Lecture 14

Decision tree

Decision tree

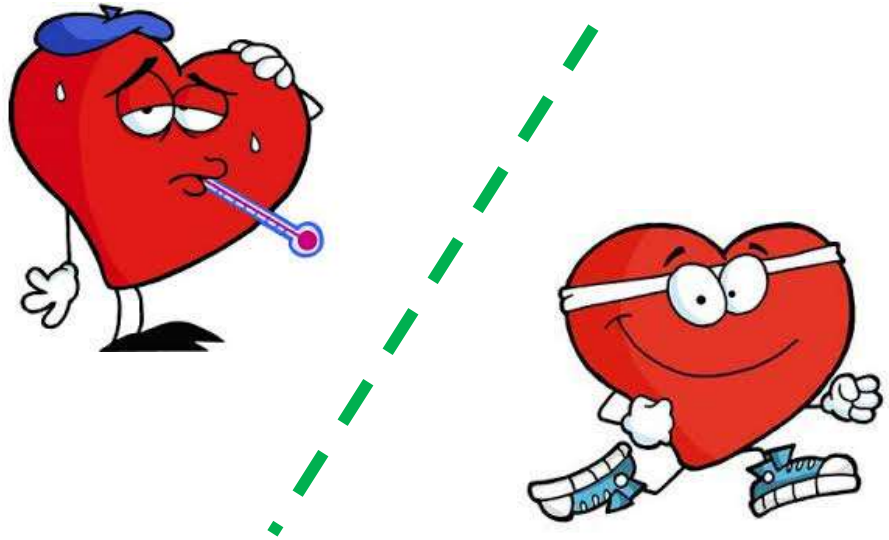
- Decision tree is a simple model for classification or regression.
- Its interpretation is easy.
- It is very common in some communities like health or biostatistics.



Example

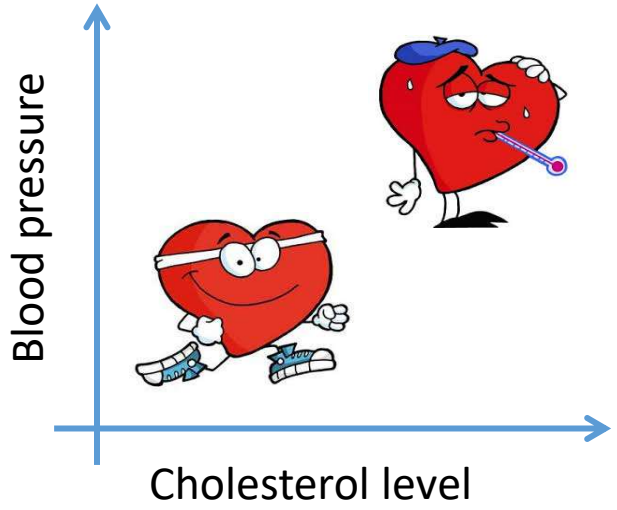
- Lets begin with an example!

- Consider we want to classify people who have heart disease vs. who do not have.



Example

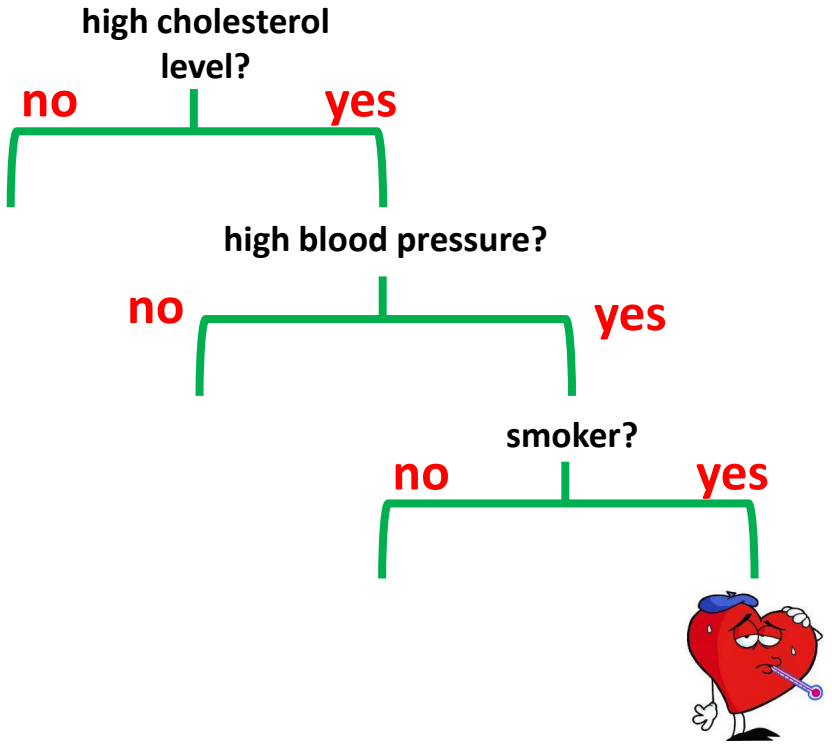
- For this purpose consider that we have some measures.
- We can describe each patient by a number of features.
  - Cholesterol level
  - Blood pressure
  - Weight
  - …
- We can represent the state of each patient by a vector.



$$\begin{bmatrix} CL \\ BP \\ W \\ \vdots \end{bmatrix}$$

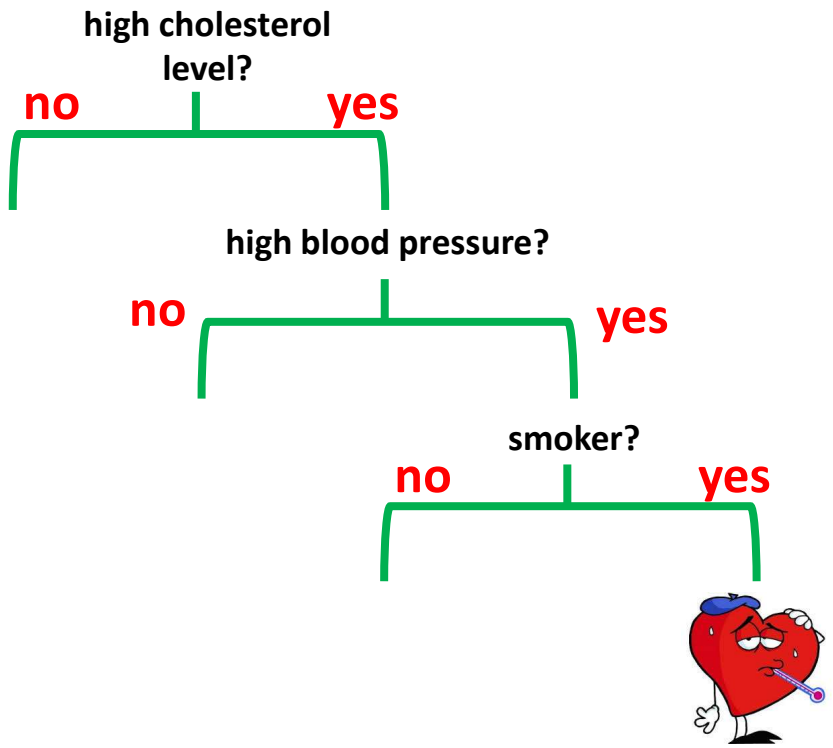
Example

- A decision tree classifier can categories the condition of a person based on every measure.
  - **Does he have high cholesterol level?**
  - **Does he have blood pressure?**
  - **Is he a smoker?**



Example

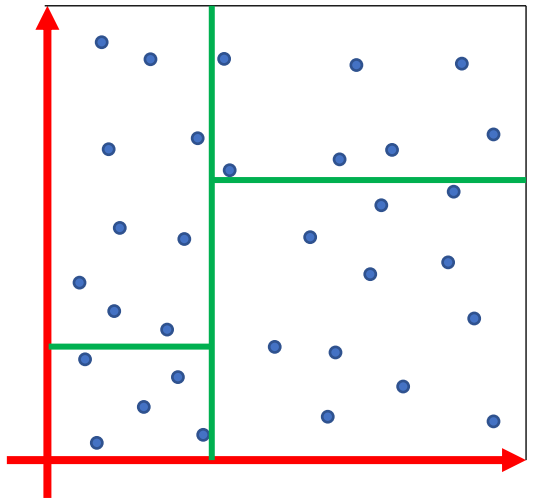
- This is the way that we treat many problems personally.
- Decision tree is a model which divides the whole feature space into two parts in each step and at the end it decides about the class of each region.
- One of the features of this model is its interpretability.



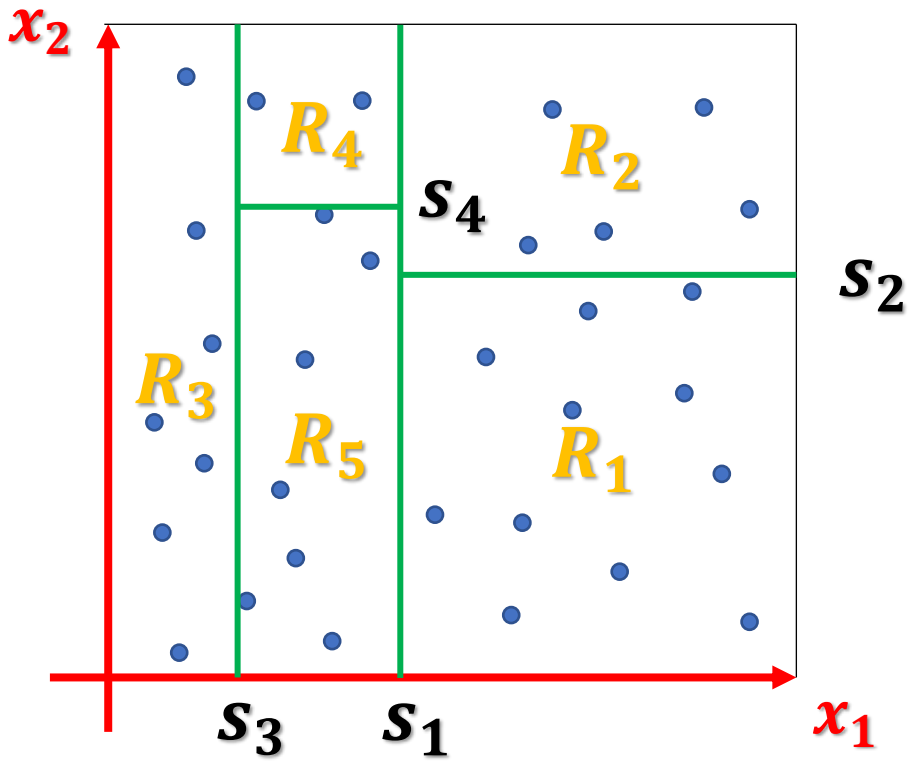
Decision tree

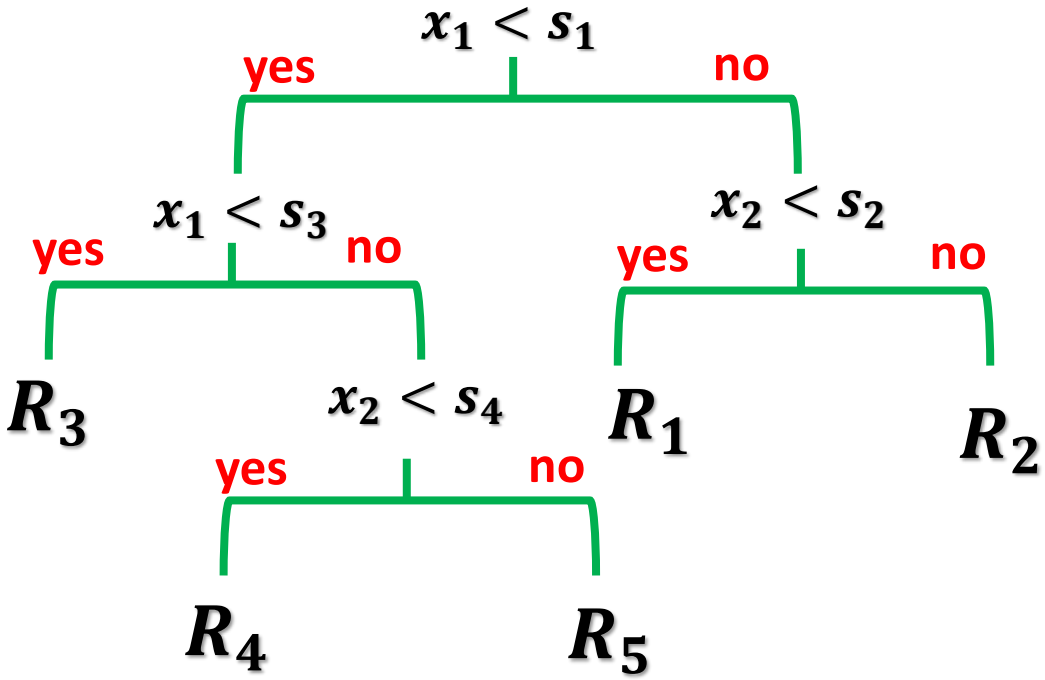
- The idea behind tree is splitting the feature space into set of rectangles.

- Fit a simple model to each rectangle (e.g. constant, linear, …)



Example


x_2
R_4
R_2
s_4
s_2
R_3
R_5
R_1
s_3
s_1
x_1


$x_1 < s_1$
yes
no
$x_1 < s_3$
$x_2 < s_2$
yes
no
yes
no
R_3
$x_2 < s_4$
R_1
R_2
yes
no
R_4
R_5

- Consider the data :

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{n} \qquad \mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]_{d \times n} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1}$$

Define:

$\mathbf{x}_{j:} \rightarrow j$'th row and all columns. (each row of $\mathbf{x}$ matrix is a feature)

$\mathbf{x}_{:i}$ one column and all rows. (each column of $\mathbf{x}$ matrix is a data sample)

- At each split we have two regions.

$$R_1(j, s) = \{\mathbf{x}_{:i} | \mathbf{x}_{j:} \leq s\}$$
$$R_2(j, s) = \{\mathbf{x}_{:i} | \mathbf{x}_{j:} > s\}$$

- We need to decide about j and s.

- To decide about these parameters we can define an objective function.

Regression Case

- For simplicity, consider the model we want to fit for each region be a constant value.
- Therefore in region one for example, we want all the target values to be as close as possible to constant C_1.

$$\min_{C_1} \sum_{y_i \in R_1} (y_i - C_1)^2$$

- Since in each split we have two regions, we can write similar objective for region two.

$$\min_{C_2} \sum_{y_i \in R_2} (y_i - C_2)^2$$

- Therefore we have:

$$\min_{C_1} \sum_{y_i \in R_1} (y_i - C_1)^2 + \min_{C_2} \sum_{y_i \in R_2} (y_i - C_2)^2$$

- Finally we should also decide about j and s:

$$\min_{j,s} \left[\min_{C_1} \sum_{y_i \in R_1} (y_i - C_1)^2 + \min_{C_2} \sum_{y_i \in R_2} (y_i - C_2)^2 \right]$$

- In each split, this optimization problem must be solved.

$$\min_{j,s}\left[\min_{C_1}\sum_{y_i\in R_1}(y_i-C_1)^2+\min_{C_2}\sum_{y_i\in R_2}(y_i-C_2)^2\right]$$

?

- If j and s are known. C_1 and C_2 would be:

$$C_1 = average\{y_i | y_i \in R_1\}$$
$$C_2 = average\{y_i | y_i \in R_2\}$$

- How many choices we have for splitting if we know about feature j?

- Given j, we have only n possible choice for s.

$$\mathbf{x}_j = [x_{j1}, x_{j2}, \ldots, x_{jn}]$$

- One solution for this problem can be a brute force search.

1. Consider feature j each time
2. Test all n possible choices for s
3. Find C_1 and C_2 for given s and j
4. Repeat until finding minimum value for objective function

This is a trivial solution and we will see a better solution later

But we can build a tree any way

Define:

$$Q_m(T) = \frac{1}{n_m} \sum_{y_i \in R_m} (y_i - C_m)^2$$

where:

- m is a terminal node of the tree.
- R_m is a region.
- $n_m$is the number of points in R_m.
- C_m is the constant that we fit in R_m.
- $|T|$ is the number of terminal nodes.

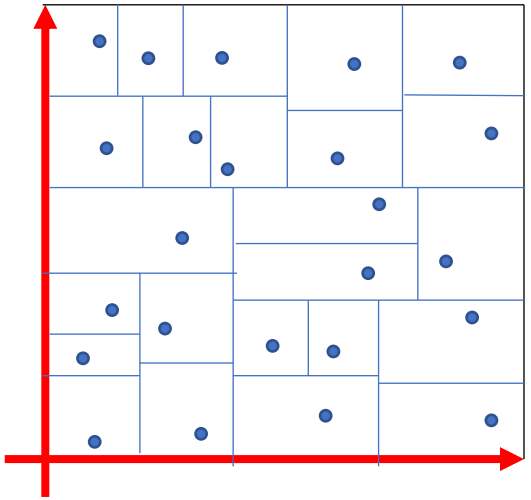
- Therefore the total error would be:

$$\sum_{m=1}^{|T|} n_m Q_m$$

- When should we stop learning?

- It can be observe as we grow the tree the error will get smaller.

- Like any other methods, decision tree model can overfit.



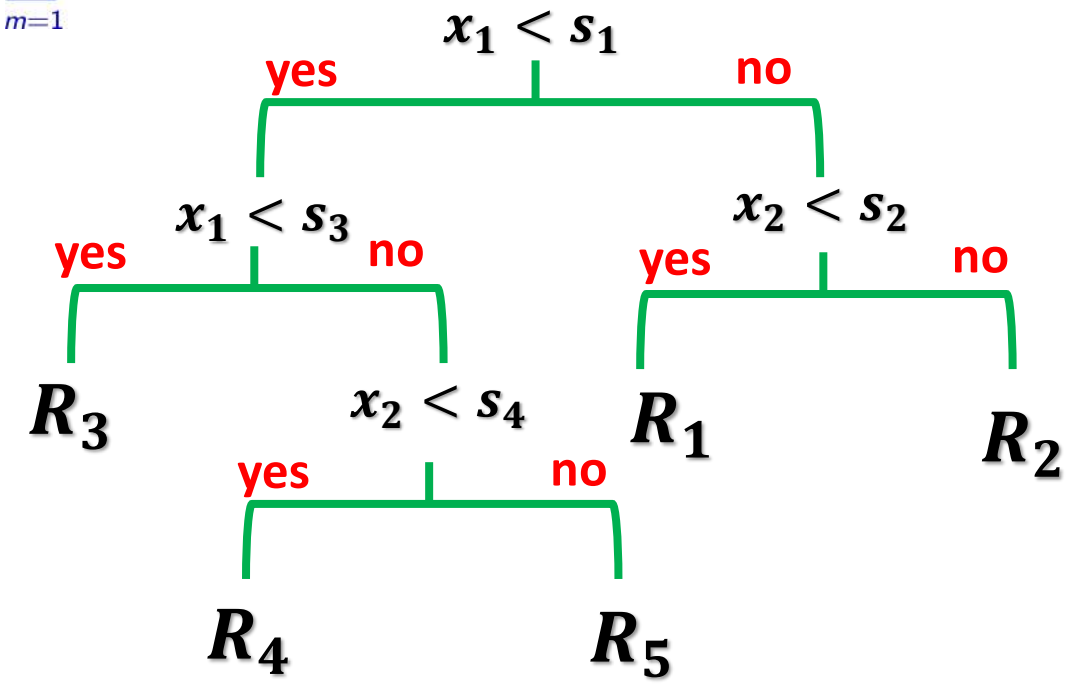
- How can we avoid overfitting?
  - Cross validation
  - Split until the error reduction is more than a threshold. (which is a bad idea!)
  - pruning

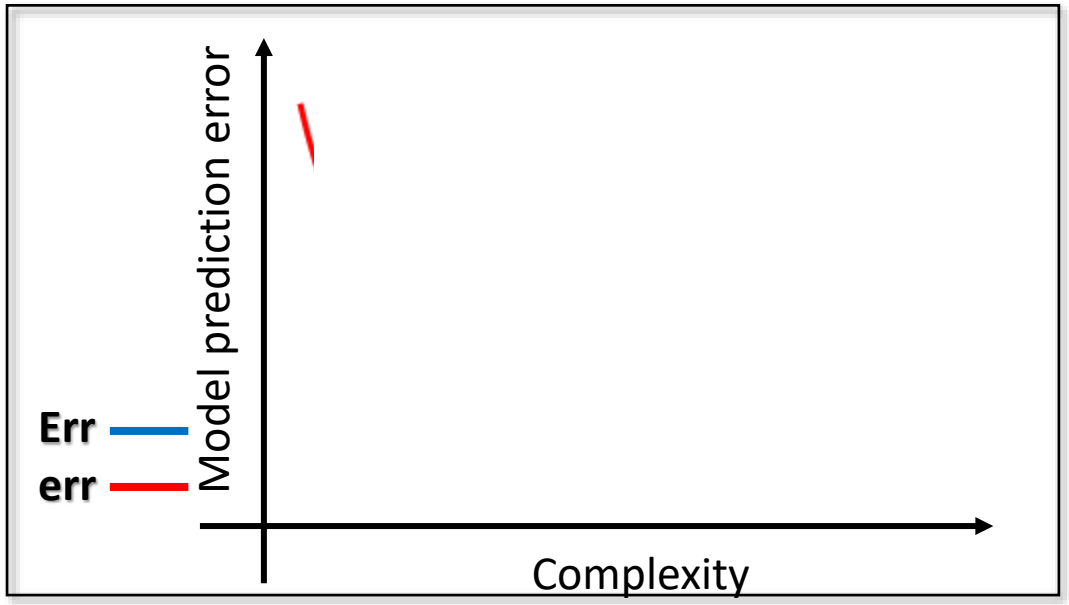
Pruning

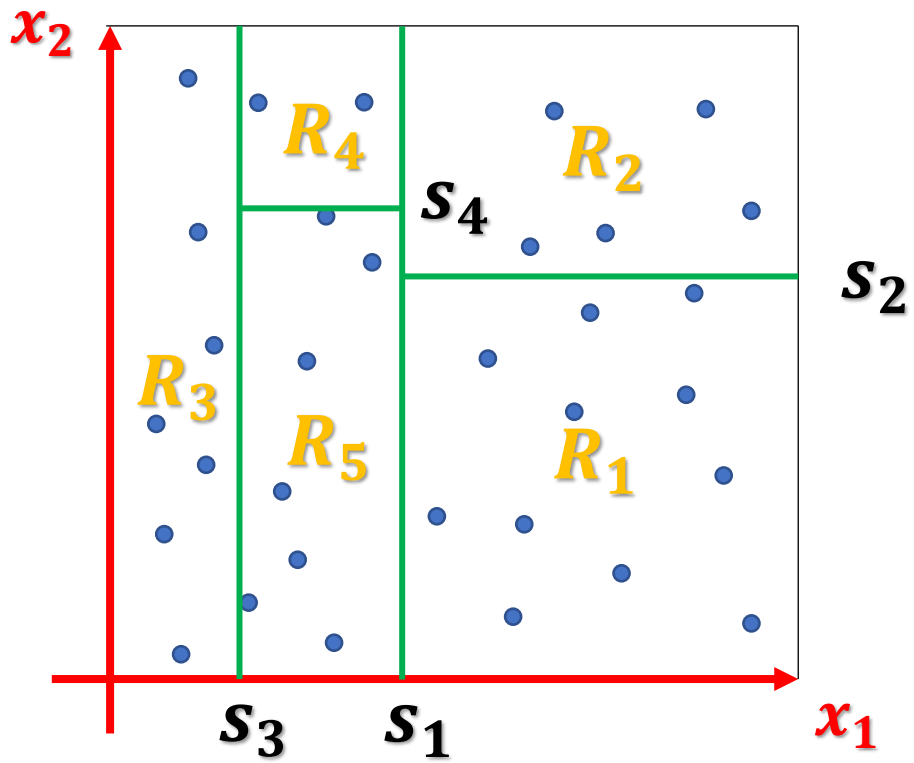
Suppose that we make a tree T_0 by minimizing the cost function.

$$Q_m(T) = \frac{1}{n_m} \sum_{y_i \in R_m} (y_i - C_m)^2$$

$$\sum_{m=1}^{|T|} n_m Q_m$$

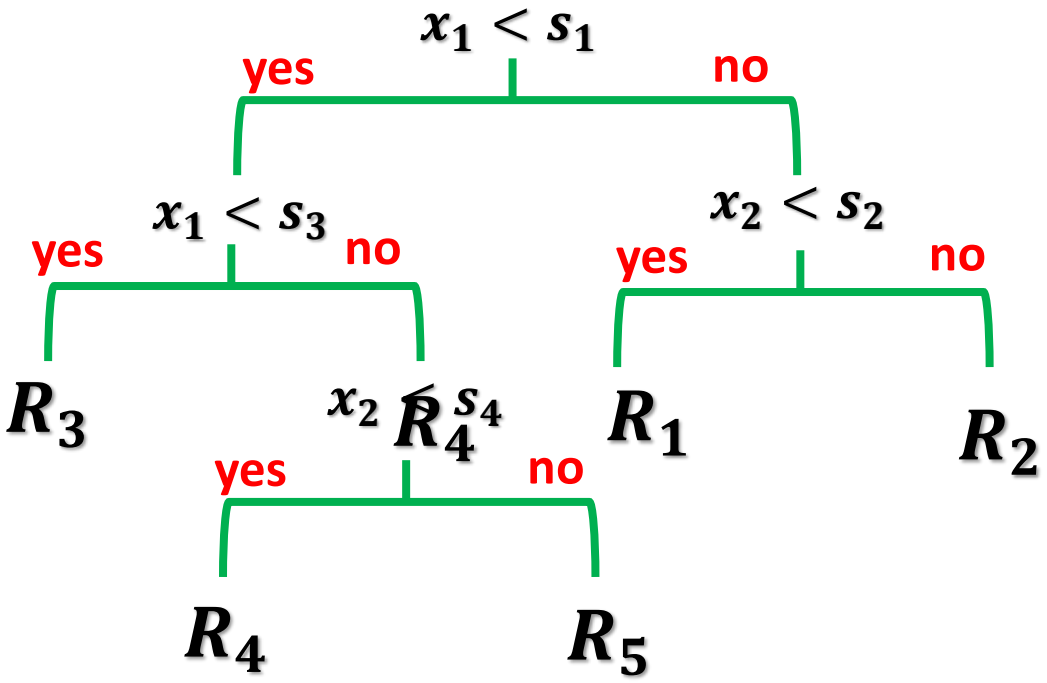


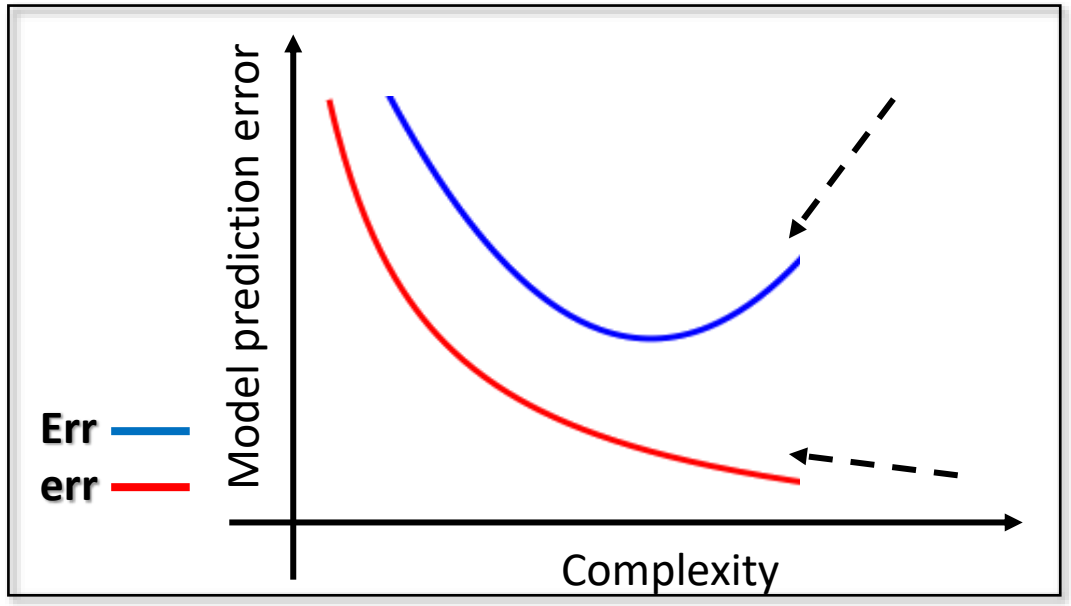




Pruning

- Then we can start to merge the regions in order to make the tree smaller which causes the training error to increase.



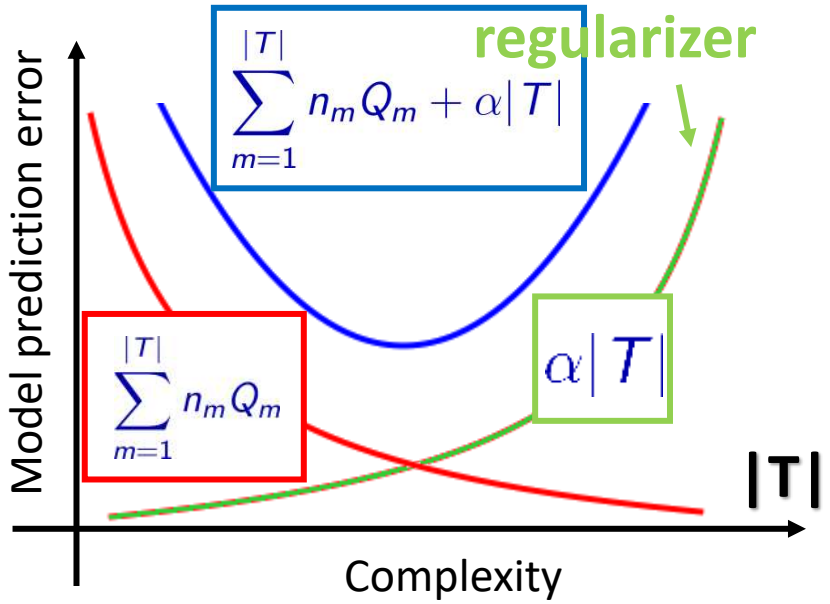


Pruning

- We can add a regularization term to the objective function in order to control the complexity of model.

$$\sum_{m=1}^{|T|} n_m Q_m + \alpha |T|$$

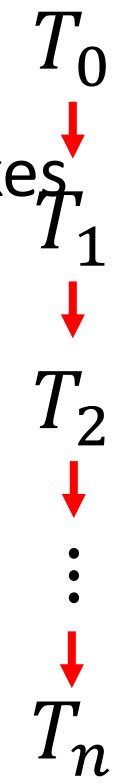
regularizer



Pruning

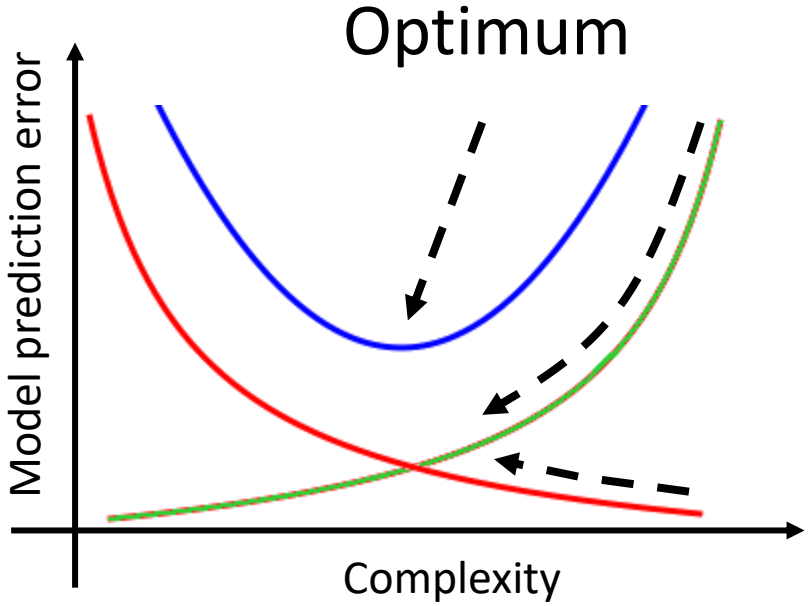
Algorithm

1. Build tree T_1 based on tree T_0 by collapsing nodes that makes smallest per-node increase.
2. Tree T_2 can be build based on tree T_1.
3. and so on.



Pruning

Based on this paradigm, the main cost function increases and the regularization cost decreases in each step until to reach an optimum point.



Classification

- In regression case, we tried to fit a constant to a region.

$$Q_m(T) = \frac{1}{n_m} \sum_{y_i \in R_m} (y_i - C_m)^2$$

- But in classification case we need to find the regions which contain samples from same class.

Change

- We can change regression cost with some sort of misclassification cost.

Node impurity measures

- Consider the classes of $C = \{1, 2, 3, \ldots, k\}$
- Impurity:

$$P_{mk} = \frac{1}{n_m} \sum_{y_i \in R_m} I(y_i = k)$$

- Misclassification:

$$\frac{1}{n_m} \sum_{y_i \in R_m} I(y_i \neq k) = 1 - P_{mk}$$

- Gini Index:

$$\sum_{i \neq j} P_{mi} P_{mj}$$

- Cross Entropy

$$-\sum_{k=1}^{K} P_{mk} log(P_{mk})$$

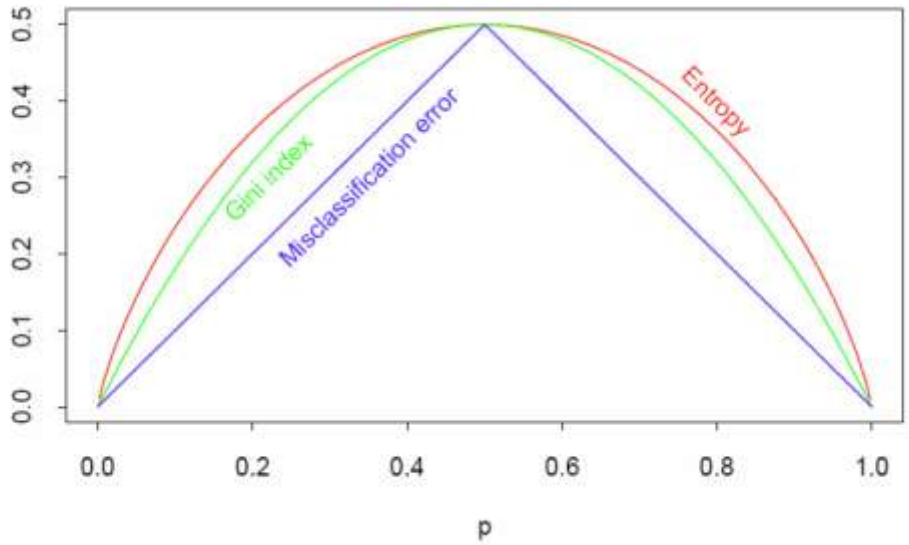


Figure 9.3: *Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through* $(0.5, 0.5)$.

Credit: elements of statistical learning

Main decision tree algorithms

Algorithm	What makes it different?
ID3	Uses information gain (entropy) to choose splits
C4.5	Improves ID3 using gain ratio, continuous features, missing values, and pruning
C5.0	A more efficient successor to C4.5, with additional capabilities
CART	Uses Gini impurity for classification by default and creates binary splits

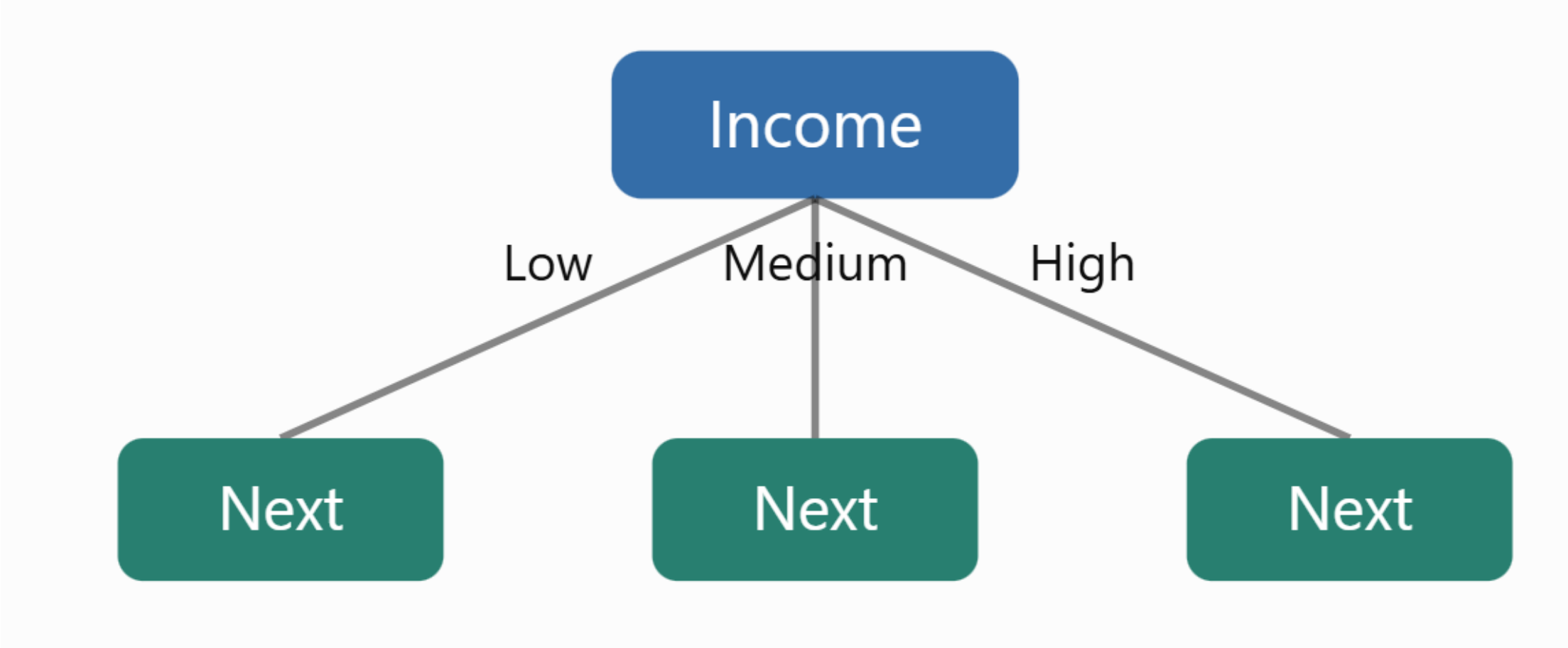
CART stands for *Classification and Regression Trees*. Unlike the original ID3 and C4.5 algorithms, it directly supports regression trees.

Example: How they construct different trees

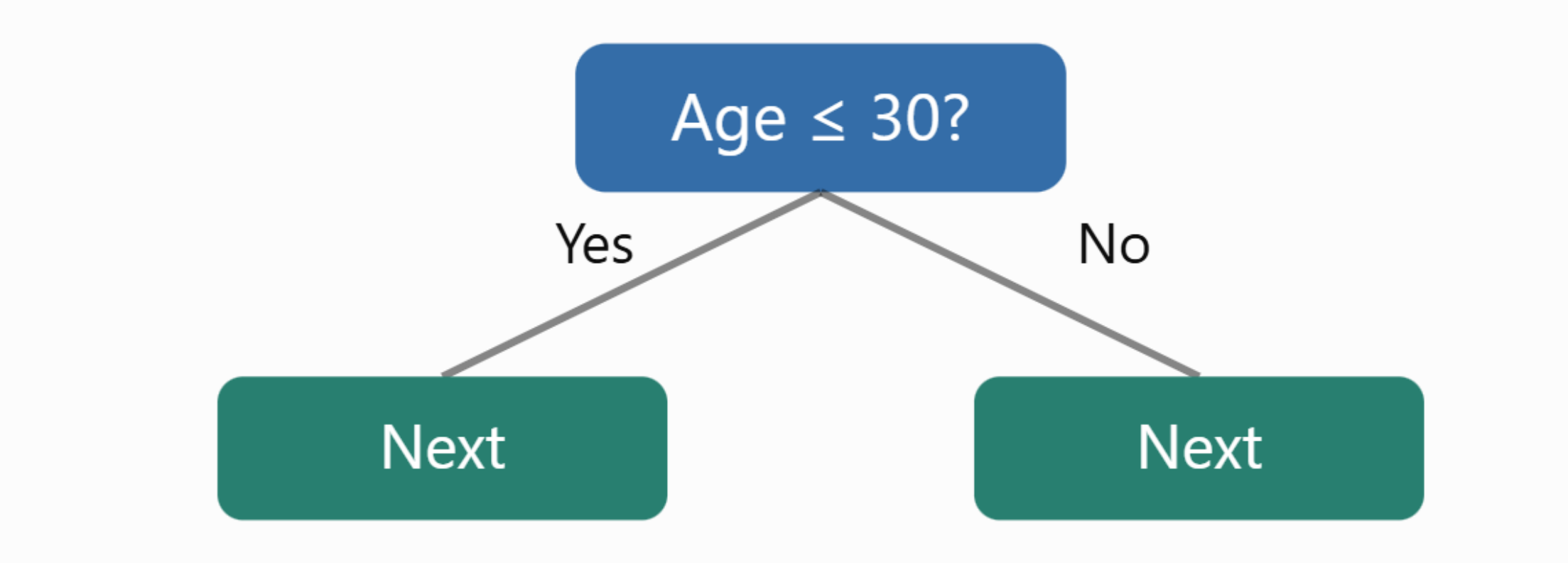
Suppose you're predicting whether someone buys a laptop, using:

- Age: 22, 35, 48...
- Income: Low, Medium, High
- Student: Yes or No

ID3



CART



- Here's a useful practical connection:
- Scikit-learn's DecisionTreeClassifier and RandomForestClassifier use an optimized CART-style tree algorithm.
- GitHub (scikit-learn/doc/modules/tree.rst at main · scikit-learn/scikit-learn)

when you write:

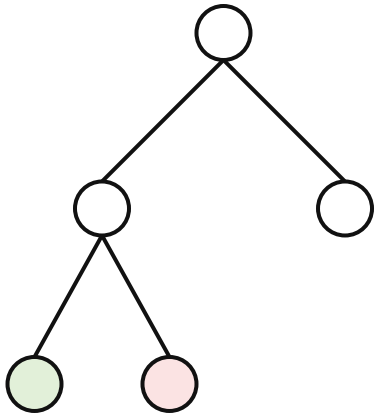
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

tree = DecisionTreeClassifier()
forest = RandomForestClassifier(n_estimators=100)
```

You're effectively choosing between one CART-style tree and an ensemble of 100 CART-style trees, with additional randomness in the forest.

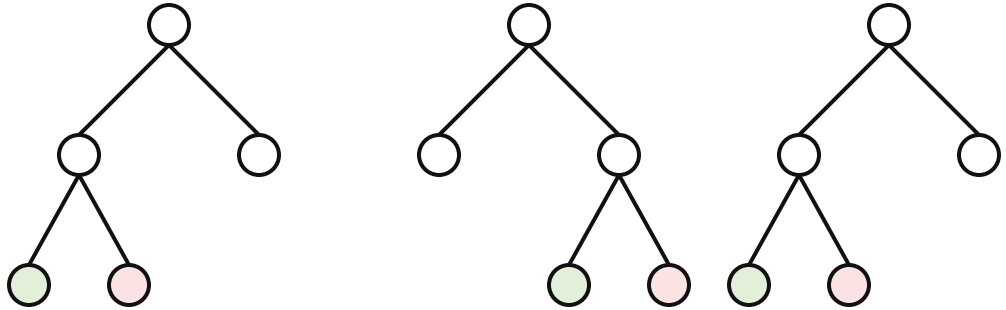
A forest is many decision trees

One tree makes a prediction.



One prediction

Many trees combine their predictions.

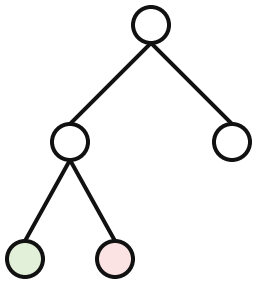


One combined prediction

Classification by majority vote

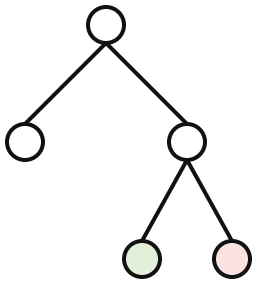
Will this customer buy the product?

Tree 1



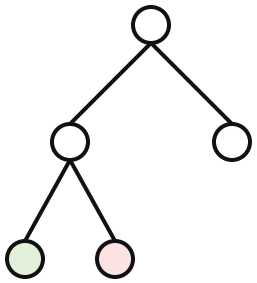
Yes

Tree 2



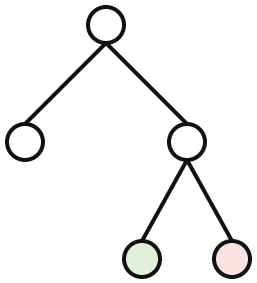
Yes

Tree 3



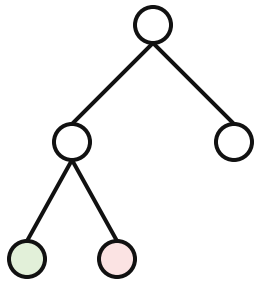
No

Tree 4



Yes

Tree 5

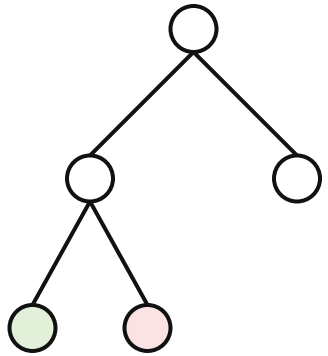


No

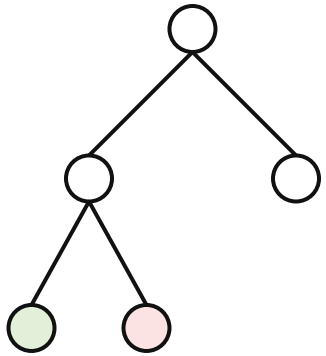
Final prediction Yes (3 of 5 votes)

Why do we need different trees?

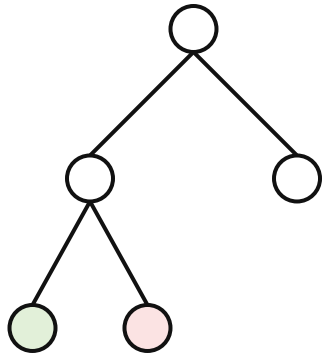
Same data + same deterministic training procedure



Same prediction



Same prediction



Same prediction

Identical trees make the same mistakes.

Random observations for each tree

$X \in \mathbb{R}^{dxn}$ Rows = features Columns = observations

Original data (d = 5, n = 6)

	1	2	3	4	5	6
Age	22	35	41	54	46	29
Income	35	60	75	90	65	45
Education	2	3	3	4	2	3
Location	1	2	1	3	2	1
Credit	620	710	680	760	690	650

Training data for tree 1

	1	3	3	6	2	6
Age	22	41	41	29	35	29
Income	35	75	75	45	60	45
Education	2	3	3	3	3	3
Location	1	1	1	1	2	1
Credit	620	680	680	650	710	650

Draw n columns with replacement. Repeated columns are allowed.

Random candidate features at a split

At the root of tree 1, randomly choose candidate rows.

	1	3	3	6	2	6
Age	22	41	41	29	35	29
Income	35	75	75	45	60	45
Education	2	3	3	3	3	3
Location	1	1	1	1	2	1
Credit	620	680	680	650	710	650

Candidate features

Age and Income

Compare their possible splits.

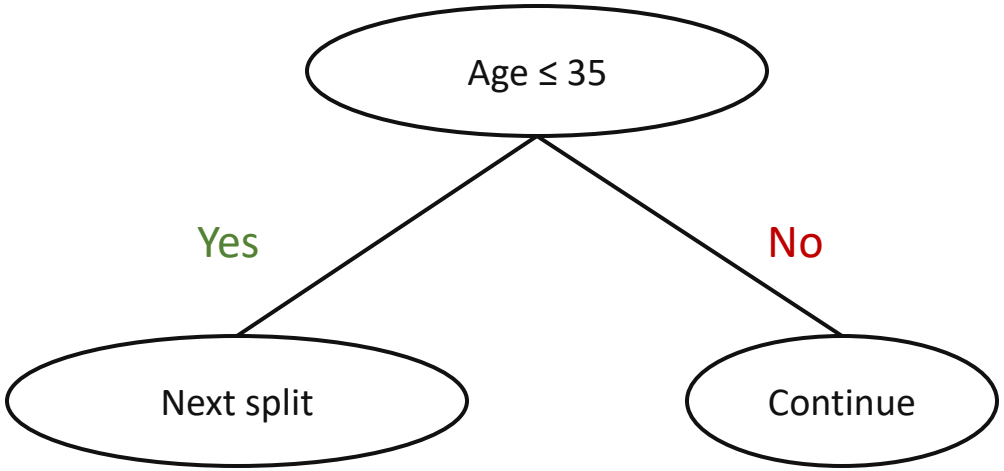
Choose the split with the largest impurity reduction.

Example selected split Age ≤ 35

Fresh features at each node

Observations reaching the left child

	1	6	2	6
Age	22	29	35	29
Income	35	45	60	45
Education	2	3	3	3
Location	1	1	2	1
Credit	620	650	710	650



New candidates

Income and Credit

Repeat at every node. Each tree also gets its own bootstrap sample.

Why combining trees helps

A deep tree can fit noise in its training sample.

Different samples and feature candidates create diverse trees.

Combining their predictions usually reduces variance
and improves performance on unseen data.

Trade-off

More computation and less interpretability than one tree.