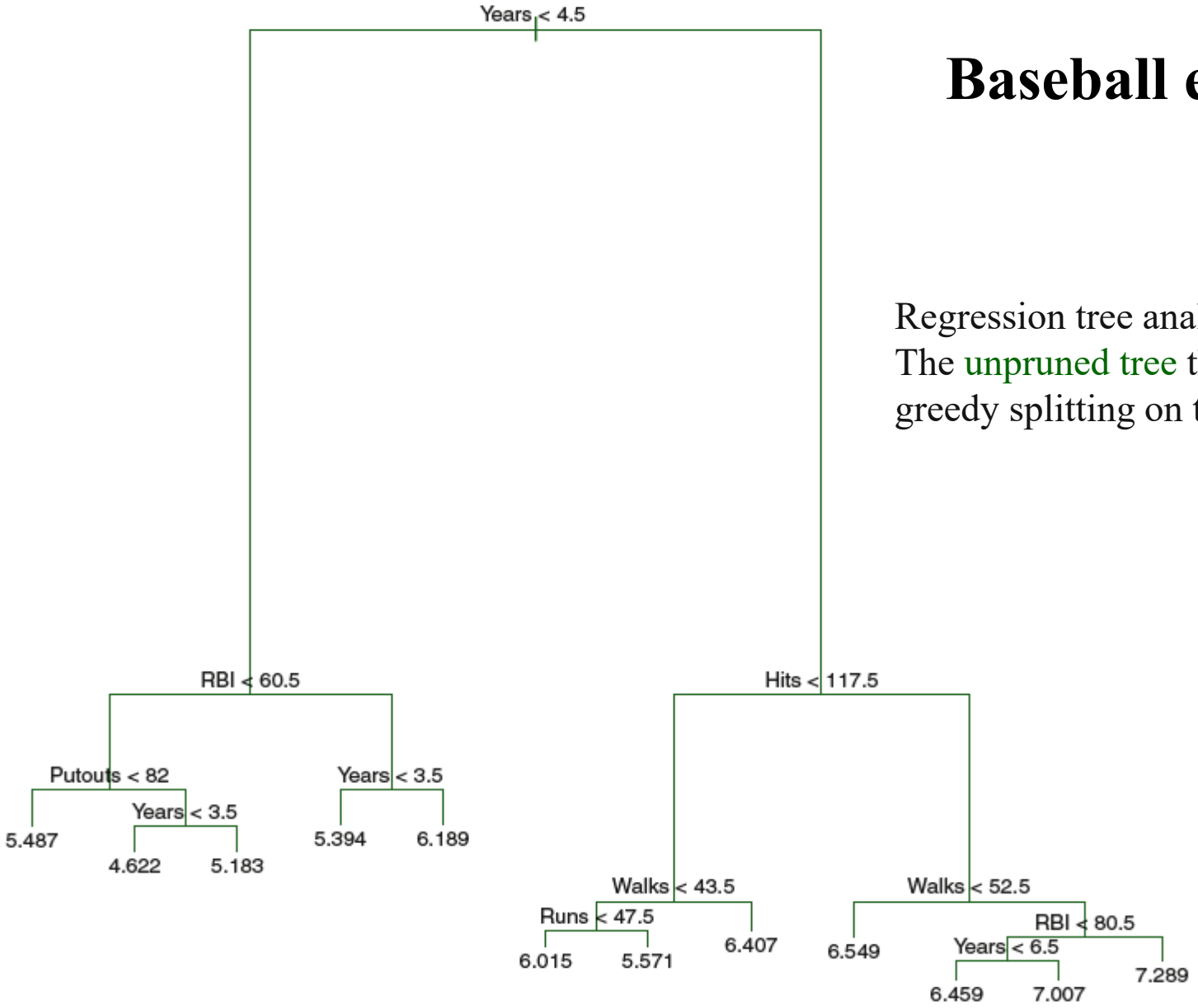
Baseball example continued

- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.

- We then built a large regression tree on the training data and varied α to create subtrees with different numbers of terminal nodes.

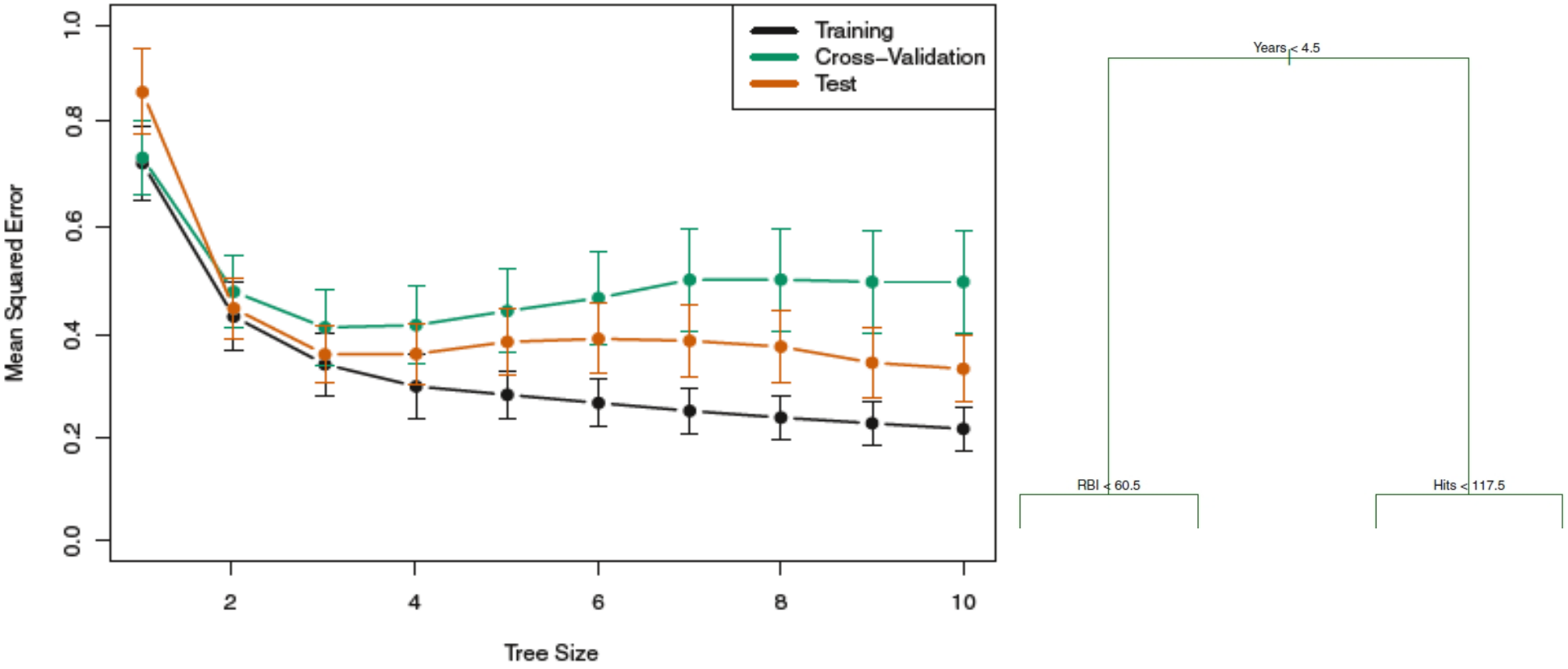
- Finally, we performed six-fold cross-validation to estimate the cross-validated MSE of the trees as a function of α.

Baseball example continued



Regression tree analysis for the Hitters data. The unpruned tree that results from top-down greedy splitting on the training data is shown.

Baseball example continued



Regression tree analysis for the Hitters data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.

Classification Trees

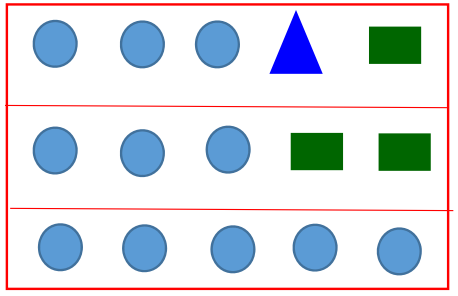
Classification Trees

- Predict that each observation belongs to **the most commonly occurring class** of training observations in the region.
- Use the recursive binary splitting to grow a classification tree.
- **Gini index**

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^{K} \hat{p}_{mk}^{\,2},$$

a measure of total variance (impurity) across K classes. $\hat{p}_{mk}$ represents the proportion of training observations in the mth region from the kth class

- Gini index is a measure of node **purity**—a small value indicates that a node contains predominantly observations from a single class.



$G = \frac{3}{5}\frac{2}{5} + \frac{1}{5}\frac{4}{5} + \frac{1}{5}\frac{4}{5} = \frac{14}{25}$

$G = \frac{3}{5}\frac{2}{5} + \frac{2}{5}\frac{3}{5} = \frac{12}{25}$ less variant !

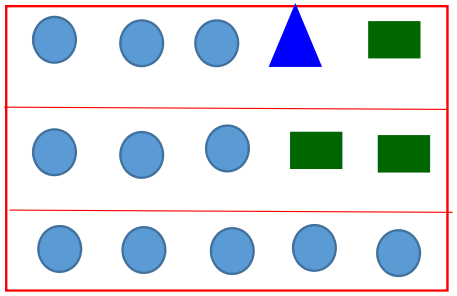
$G = \frac{5}{5}\frac{0}{5} = 0$ pure !

Entropy

- An alternative to the Gini index is ***entropy***, given by

$$H = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

- It turns out that the Gini index and the cross-entropy are quite similar numerically.



$$\text{H} = -\frac{3}{5} log_2\left(\frac{3}{5}\right) - \frac{1}{5} log_2\left(\frac{1}{5}\right) - \frac{1}{5} log_2\left(\frac{1}{5}\right) = 1.371$$

$$\text{H} = -\frac{3}{5} log_2\left(\frac{3}{5}\right) - \frac{2}{5} log_2\left(\frac{2}{5}\right) = 0.971$$ less variant !

$$\text{H} = -\frac{5}{5} log_2\left(\frac{5}{5}\right) = 0$$ pure !

Select the feature producing the highest Information gain

1. Compute entropy for a dataset with respect to a target feature

$$H(t, D) = -\sum_{l \in levels(t)} (P(t = l) \times log_2(P(t = l)))$$

where *levels*(t) is the set of levels of the target feature t, and $P(t = l)$ is the probability of a randomly selected instance having the target feature level l.

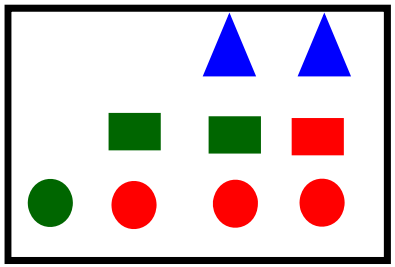
2. Use a particular feature d to create partitions $D_{d=l_1}, \cdots, D_{d=l_k}$, where $l_1, \cdots, l_k$ are the k levels that feature d can take. Each partition, $D_{d=l_i}$, contains the instances in that have a value of level l_i for the d feature. Compute the entropy remaining after partition

$$\text{Rem}(t, D) = \sum_{l \in levels(t)} \underbrace{\frac{|D_{d=l}|}{|D|}}_{\text{weighting}} \times \underbrace{H(t, D_{d=l})}_{\substack{entropy\ of \\ Partition\ D_{d=l}}}$$

3. Information gain made from splitting the dataset using the feature d

$$IG(t, D) = H(t, D) - \text{Rem}(t, D)$$

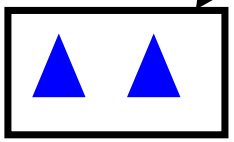
Choose a color feature to classify the shapes



$$H(shape: circle, square, triangle)$$
$$= -\frac{4}{9}log_2\left(\frac{4}{9}\right) - \frac{3}{9}log_2\left(\frac{3}{9}\right) - \frac{2}{9}log_2\left(\frac{2}{9}\right)$$
$$= 1.5305$$

Blue ?

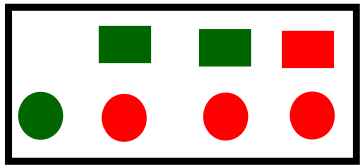
True



$$H(triangle)$$
$$= -\frac{2}{2}log_2\left(\frac{2}{2}\right)$$
$$= 0$$

Partition entropy

False

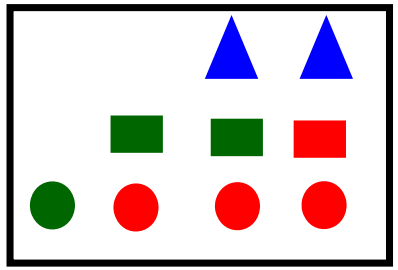


$$H(circle, square)$$
$$= -\frac{4}{7}log_2\left(\frac{4}{7}\right) - \frac{3}{7}log_2\left(\frac{3}{7}\right)$$
$$= 0.9852$$

Partition entropy

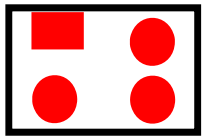
Entropy remaining: Rem $= \frac{2}{9} \times 0 + \frac{7}{9} \times 0.9852 = 0.7663$

Information gain: IG $= 1.5305 - \frac{7}{9} \times 0.9852 = 0.7642$



Red ?

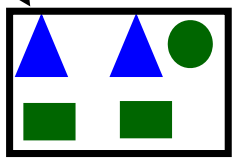
True



$$H(circle, square)$$
$$= -\frac{3}{4}log_2\left(\frac{3}{4}\right) - \frac{1}{4}log_2\left(\frac{1}{4}\right)$$
$$= 0.8113$$

Partition entropy

False



$$H(circle, square, triangle)$$
$$= -\frac{1}{5}log_2\left(\frac{1}{5}\right) - \frac{2}{5}log_2\left(\frac{2}{5}\right)$$
$$-\frac{2}{5}log_2\left(\frac{2}{5}\right) = 1.5219$$

Partition entropy

Rem $= \frac{4}{9} \times 0.8113 + \frac{5}{9} \times 1.5219 = 1.2061$

IG $= 1.5305 - 1.2061 = 0.3244$

Example 1

A convicted criminal who reoffends after release is known as a recidivist. The dataset describes prisoners released on parole, and whether they reoffended within two years of release.

ID	GOOD BEHAVIOR	AGE < 30	DRUG DEPENDENT	RECIDIVIST
1	false	true	false	true
2	false	false	false	false
3	false	true	false	true
4	true	false	false	false
5	true	false	true	true
6	true	false	false	false

Chap4. Exercise 2. John D. Kelleher, Brian Mac Namee, Aoife D'Arcy, **FUNDAMENTALS OF MACHINE LEARNING FOR PREDICTIVE DATA ANALYTICS, 2015**

- The first step: figure out which of the three features is the best one on which to split the dataset at the root node (i.e., which descriptive feature has the highest information gain).

- The total entropy for this dataset is

$$
\begin{aligned}
& H(\text{RECIDIVIST}, \mathcal{D}) \\
& = - \sum_{l \in \{true, false\}} P(\text{RECIDIVIST} = l) \times log_2 \left(P(\text{RECIDIVIST} = l)\right) \\
& = - \left(\left({}^3/_6 \times log_2({}^3/_6) \right) + \left({}^3/_6 \times log_2({}^3/_6) \right) \right) = 1.00\ bit
\end{aligned}
$$

Chap4. Exercise 2. John D. Kelleher, Brian Mac Namee, Aoife D'Arcy, **FUNDAMENTALS OF MACHINE LEARNING FOR PREDICTIVE DATA ANALYTICS, 2015**

The table below illustrates the information gain for features:

Split by Feature	Level	Part.	Instances	Partition Entropy	Rem.	Info. Gain
GOOD BEHAVIOR	*true*	$\mathcal{D}_1$	$\mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6$	0.9183	0.9183	0.0817
	false	$\mathcal{D}_2$	$\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3$	0.9183		
AGE < 30	*true*	$\mathcal{D}_3$	$\mathbf{d}_1, \mathbf{d}_3$	0	0.5409	0.4591
	false	$\mathcal{D}_4$	$\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6$	0.8113		
DRUG DEPENDENT	*true*	$\mathcal{D}_5$	$\mathbf{d}_5$	0	0.8091	0.1909
	false	$\mathcal{D}_6$	$\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_6$	0.9709		

$-1/3*log_2\left(\frac{1}{3}\right) - 2/3*log_2\left(\frac{2}{3}\right) = 0.9183$

=1.00-0.9183

=.9183*3/6+.9183*3/6

$-3/4*log_2\left(\frac{3}{4}\right) - 1/4*log_2\left(\frac{1}{4}\right) = 0.8113$

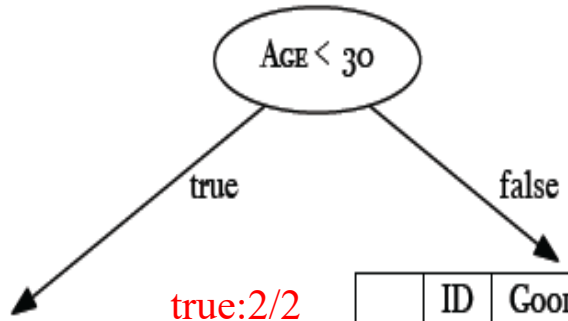
=1.00-0.5409

=0*2/6+.8113*4/6

$-3/5*log_2\left(\frac{3}{5}\right) - 2/5*log_2\left(\frac{2}{5}\right) = 0.9709$

=1.00-0.8091

=0*1/6+.9709*5/6



$H(\text{Recidivist}) = -\frac{2}{2}log_2\left(\frac{2}{2}\right) = 0$

Partition Entropy

$H(\text{Recidivist}) = -\frac{1}{4}log_2\left(\frac{1}{4}\right) - \frac{3}{4}log_2\left(\frac{3}{4}\right) = 0.81130$

Partition Entropy

true:2/2

	ID	GOOD BEHAVIOR	DRUG DEPENDENT	RECIDIVIST
D3	1	false	false	true
	3	false	false	true

No further split

	ID	GOOD BEHAVIOR	DRUG DEPENDENT	RECIDIVIST
D4	2	false	false	false
	4	true	false	false
	5	true	true	true
	6	true	false	false

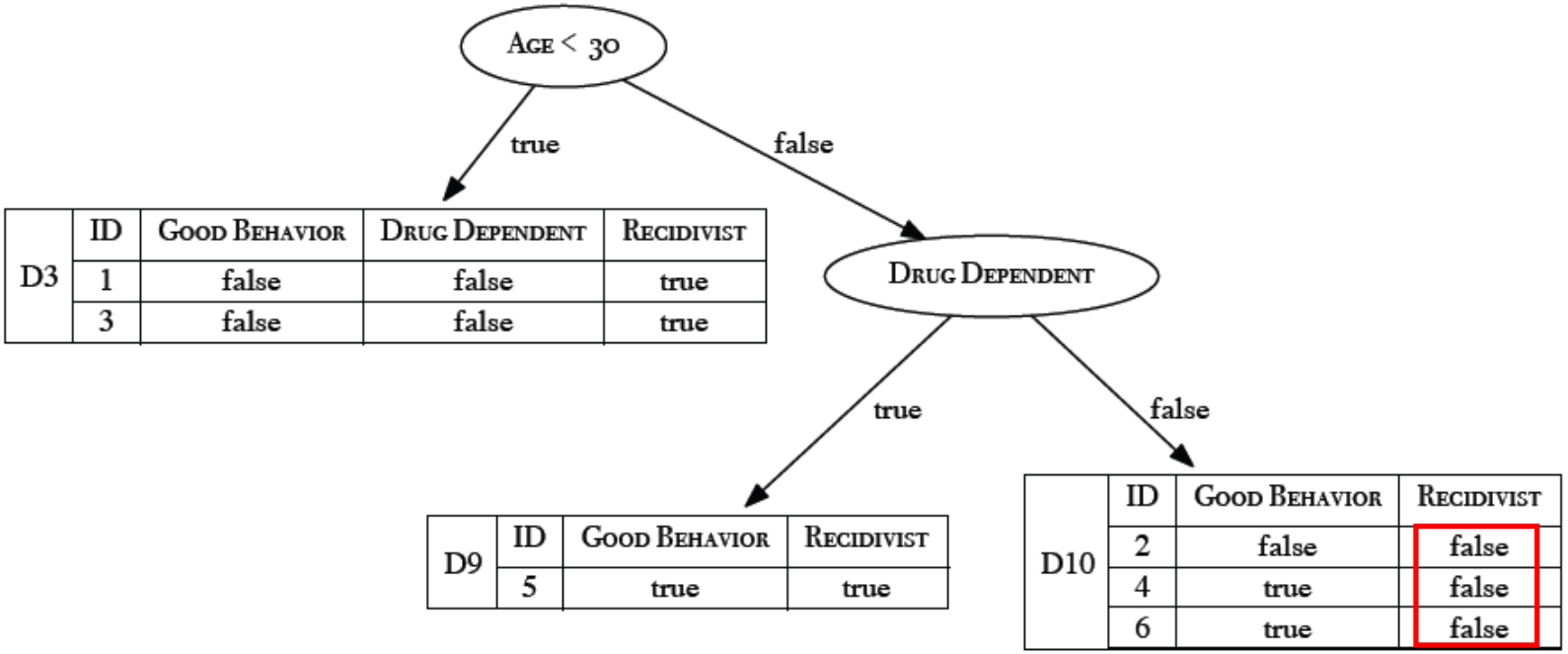
true:1/4

False:3/4

- The dataset on the right branch of the tree (D_4) is not homogenous, so we need to grow this branch of the tree. The entropy for this dataset, D_4, is:

$$H(\text{RECIDIVIST}, \mathcal{D}_4)$$
$$= -\sum_{l \in \{true, false\}} P(\text{RECIDIVIST} = l) \times log_2 (P(\text{RECIDIVIST} = l))$$
$$= -\left(\left({}^1/_4 \times log_2({}^1/_4)\right) + \left({}^3/_4 \times log_2({}^3/_4)\right)\right) = 0.8113 \; bits$$

Split by Feature	Level	Part.	Instances	Partition Entropy	Rem.	Info. Gain	
GOOD BEHAVIOR	*true*	$\mathcal{D}_7$	$\mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6$	0.918295834	0.4591	0.3522	=0.8113-0.4591
	false	$\mathcal{D}_8$	$\mathbf{d}_2$	0			
DRUG DEPENDENT	*true*	$\mathcal{D}_9$	$\mathbf{d}_5$	0	0	0.8113	=0.8113-0
	false	$\mathcal{D}_{10}$	$\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_6$	0			



	ID	Good Behavior	Drug Dependent	Recidivist
D3	1	false	false	true
	3	false	false	true

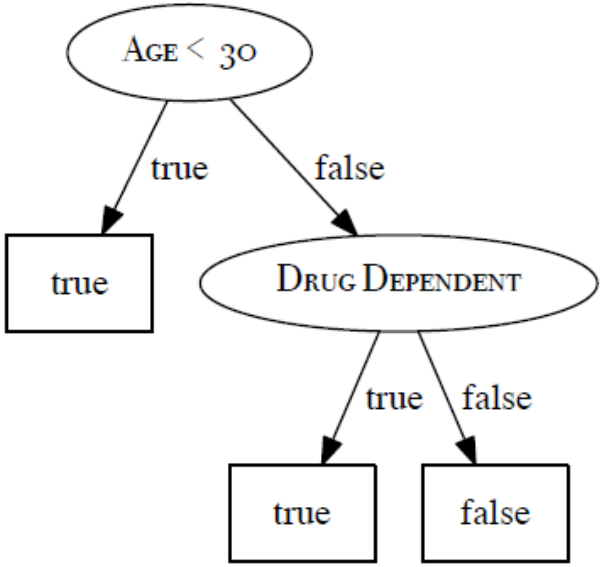
	ID	Good Behavior	Recidivist
D9	5	true	true

	ID	Good Behavior	Recidivist
D10	2	false	false
	4	true	false
	6	true	false

No further split

AGE <30 = false,
DRUG DEPENDENT = true
GOOD BEHAVIOR = false, (無用)
→RECIDIVIST = true

AGE <30 = true,
GOOD BEHAVIOR = true, (無用)
DRUG DEPENDENT = false (無用)
→RECIDIVIST = true



Example 2

ID	AGE	EDUCATION	MARITAL STATUS	OCCUPATION	ANNUAL INCOME
1	39	bachelors	never married	transport	25K–50K
2	50	bachelors	married	professional	25K–50K
3	18	high school	never married	agriculture	<25K
4	28	bachelors	married	professional	25K–50K
5	37	high school	married	agriculture	25K–50K
6	24	high school	never married	armed forces	<25K
7	52	high school	divorced	transport	25K–50K
8	40	doctorate	married	professional	>50K

OCCUPATION :
transport = works in the transportation industry;
professional= doctors, lawyers, etc.;
agriculture = works in the agricultural industry;
armed forces = is a member of the armed forces;

Chap4. Exercise 3. John D. Kelleher, Brian Mac Namee, Aoife D'Arcy, **FUNDAMENTALS OF MACHINE LEARNING FOR PREDICTIVE DATA ANALYTICS, 2015**

Calculate the entropy:

$$
\begin{aligned}
&H(\text{ANNUAL INCOME}, \mathcal{D}) \\
&= - \sum_{l \in \left\{ \begin{smallmatrix} <25K, \\ 25K\text{–}50K, \\ >50K \end{smallmatrix} \right\}} P(\text{AN. INC.} = l) \times log_2 \left(P(\text{AN. INC.} = l) \right) \\
&= - \left(\left(\frac{2}{8} \times log_2 \left(\frac{2}{8} \right) \right) + \left(\frac{5}{8} \times log_2 \left(\frac{5}{8} \right) \right) + \left(\frac{1}{8} \times log_2 \left(\frac{1}{8} \right) \right) \right) \\
&= 1.2988 \; bits
\end{aligned}
$$

Calculate the Gini index:

$$
\begin{aligned}
&Gini(\text{ANNUAL INCOME}, \mathcal{D}) \\
&= 1 - \sum_{l \in \left\{ \begin{smallmatrix} <25K, \\ 25K\text{–}50K, \\ >50K \end{smallmatrix} \right\}} P(\text{AN. INC.} = l)^2 \\
&= 1 - \left(\left(\frac{2}{8} \right)^2 + \left(\frac{5}{8} \right)^2 + \left(\frac{1}{8} \right)^2 \right) = 0.5313
\end{aligned}
$$

First sort the instances according to the AGE feature:

ID	AGE	ANNUAL INCOME
3	18	<25K
6	24	<25K
4	28	25K–50K
5	37	25K–50K
1	39	25K–50K
8	40	>50K
2	50	25K–50K
7	52	25K–50K

26

39.5

45

The mid-points in the AGE values that are adjacent in the new ordering but that have different target levels define the possible threshold points: 26, 39.5, and 45.

Split by Feature	Partition	Instances	Partition Entropy	Rem.	Info. Gain
>26	$\mathcal{D}_1$	$\mathbf{d}_3, \mathbf{d}_6$	0	0.4875	0.8113
	$\mathcal{D}_2$	$\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_7, \mathbf{d}_8$	0.6500		
>39.5	$\mathcal{D}_3$	$\mathbf{d}_1, \mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6$	0.9710	0.9456	0.3532
	$\mathcal{D}_4$	$\mathbf{d}_2, \mathbf{d}_7, \mathbf{d}_8$	0.9033		
>45	$\mathcal{D}_5$	$\mathbf{d}_1, \mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_8$	1.4591	1.0944	0.2044
	$\mathcal{D}_6$	$\mathbf{d}_2, \mathbf{d}_7$	0		

Split by Feature	Level	Instances	Partition Entropy	Rem.	Info. Gain
EDUCATION	*high school*	$\mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_7$	1.0	0.5	0.7988
	bachelors	$\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3$	0		
	doctorate	$\mathbf{d}_8$	0		
MARITAL STATUS	*never married*	$\mathbf{d}_1, \mathbf{d}_3, \mathbf{d}_6$	0.9183	0.75	0.5488
	married	$\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_8$	0.8113		
	divorced	$\mathbf{d}_7$	0		
OCCUPATION	*transport*	$\mathbf{d}_1, \mathbf{d}_7$	0	0.5944	0.7044
	professional	$\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_8$	0.9183		
	agriculture	$\mathbf{d}_3, \mathbf{d}_5$	1.0		
	armed forces	$\mathbf{d}_6$	0		

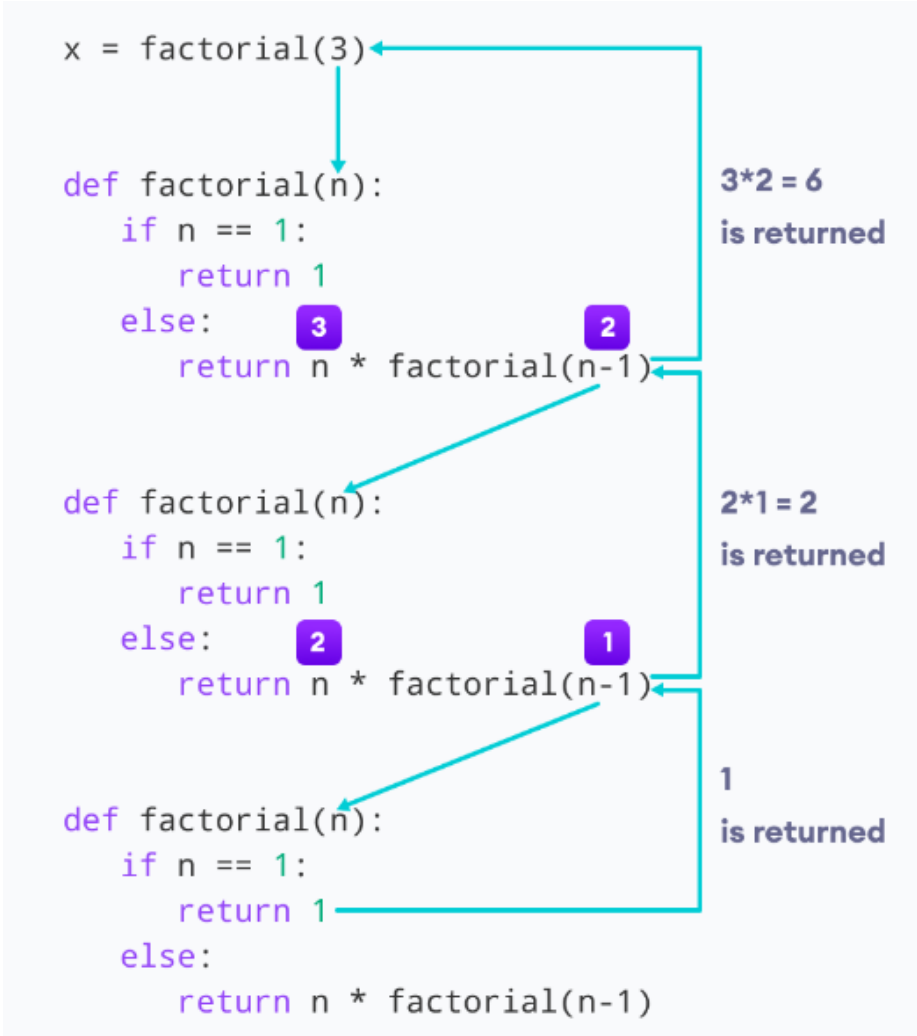
Finish the tree splitting of Example 2. (Homework 13-1)

Chap4. Exercise 3. John D. Kelleher, Brian Mac Namee, Aoife D'Arcy, **FUNDAMENTALS OF MACHINE LEARNING FOR PREDICTIVE DATA ANALYTICS, 2015**

Example of a recursive function

```
# Factorial of a number using recursion

def factorial(n):
    if n == 1:
        return n
    else:
        print("n is", n)
        return n*factorial(n-1)
num = 3

# check if the number is negative
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    print("The final factorial of", num, "is", factorial(num))
```

```
n is 3
n is 2
The final factorial of 3 is 6
```



Python Recursion (Recursive Function) (programiz.com)

Algorithm of Decision Tree

Training phase: Build the tree

1. Start at the top node and at each node select the best split based on the largest information gain.
2. Greedy search: Loop over all features and over all thresholds (all possible feature values).
3. Save the best split feature and split threshold at each node.
4. Build the tree recursively.
5. Apply the stopping criteria (maximum depth, minimum samples at node, no more class distribution in node) to stop growing
6. When we reach a leaf node, store the most common class label of this node

Algorithm of Decision Tree

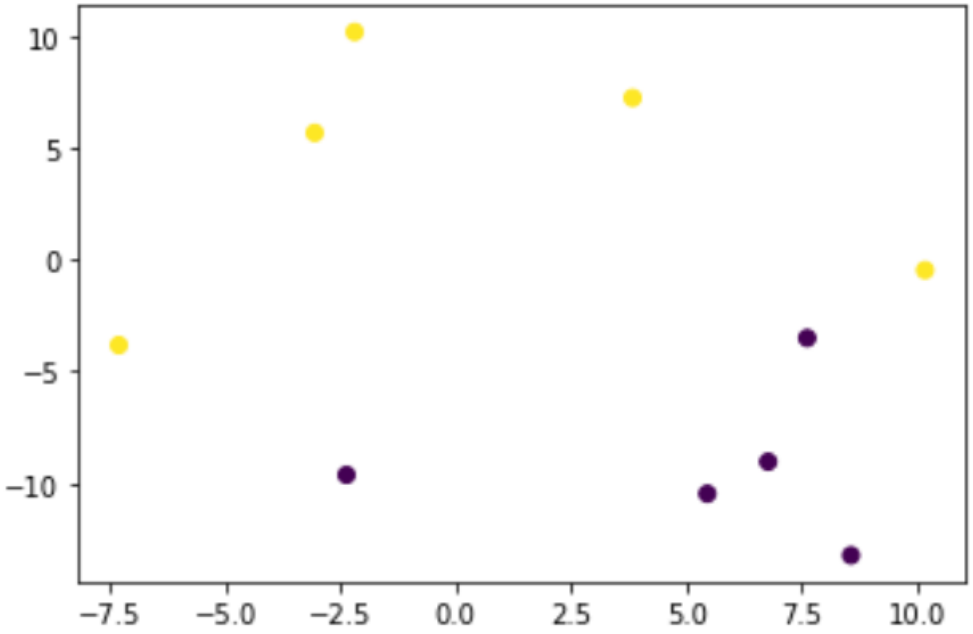
Testing phase: Traverse the tree

1. Traverse the tree recursively.
2. At each node, look at the best split feature of the test feature vector ***x*** and go left or right depending on ***x***[feature_idx] <= threshold
3. When we reach a leaf node, we return the stored most common class label of the node

Lecture 13 decision_tree_oop.jpynb

```python
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt

X, y = datasets.make_blobs(n_samples=10, n_features=2, centers=2, cluster_std=5.05, random_state=10)
print(y)
# y = np.where(y == 0, -1, 1) #將 y = 0 變成 -1，計算entropy會出錯
print(X)
```

```
[0 1 1 1 0 1 0 0 0 1]
[[  8.56415953 -13.22139309]
 [  3.82754686   7.2240226 ]
 [ 10.16987656  -0.47693702]
 [ -3.06687647   5.65851889]
 [ -2.37785861  -9.62729945]
 [ -2.20061694  10.16886174]
 [  6.76724637  -9.03679096]
 [  5.44808459 -10.46669208]
 [  7.61319512  -3.50962228]
 [ -7.31456312  -3.82795244]]
```



Lecture 13 decision_tree_oop.jpynb

```
n_samples, n_features = X.shape
n_labels = len(np.unique(y))
feat_idxs = np.random.choice(n_features, n_features, replace=False)
print(feat_idxs)
X_column = X[:, 1]
print(X_column)
X_column = X[:, 0]
print(X_column)
```

```
[1 0]
[-13.22139309   7.2240226   -0.47693702   5.65851889  -9.62729945
  10.16886174  -9.03679096 -10.46669208  -3.50962228  -3.82795244]
[ 8.56415953  3.82754686 10.16987656 -3.06687647 -2.37785861 -2.20061694
  6.76724637  5.44808459  7.61319512 -7.31456312]
```

```
[0 1 1 1 0 1 0 0 0 1]
[[  8.56415953 -13.22139309]
 [  3.82754686   7.2240226 ]
 [ 10.16987656  -0.47693702]
 [ -3.06687647   5.65851889]
 [ -2.37785861  -9.62729945]
 [ -2.20061694  10.16886174]
 [  6.76724637  -9.03679096]
 [  5.44808459 -10.46669208]
 [  7.61319512  -3.50962228]
 [ -7.31456312  -3.82795244]]
```

```
def entropy(y):
    hist = np.bincount(y)
    ps = hist / len(y)
    return -np.sum([p * np.log2(p) for p in ps if p > 0])
```

```
[0 1 1 1 0 1 0 0 0 1]
[5 5]
[0.5 0.5]
```

$$H = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Lecture 13 decision_tree_oop.jpynb

```
parent_entropy = entropy(y)
print(parent_entropy)
for feat_idx in feat_idxs:
    X_column = X[:, feat_idx]
    thresholds = np.unique(X_column)
    print(thresholds)
    for split_thresh in thresholds:
        print(np.argwhere(X_column <= split_thresh))
        left_idxs = np.argwhere(X_column <= split_thresh).flatten()
        print('left_idxs=',left_idxs)
        right_idxs = np.argwhere(X_column > split_thresh).flatten()
        print('right_idxs=',right_idxs)

        # compute the weighted avg. of the loss for the children
        n = len(y)
        n_l, n_r = len(left_idxs), len(right_idxs)
        e_l, e_r = entropy(y[left_idxs]), entropy(y[right_idxs])
        child_entropy = (n_l / n) * e_l + (n_r / n) * e_r

        # information gain is difference in loss before vs. after split
        ig = parent_entropy - child_entropy
        print('ig=',ig)
```

```
1.0
[-13.22139309 -10.46669208  -9.62729945  -9.03679096  -3.82795244
  -3.50962228  -0.47693702   5.65851889   7.2240226   10.16886174]
[[0]]
left_idxs= [0]
right_idxs= [1 2 3 4 5 6 7 8 9]
ig= 0.10803154614559995
[[0]
 [7]]
left_idxs= [0 7]
right_idxs= [1 2 3 4 5 6 8 9]
ig= 0.2364527976600279
[[0]
 [4]
 [7]]
left_idxs= [0 4 7]
right_idxs= [1 2 3 5 6 8 9]
ig= 0.3958156020033583
[[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
left_idxs= [0 1 2 3 4 5 6 7 8 9]
right_idxs= []
ig= 0.0
```

Lecture 13 decision_tree_oop.jpynb

```
def _split(X_column, split_thresh):
    left_idxs = np.argwhere(X_column <= split_thresh).flatten()
    right_idxs = np.argwhere(X_column > split_thresh).flatten()
    return left_idxs, right_idxs

def _information_gain(y, X_column, split_thresh):
     # parent loss
    parent_entropy = entropy(y)
     # generate split
    left_idxs, right_idxs = _split(X_column, split_thresh)
    if len(left_idxs) == 0 or len(right_idxs) == 0:
        return 0

    # compute the weighted avg. of the loss for the children
    n = len(y)
    n_l, n_r = len(left_idxs), len(right_idxs)
    e_l, e_r = entropy(y[left_idxs]), entropy(y[right_idxs])
    child_entropy = (n_l / n) * e_l + (n_r / n) * e_r

    # information gain is difference in loss before vs. after split
    ig = parent_entropy - child_entropy
    return ig
```

Lecture 13 decision_tree_oop.jpynb

1. Start at the top node and at each node select the best split based on the largest information gain.
2. Greedy search: Loop over all features and over all thresholds (all possible feature values).

```
def _best_criteria(self, X, y, feat_idxs):
best_gain = -1
split_idx, split_thresh = None, None
for feat_idx in feat_idxs:
    X_column = X[:, feat_idx]
    thresholds = np.unique(X_column)
    for threshold in thresholds:
        gain = _information_gain(y, X_column, threshold)
        if gain > best_gain:
            best_gain = gain
            split_idx = feat_idx
            split_thresh = threshold
            print('split_idx=',split_idx,',split_thresh=',split_thresh, ',gain=',gain)
print('best_gain=',best_gain,' best_thresh=',split_thresh)
```

```
split_idx= 0 ,split_thresh= -7.314563118796922 ,gain= 0.10803154614559995
split_idx= 0 ,split_thresh= -3.0668764712212644 ,gain= 0.2364527976600279
split_idx= 0 ,split_thresh= 3.827546855655039 ,gain= 0.2780719051126377
split_idx= 1 ,split_thresh= -9.627299454951007 ,gain= 0.3958156020033583
split_idx= 1 ,split_thresh= -9.036790957951375 ,gain= 0.6099865470109875
best_gain= 0.6099865470109875  best_thresh= -9.036790957951375
```

Lecture 13 decision_tree_oop.jpynb

```
def _grow_tree(self, X, y, depth=0):
    n_samples, n_features = X.shape
    n_labels = len(np.unique(y))
    # stopping criteria
    if (
        depth >= self.max_depth
        or n_labels == 1
        or n_samples < self.min_samples_split
    ):
        leaf_value = self._most_common_label(y)
        return Node(value=leaf_value)
    print('n_features, self.n_feats',n_features, self.n_feats,)
    feat_idxs = np.random.choice(n_features, self.n_feats, replace=False)

    # greedily select the best split according to information gain
    best_feat, best_thresh = self._best_criteria(X, y, feat_idxs)

    # grow the children that result from the split
    left_idxs, right_idxs = self._split(X[:, best_feat], best_thresh)
    left = self._grow_tree(X[left_idxs, :], y[left_idxs], depth + 1)
    right = self._grow_tree(X[right_idxs, :], y[right_idxs], depth + 1)
    return Node(best_feat, best_thresh, left, right)
```

recursive

Lecture 13 decision_tree_oop.jpynb

```python
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None, *, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf_node(self):
        return self.value is not None # self.value != None
```

Lecture 13 decision_tree_oop.jpynb

```
def predict(self, X):
    return np.array([self._traverse_tree(x, self.root) for x in X])
```

```
def _traverse_tree(self, x, node):
    if node.is_leaf_node():
        return node.value

    if x[node.feature] <= node.threshold:
        return self._traverse_tree(x, node.left)
    return self._traverse_tree(x, node.right)
```

```
def fit(self, X, y):
    self.n_feats = X.shape[1] if not self.n_feats else min(self.n_feats, X.shape[1])
    self.root = self._grow_tree(X, y)
```

```
clf = DecisionTree(max_depth=3)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
acc = accuracy(y_test, y_pred)
```

Lecture 13 Node_example.jpynb

Create Root:

We just create a Node class and add assign a value to the node. This becomes tree with only a root node

```
class Node:
   def __init__(self, data):
      self.left = None
      self.right = None
      self.data = data
   def PrintTree(self):
      print(self.data)

root = Node(10)
root.PrintTree()
```

```
10
```

Lecture 13 Node_example.jpynb

Inserting into a Tree:

To insert into a tree we use the same node class created above and add a insert class to it. The insert class compares the value of the node to the parent node and decides to add it as a left node or a right node. Finally the PrintTree class is used to print the tree.

```
class Node:
   def __init__(self, data):
      self.left = None
      self.right = None
      self.data = data

   def insert(self, data):
# Compare the new value with the parent node
      if self.data:
         if data < self.data:
            if self.left is None:
               self.left = Node(data)
            else:
               self.left.insert(data)
         elif data > self.data:
               if self.right is None:
                  self.right = Node(data)
         else:
            self.right.insert(data)
      else:
         self.data = data
```

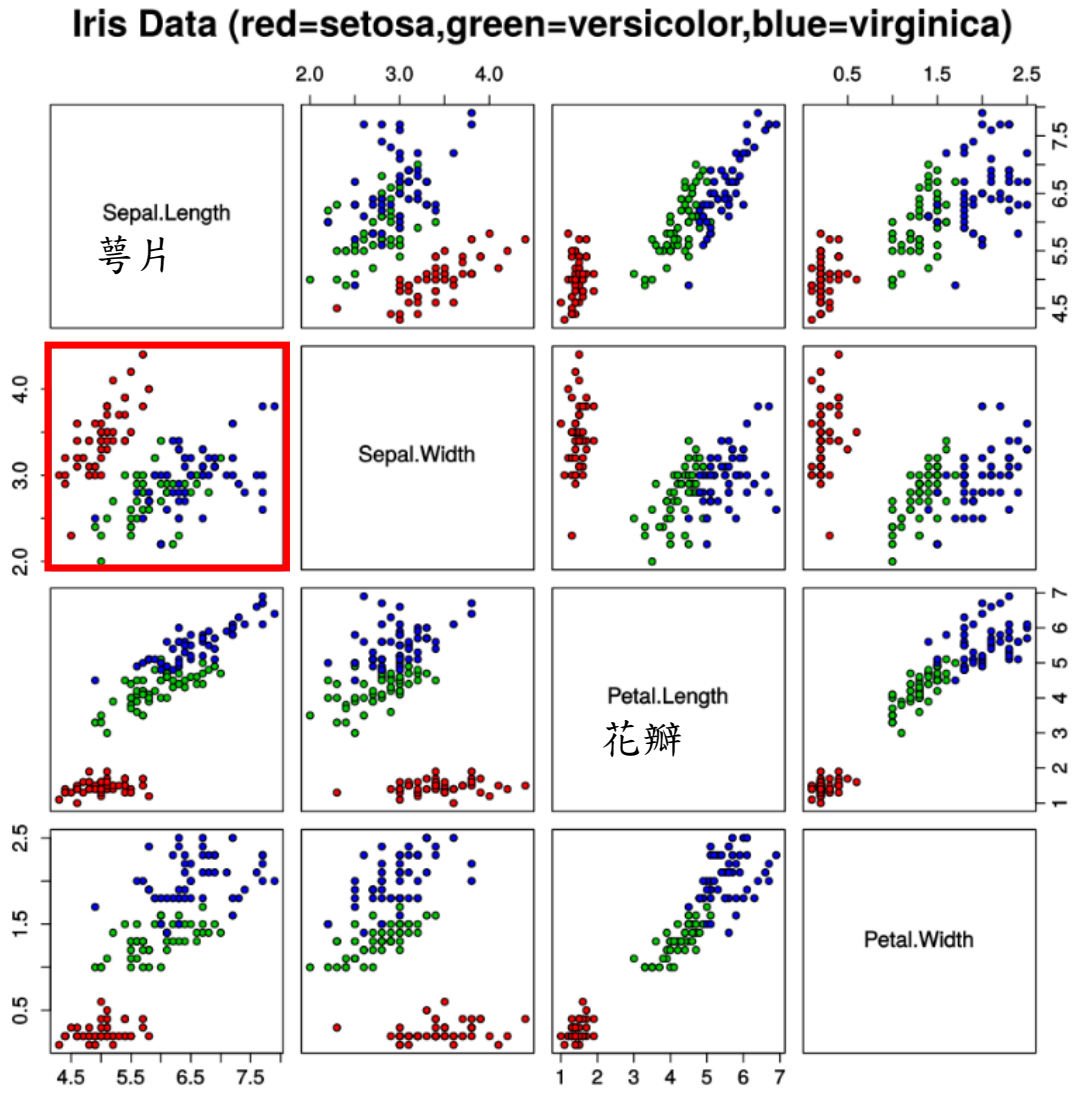
```
# Print the tree
   def PrintTree(self):
      if self.left:
         self.left.PrintTree()
      print( self.data),
      if self.right:
         self.right.PrintTree()

# Use the insert method to add nodes
root = Node(12)
root.insert(6)
root.insert(14)
root.insert(3)
root.PrintTree()
```

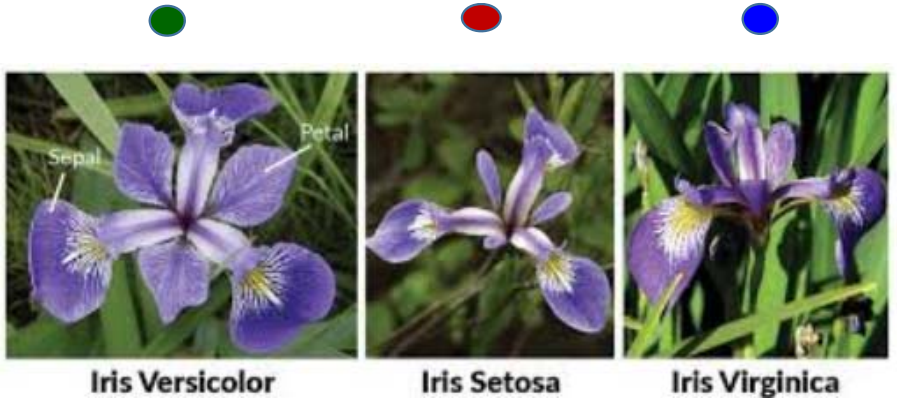
```
3
6
12
14
```

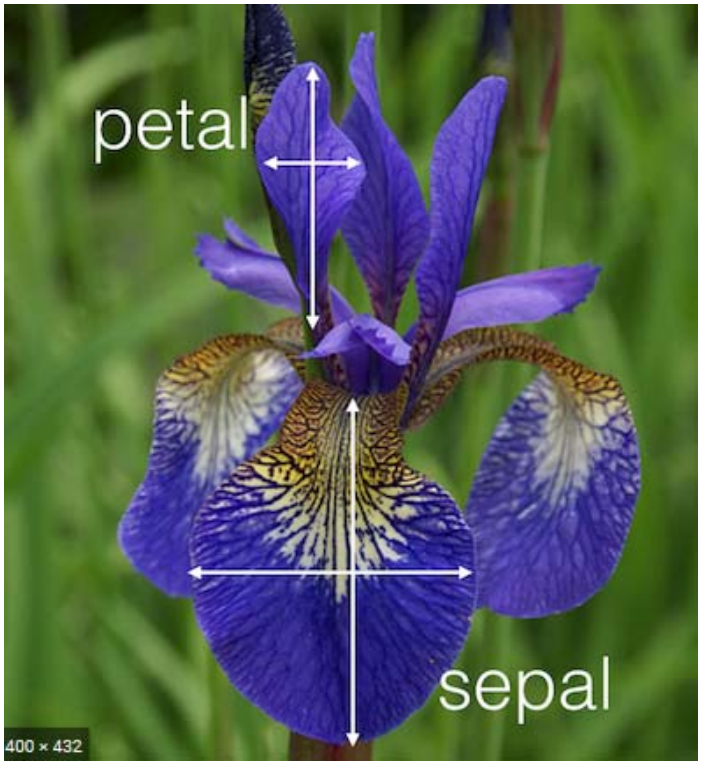
安德森鳶尾花卉數據集

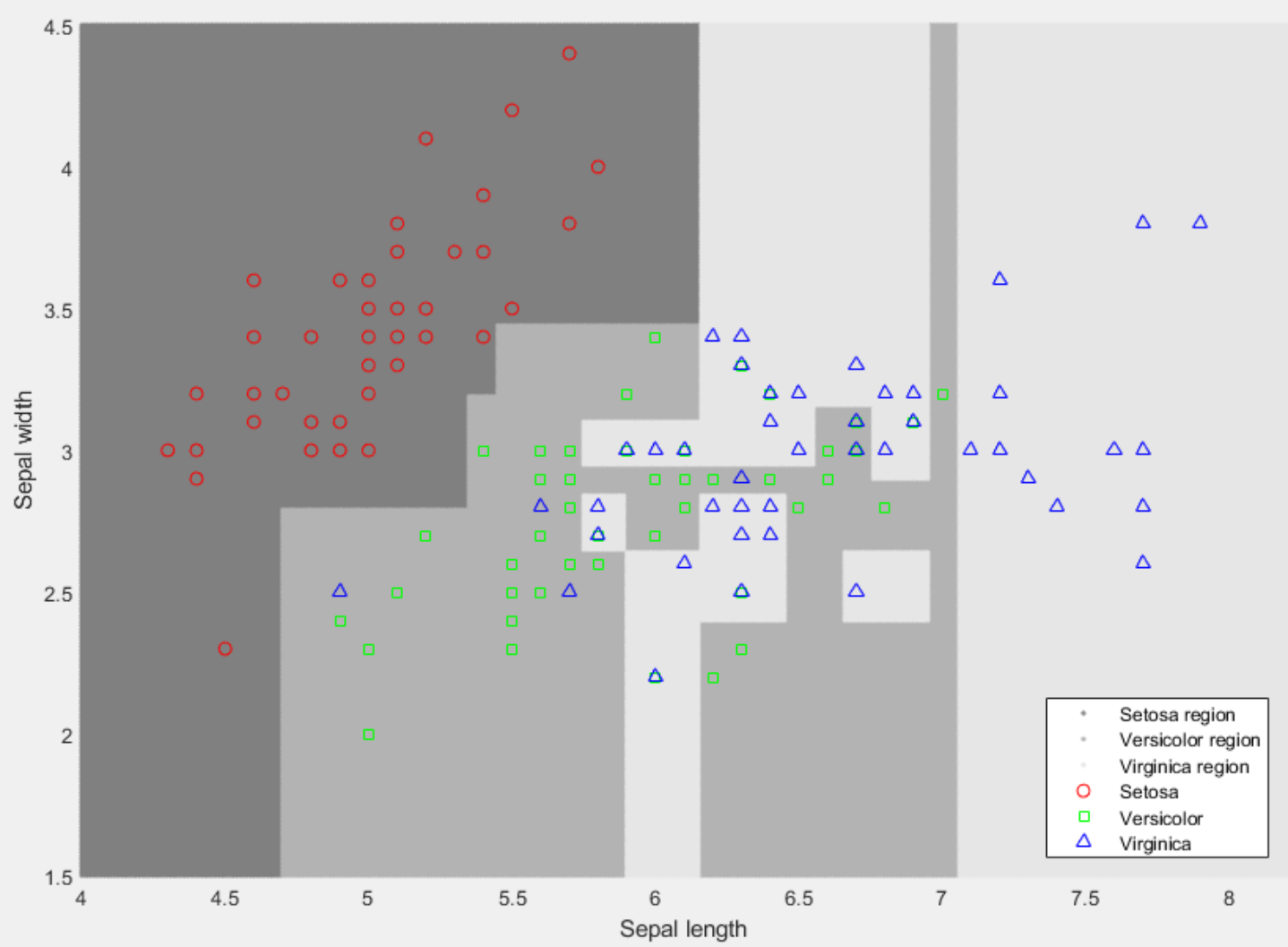
- **安德森鳶尾花卉數據集**（Anderson's Iris data set），也稱**鳶尾花卉數據集**（Iris flower data set）或**費雪鳶尾花卉數據集**（Fisher's Iris data set），是一類多重變量分析的數據集。
- 它最初是埃德加·安德森（Edgar Anderson）從加拿大加斯帕半島上的鳶尾屬花朵中提取的形態學變異數據，後由羅納德·費雪作為判別分析的一個例子，運用到統計學中。
- 其數據集包含了150個樣本，都屬於鳶尾屬下的三個亞屬，分別是山鳶尾、變色鳶尾和維吉尼亞鳶尾（Virginia Iris）。
- 四個特徵被用作樣本的定量分析，它們分別是花萼和花瓣的長度和寬度。

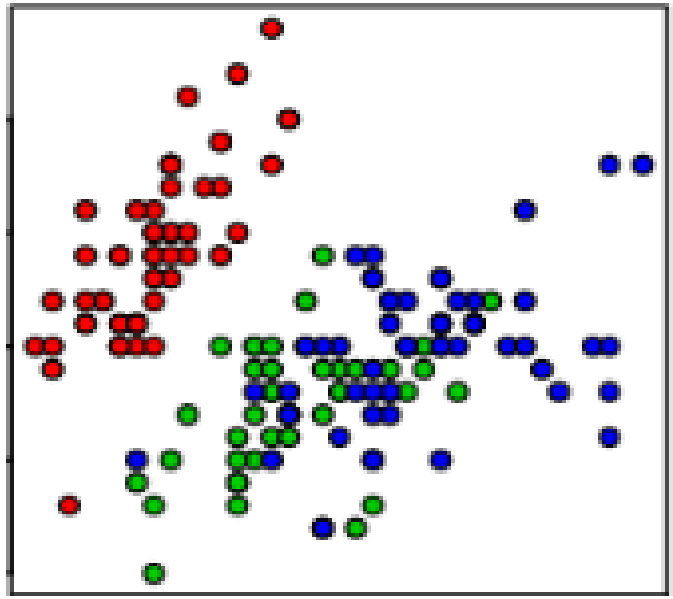
https://zh.wikipedia.org/wiki/%E5%AE%89%E5%BE%B7%E6%A3%AE%E9%B8%A2%E5%B0%BE%E8%8A%B1%E5%8D%89%E6%95%B0%E6%8D%AE%E9%9B%86

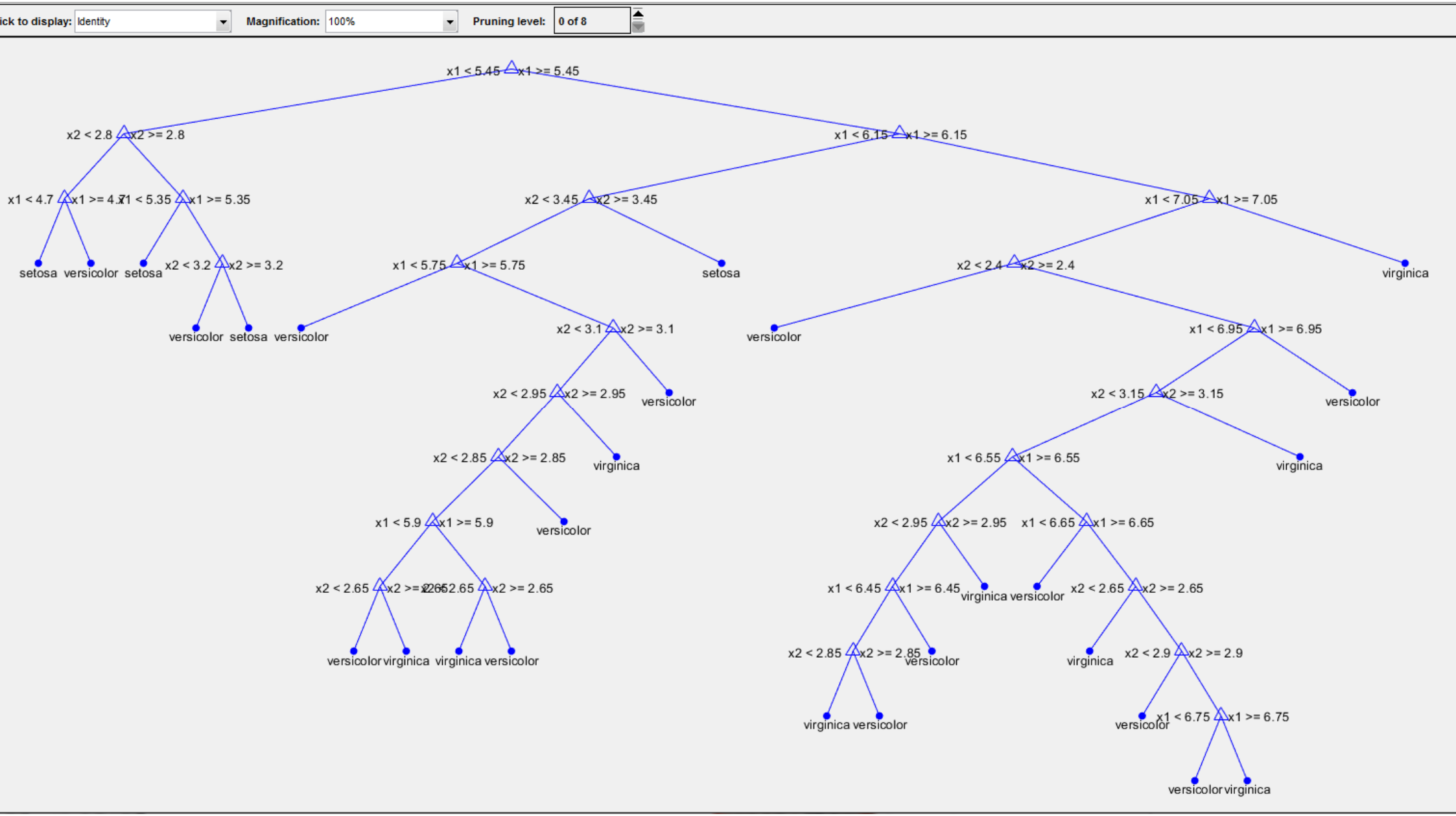

Iris Data (red=setosa,green=versicolor,blue=virginica)
Sepal.Length
萼片
Sepal.Width
Petal.Length
花瓣
Petal.Width


Sepal
Petal
Iris Versicolor
Iris Setosa
Iris Virginica


petal
sepal
400 × 432


4.5
4
3.5
3
2.5
2
1.5
Sepal width
4
4.5
5
5.5
6
6.5
7
7.5
8
Sepal length
Setosa region
Versicolor region
Virginica region
Setosa
Versicolor
Virginica

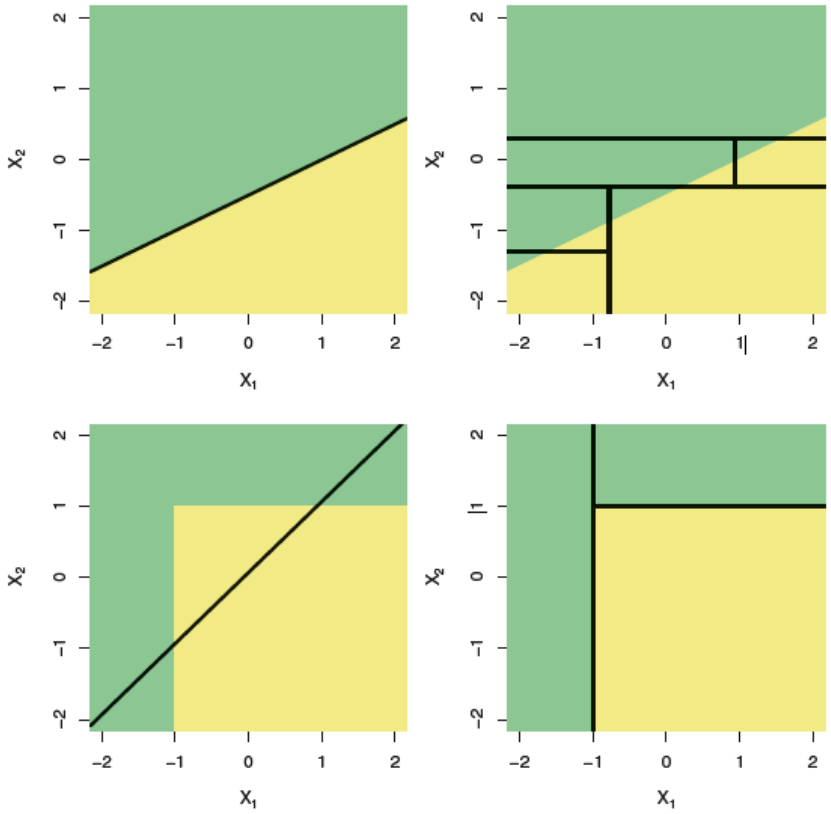



ick to display: Identity
Magnification: 100%
Pruning level: 0 of 8
x1 < 5.45
x1 >= 5.45
x2 < 2.8
x2 >= 2.8
x1 < 6.15
x1 >= 6.15
x1 < 4.7
x1 >= 4.7
x1 < 5.35
x1 >= 5.35
x2 < 3.45
x2 >= 3.45
x1 < 7.05
x1 >= 7.05
setosa
versicolor
setosa
x2 < 3.2
x2 >= 3.2
x1 < 5.75
x1 >= 5.75
setosa
x2 < 2.4
x2 >= 2.4
virginica
versicolor
setosa
versicolor
x2 < 3.1
x2 >= 3.1
versicolor
x1 < 6.95
x1 >= 6.95
x2 < 2.95
x2 >= 2.95
versicolor
x2 < 3.15
x2 >= 3.15
versicolor
x2 < 2.85
x2 >= 2.85
virginica
x1 < 6.55
x1 >= 6.55
virginica
x1 < 5.9
x1 >= 5.9
versicolor
x2 < 2.95
x2 >= 2.95
x1 < 6.65
x1 >= 6.65
x2 < 2.65
x2 >= 2.65
x2 < 2.65
x2 >= 2.65
x1 < 6.45
x1 >= 6.45
virginica
versicolor
x2 < 2.65
x2 >= 2.65
versicolor
virginica
virginica
versicolor
x2 < 2.85
x2 >= 2.85
versicolor
virginica
x2 < 2.9
x2 >= 2.9
virginica
versicolor
versicolor
x1 < 6.75
x1 >= 6.75
versicolor
virginica

```
% Fishertree.m from  A Concise Introduction to Machine Learning, 2020 Anita C. Faul
load fisheriris
% Extract two attributes.
sl = meas(:,1); % sepal length
sw = meas(:,2); % sepal width
X = [sl,sw];

% Create classifier.
% The depth of a decision tree is governed by three arguments:
% Maximum number of branch node splits; a large value results in a deep tree.
MaxNumSplits = size(X,1) - 1;

% Minimum number of samples per branch node; a small number results in a deep tree.
MinParentSize = 5;

% Minimum number of samples per leaf; a small number results in a deep tree.
MinLeafSize = 1;

treeModel = fitctree(X,species,...
    'MaxNumSplits',MaxNumSplits,...
    'MinLeafSize',MinLeafSize,...
    'MinParentSize',MinParentSize);
view(treeModel,'mode','graph') % visualization
```

```
% Lay grid over the region
d = 0.01;
[x1Grid,x2Grid] = meshgrid(4:d:8.2,1.5:d:4.5);
xGrid = [x1Grid(:),x2Grid(:)];  N = size(xGrid,1);
% For each grid point calculate the score of each class.
% 'predict' returns the predicted class labels corresponding to the
% minimum misclassification cost, the score (posterior probability)
% for each class as well as the predicted node number and class number.

[~,score,~,~] = predict(treeModel,xGrid);
% Classify according to the maximum score.
[~,maxScore] = max(score,[],2);

% Plot classifier regions.
figure
h(1:3) = gscatter(xGrid(:,1),xGrid(:,2),maxScore,...
    [0.5 0.5 0.5; 0.7 0.7 0.7; 0.9 0.9 0.9]);
hold on
% Plot data.
h(4:6) = gscatter(sl, sw, species,'rgb','os^');
xlabel('Sepal length'); ylabel('Sepal width');
legend(h,{'Setosa region','Versicolor region','Virginica region',...
          'Setosa','Versicolor','Virginica'},'Location','Southeast');
axis([4 8.2 1.5 4.5])
```

Tree Versus Linear Models



Top Row: A 2D classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).
Bottom Row: Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).

Advantages and Disadvantages of Trees

▲ Easy to explain to people !

▲ More closely mirror human decision making.

▲ Can be displayed graphically and easily interpreted even by a non-expert.

▲ Can easily handle qualitative predictors without creating dummy variables.

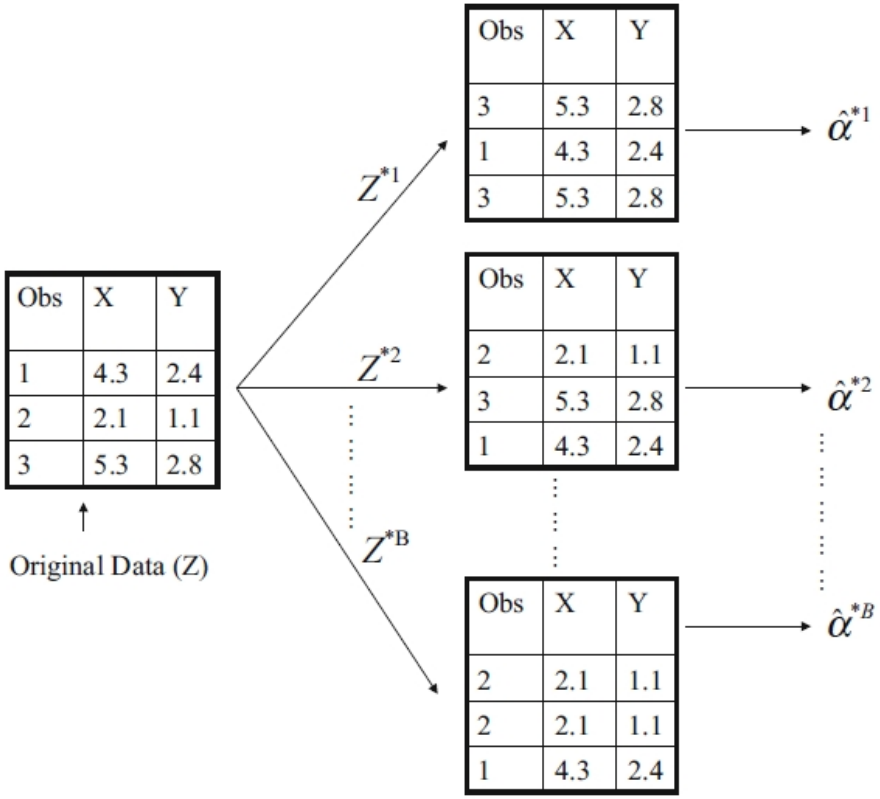
▼ Generally do not have the same level of predictive accuracy as some of the other classification approaches.

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

Bagging

- The bootstrap is an extremely powerful idea. It is used in many situations in which it is hard or even impossible to directly compute the standard deviation of a quantity of interest.
- **Bootstrap aggregation**, or **bagging**, is a general-purpose procedure for reducing the variance of a statistical learning method.

Example with just 3 observations

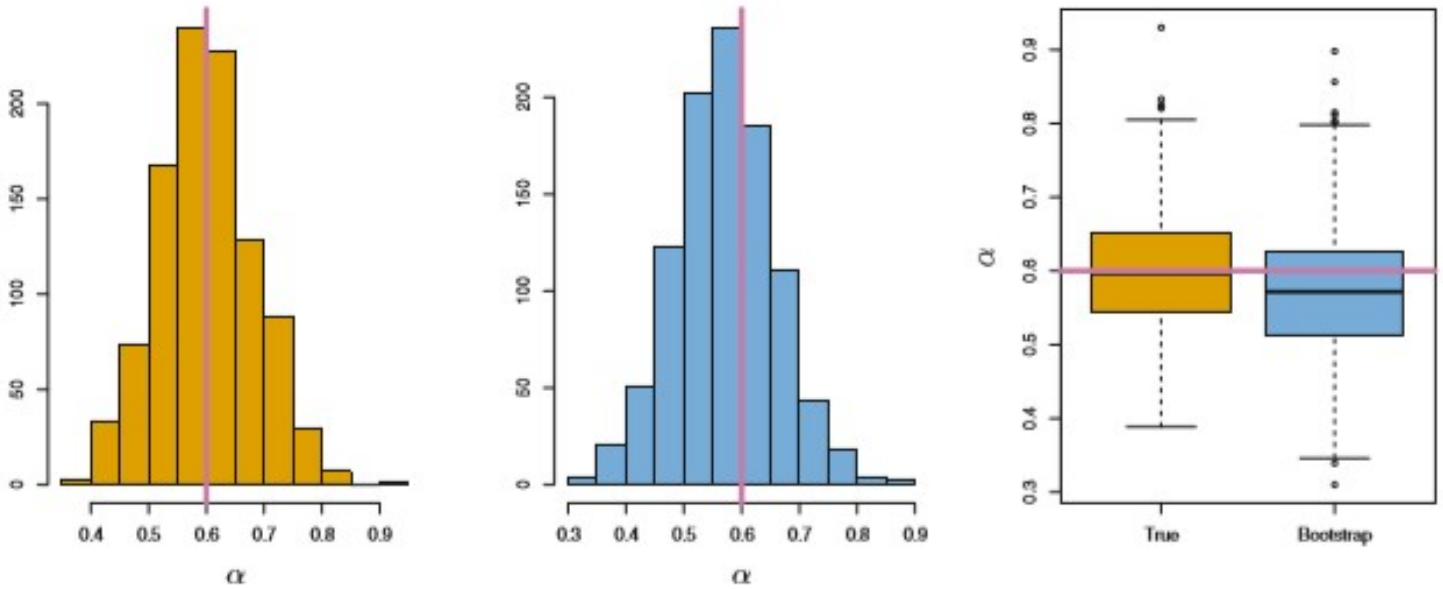


$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}.$$

A graphical illustration of the bootstrap approach on a small sample containing

n =3 observations. Each bootstrap data set contains n observations, sampled with replacement from the original data set. Each bootstrap data set is used to obtain an estimate of α

Results



Left: A histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population. Center: A histogram of the estimates of α obtained from 1,000 bootstrap samples from a single data set. Right: The estimates of α displayed in the left and center panels are shown as boxplots. In each panel, the pink line indicates the true value of α.

Bagging classification trees

- Bootstrap by taking repeated samples from the training data set.
- First generate b different bootstrapped training data sets.
- Then train the jth bootstrapped training set to get the predictions $\boldsymbol{x}$ at $\varphi_j(\boldsymbol{x})$.
- We then average all the predictions to obtain

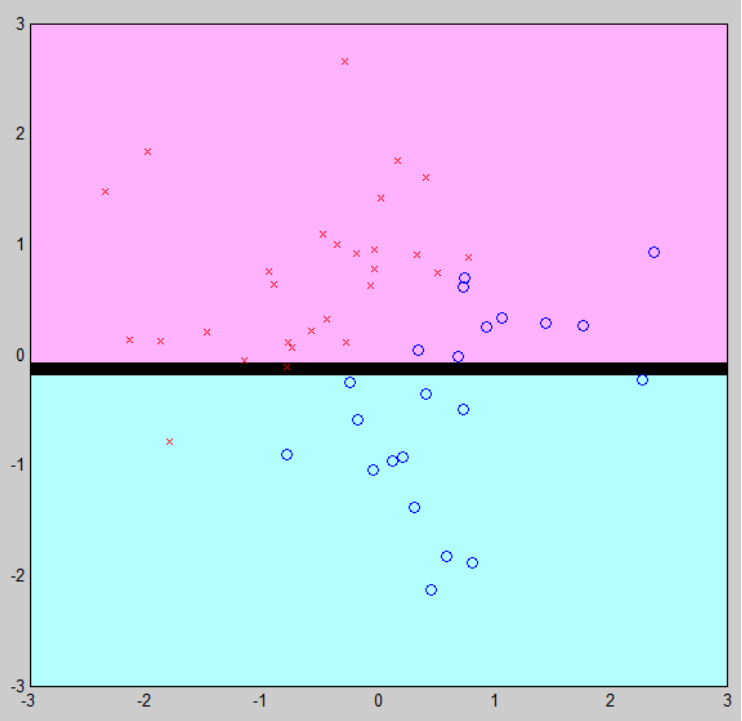
$$f \leftarrow \frac{1}{b}\sum_{j=1}^{b} \varphi_j(\boldsymbol{x})$$

This is called **bagging**.

- For each test observation, we record the class predicted by each of the j trees, and take a **majority vote**: the overall prediction is the most commonly occurring class among the b predictions.

MATLAB code for decision stump classification.

```
% A decision stump is a depth-one version of a decision tree.
% tree_stump.m
x=randn(50,2);
y=2*(x(:,1)>x(:,2))-1;
X0=linspace(-3,3,50);
[X(:,:,1) X(:,:,2)]=meshgrid(X0);
d=ceil(2*rand);
[xs,xi]=sort(x(:,d));
el=cumsum(y(xi));
eu=cumsum(y(xi(end:-1:1)));
e=eu(end-1:-1:1)-el(1:end-1);
[em,ei]=max(abs(e));
c=mean(xs(ei:ei+1));
s=sign(e(ei));
Y=sign(s*(X(:,:,d)-c));
figure(1); clf; hold on; axis([-3 3 -3 3]);
colormap([1 0.7 1; 0.7 1 1]); contourf(X0,X0,Y);
plot(x(y==1,1),x(y==1,2),'bo');
plot(x(y==-1,1),x(y==-1,2),'rx');
```



Bagging for decision stumps

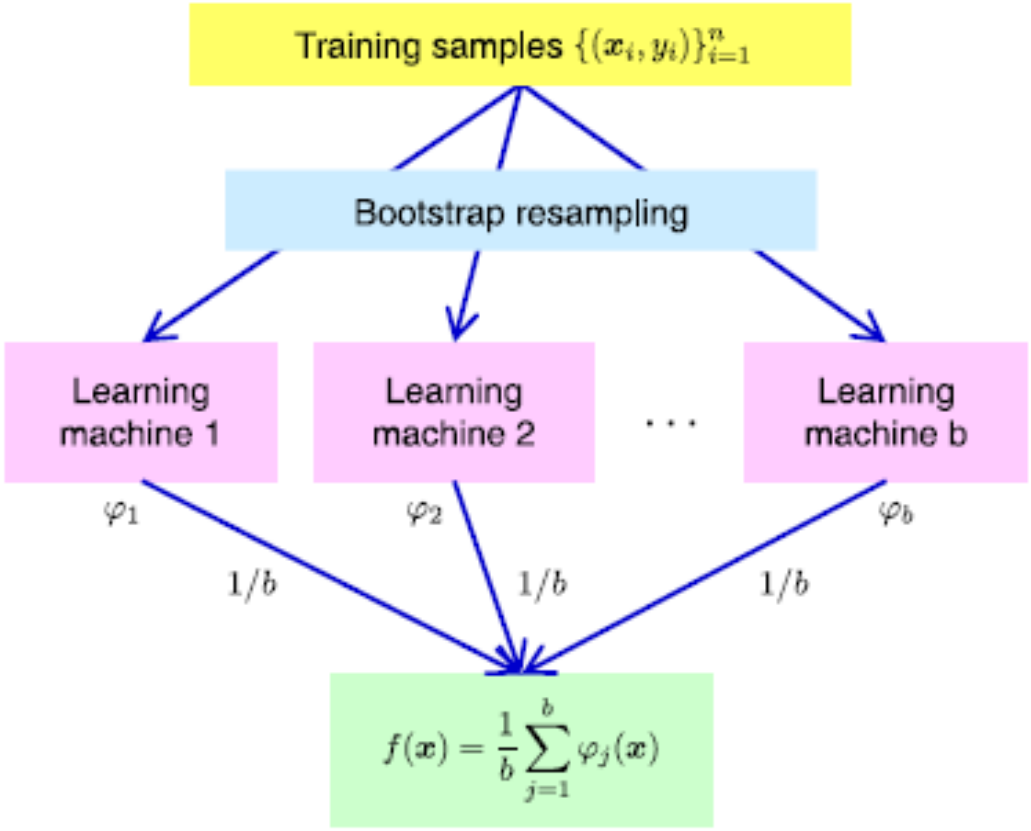
1. For $j = 1, \cdots, b$

(a) Randomly choose n samples from $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ with replacement.

(b) Train a classifier φ_j with the randomly resampled data set.

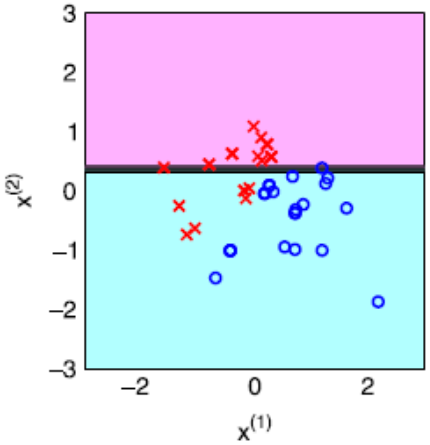
2. Output the average of $\{\varphi_j\}_{j=1}^{b}$ as the final solution f:

$$f \leftarrow \frac{1}{b}\sum_{j=1}^{b} \varphi_j(\boldsymbol{x})$$

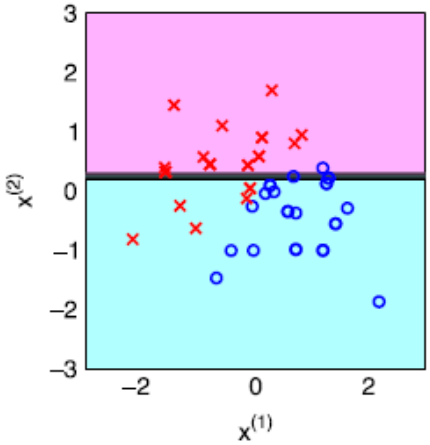


Masashi Sugiyama, Introduction to Statistical Machine Learning, 2016

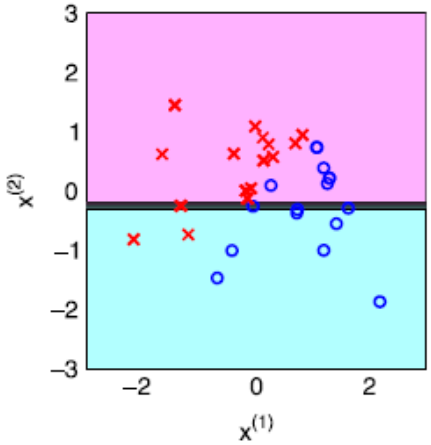
Example of bagging for decision stumps



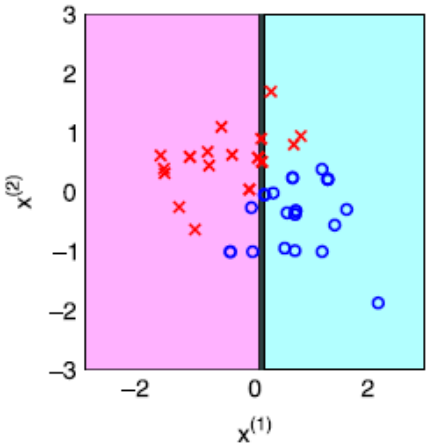
(a) $j = 1$



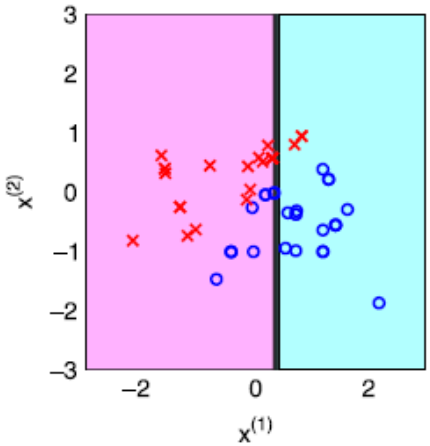
(b) $j = 2$



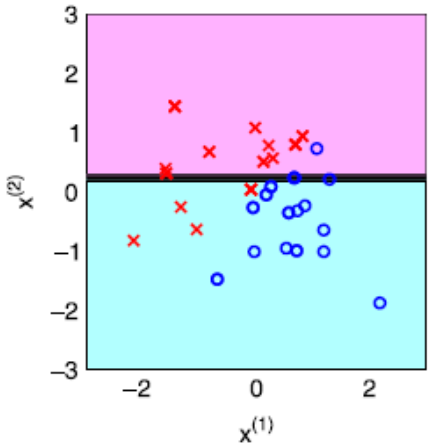
(c) $j = 3$



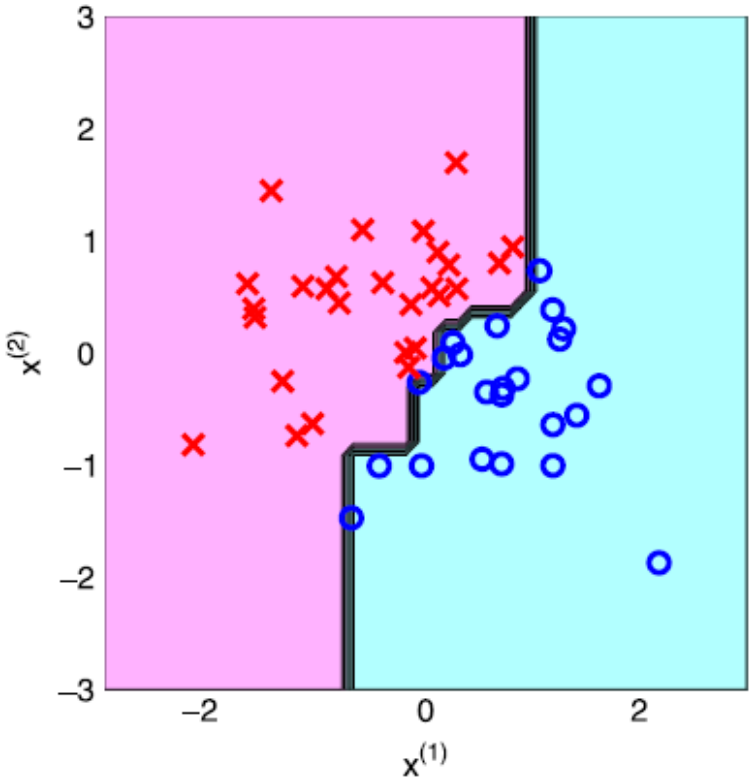
(d) $j = 4$



(e) $j = 5$



(f) $j = 6$



(g) Average of $b = 5000$ decision stumps

Masashi Sugiyama, Introduction to Statistical Machine Learning, 2016

```
% bagging for decision stumps
% bagging.m
n=50; x=randn(n,2);
y=2*(x(:,1)>x(:,2))-1;
b=5000; a=50; Y=zeros(a,a);
X0=linspace(-3,3,a);
[X(:,:,1) X(:,:,2)]=meshgrid(X0);
for j=1:b
  db=ceil(2*rand);
  r=ceil(n*rand(n,1));
  xb=x(r,:); yb=y(r);
  [xs,xi]=sort(xb(:,db));
  el=cumsum(yb(xi));
  eu=cumsum(yb(xi(end:-1:1)));
  e=eu(end-1:-1:1)-el(1:end-1);
  [em,ei]=max(abs(e)); c=mean(xs(ei:ei+1));
  s=sign(e(ei));
  Y=Y+sign(s*(X(:,:,db)-c))/b;
end
figure(1); clf; hold on; axis([-3 3 -3 3]);
colormap([1 0.7 1; 0.7 1 1]); contourf(X0,X0,sign(Y));
plot(x(y==1,1),x(y==1,2),'bo');
plot(x(y==-1,1),x(y==-1,2),'rx');
```

Random Forests

- **Random forests** provide an improvement over bagged trees by way of a random small tweak that **decorrelates** the trees. This reduces the variance when we average the trees.
- As is bagging, we build a number of decision trees on bootstrapped training samples.
- Each time a split in a tree, **a random selection of *m* predictors** is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$—that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors. For regression purpose, use $m \approx \frac{p}{3}$.

Out-of-Bag Error Estimation

- There is a very straightforward way to estimate the test error of a bagged model.
- The key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. On average each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the **out-of-bag** (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around b/3 predictions for the ith observation, which we average.

Example of Random Forest

The table lists the details of five participants in a heart disease study, and a target feature RISK which describes their risk of heart disease.
Each patient is described in terms of four binary descriptive features

- EXERCISE, how regularly do they exercise
- SMOKER, do they smoke
- OBESE, are they overweight
- FAMILY, did any of their parents or siblings suffer from heart disease

ID	EXERCISE	SMOKER	OBESE	FAMILY	RISK
1	daily	false	false	yes	low
2	weekly	true	false	yes	high
3	daily	false	false	no	low
4	rarely	true	true	yes	high
5	rarely	true	true	no	high

Step 1. Generate bootstrap samples and random selection of m=2 features

ID	EXERCISE	FAMILY	RISK
1	daily	yes	low
2	weekly	yes	high
2	weekly	yes	high
5	rarely	no	high
5	rarely	no	high

Bootstrap Sample A

ID	SMOKER	OBESE	RISK
1	false	false	low
2	true	false	high
2	true	false	high
4	true	true	high
5	true	true	high

Bootstrap Sample B

ID	OBESE	FAMILY	RISK
1	false	yes	low
1	false	yes	low
2	false	yes	high
4	true	yes	high
5	true	no	high

Bootstrap Sample C

The entropy calculation for Sample A:

$$H(\text{RISK}, BoostrapSampleA)$$
$$= -\sum_{l \in \{low, high\}} P(\text{RISK} = l) \times log_2 (P(\text{RISK} = l))$$
$$= -\left(\left(\frac{1}{5} \times log_2\left(\frac{1}{5}\right)\right) + \left(\frac{4}{5} \times log_2\left(\frac{4}{5}\right)\right)\right)$$
$$= 0.7219\ bits$$

The entropy calculation for Sample B:

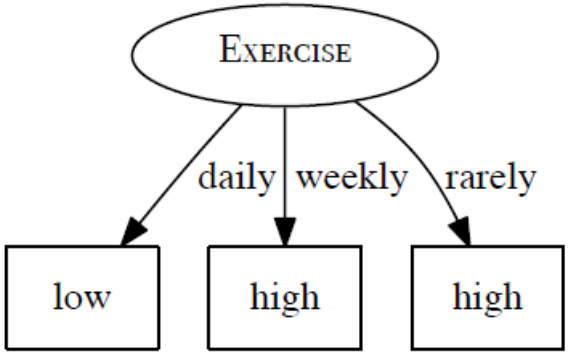
$$H(\text{RISK}, BoostrapSampleB)$$
$$= -\sum_{l \in \{low, high\}} P(\text{RISK} = l) \times log_2 (P(\text{RISK} = l))$$
$$= -\left(\left(\frac{1}{5} \times log_2\left(\frac{1}{5}\right)\right) + \left(\frac{4}{5} \times log_2\left(\frac{4}{5}\right)\right)\right)$$
$$= 0.7219\ bits$$

The entropy calculation for Sample C:

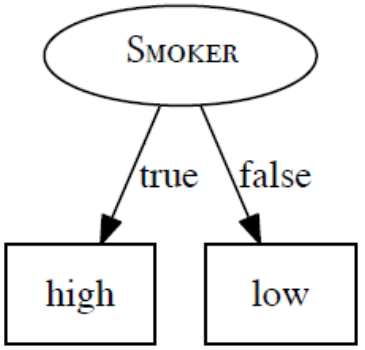
$$H(\text{RISK}, BoostrapSampleC)$$
$$= -\sum_{l \in \{low, high\}} P(\text{RISK} = l) \times log_2 (P(\text{RISK} = l))$$
$$= -\left(\left(\frac{2}{5} \times log_2\left(\frac{2}{5}\right)\right) + \left(\frac{3}{5} \times log_2\left(\frac{3}{5}\right)\right)\right)$$
$$= 0.9710\ bits$$

Step 2. Grow a tree from each bootstrap sample

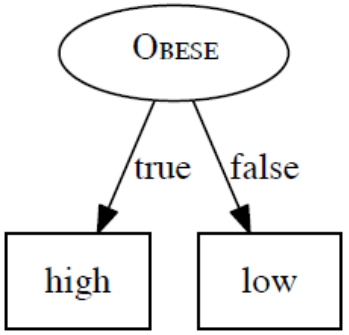
Split by Feature	Level	Instances	Partition Entropy	Rem.	Info. Gain
EXERCISE	*daily*	$\mathbf{d}_1$	0	0	0.7219
	weekly	$\mathbf{d}_2, \mathbf{d}_2$	0		
	rarely	$\mathbf{d}_5, \mathbf{d}_5,$	0		
FAMILY	*yes*	$\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_2$	0.9183	0.5510	0.1709
	no	$\mathbf{d}_5, \mathbf{d}_5$	0		



Split by Feature	Level	Instances	Partition Entropy	Rem.	Info. Gain
SMOKER	*true*	$\mathbf{d}_2, \mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_5$	0	0	0.7219
	false	$\mathbf{d}_1$	0		
OBESE	*true*	$\mathbf{d}_4, \mathbf{d}_5$	0	0.5510	0.1709
	false	$\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_2$	0.9183		



Split by Feature	Level	Instances	Partition Entropy	Rem.	Info. Gain
OBESE	*true*	$\mathbf{d}_4, \mathbf{d}_5$	0	0.5510	0.4200
	false	$\mathbf{d}_1, \mathbf{d}_1, \mathbf{d}_2$	0.9183		
FAMILY	*yes*	$\mathbf{d}_1, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_4$	1.0	0.8	0.1709
	no	$\mathbf{d}_5$	0		



Step 3. Compute Out-of-Bag Error

- The observations not used to fit a given bagged tree are the **out-of-bag** (OOB) observations.

- ID=3, EXERCISE= daily, SMOKER=false, OBESE= false, FAMILY= no

Each of the trees in the ensemble will vote as follows:
- Tree 1: EXERCISE= daily → RISK=low
- Tree 2: SMOKER=false→ RISK=low
- Tree 3: OBESE= false → RISK=low

So, the majority vote is for RISK=low, same with the target RISK=low

Step 4. Make prediction

Assuming the random forest model you have created uses majority voting, what prediction will it return for the following query:

EXERCISE=rarely, SMOKER=false, OBESE=true, FAMILY=yes

Each of the trees in the ensemble will vote as follows:

- Tree 1: EXERCISE=rarely→ RISK=high
- Tree 2: SMOKER=false→ RISK=low
- Tree 3: OBESE=true→ RISK=high

So, the majority vote is for RISK=high

Algorithm 8.4 The random forests algorithm.

1. Given a training set $(\boldsymbol{x}_i, z_i)$, $i = 1, \ldots, n$, of patterns $\boldsymbol{x}_i$ and labels z_i. Specify the number of trees in the forest, B, and the number of random features to select, m.
2. For $b = 1, \ldots, B$,
   (a) Generate a bootstrap sample of size n by sampling with replacement from the training set; some patterns will be replicated, others will be omitted.
   (b) Design a decision tree classifier, $\eta_b(\boldsymbol{x})$ using the bootstrap sample as training data, randomly selecting at each node in the tree m variables to consider for splitting.
   (c) Classify the nonbootstrap patterns (the 'out-of-bag' data) using the classifier $\eta_b(\boldsymbol{x})$.
3. Assign $\boldsymbol{x}_i$ to the class most represented by the classifiers $\eta_{b'}(\boldsymbol{x})$, where b' refers to the bootstrap samples that do not contain $\boldsymbol{x}_i$.

Andrew R.Webb , Keith D. Copsey, Statistical Pattern Recognition, 3rd ed. 2011

Summary

- Decision trees are simple and interpretable models for regression and classification.
- However they are often not competitive with other methods in terms of prediction accuracy.
- Bagging and random forests are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the prediction of the resulting ensemble of trees.
- Random forests is one of the state-of-the-art methods for supervised learning. However results can be difficult to interpret.

Additional Tutorial (StatQuest)

Decision tree:
https://www.youtube.com/watch?v=J4Wdy0Wc_xQ

Random forest:
Part I https://www.youtube.com/watch?v=J4Wdy0Wc_xQ&t=123s
Part II https://www.youtube.com/watch?v=sQ870aTKqiM

AdaBoost:
https://www.youtube.com/watch?v=LsK-xG1cLYA

Gradient Boost
Part I https://www.youtube.com/watch?v=3CC4N4z3GJc&t=50s
Part II https://www.youtube.com/watch?v=2xudPOBz-vs
Part III https://www.youtube.com/watch?v=jxuNLH5dXCs

Homework 13-1

Finish the tree splitting of Example 2.

Deadline of Homework #13: 2022/12/26 3:30pm

Homework 13-2

Consider the following $n = 16$ points in two dimensions, training a binary tree using the entropy impurity.

1. Plot the points of $\boldsymbol{\omega_1}$ and points of $\boldsymbol{\omega_2}$ in the 2D x_1-x_2 plane.
2. Provide the step-by-step split feature Table similar to Example 1.
3. Illustrate the recursive binary splitting on the 2D x_1-x_2 plane using Lecture 13_2 decision_tree.jpynb

$\boldsymbol{\omega_1}$ **(black)**		$\boldsymbol{\omega_2}$ **(red)**	
x_1	x_2	x_1	x_2
.15	.83	.10	.29
.09	.55	.08	.15
.29	.35	.23	.16
.38	.70	.70	.19
.52	.48	.62	.47
.57	.73	.91	.27
.73	.75	.65	.90
.47	.06	.75	.36

Deadline of Homework #13: 2022/12/26 3:30pm

Homework 13-3

Study the code Lecture 13_4 Combining different models for ensemble learning.jpynb by yourself. You will need to use the ensemble method for testing data in the final exam.