

INSA ROUEN NORMANDIE

Machine Learning

Decision Trees and Random Forests

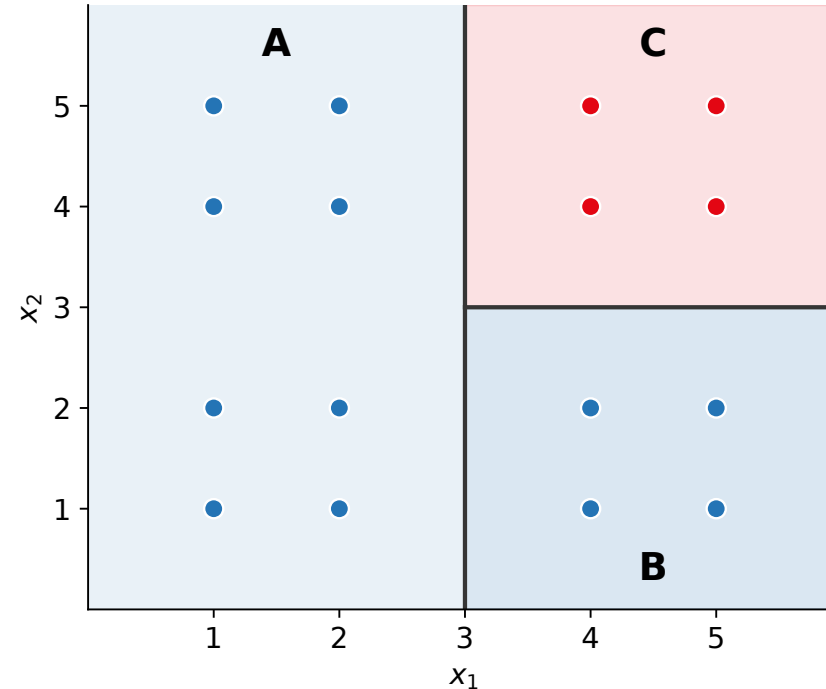
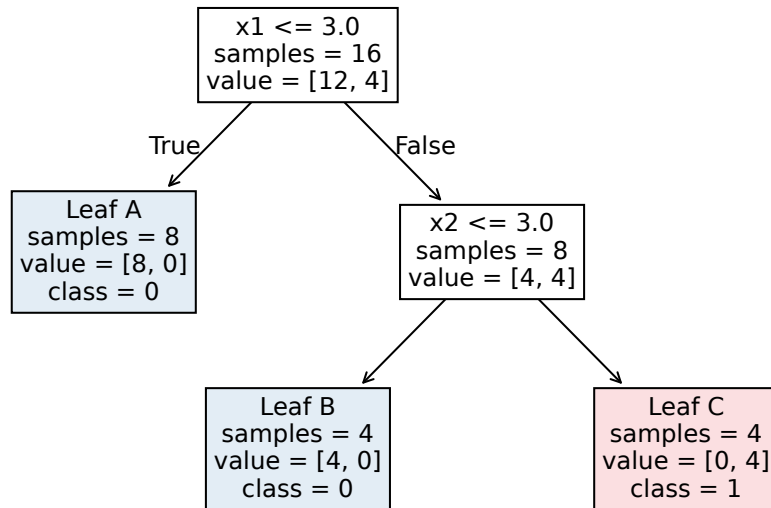
ITI 4 · Benoit Gaüzère

Decision Trees and Random Forests

Contents

1. Predict with a tree: tests, regions and leaves.
2. Learn a tree: candidate splits and impurity reduction.
3. Control its complexity.
4. Combine randomized trees into a forest.

A decision tree is a sequence of decisions



Same model, two views: decision rules and input-space regions.

Split on x_1 , then on x_2 : leaves **A**, **B** predict class 0; **C** predicts class 1.

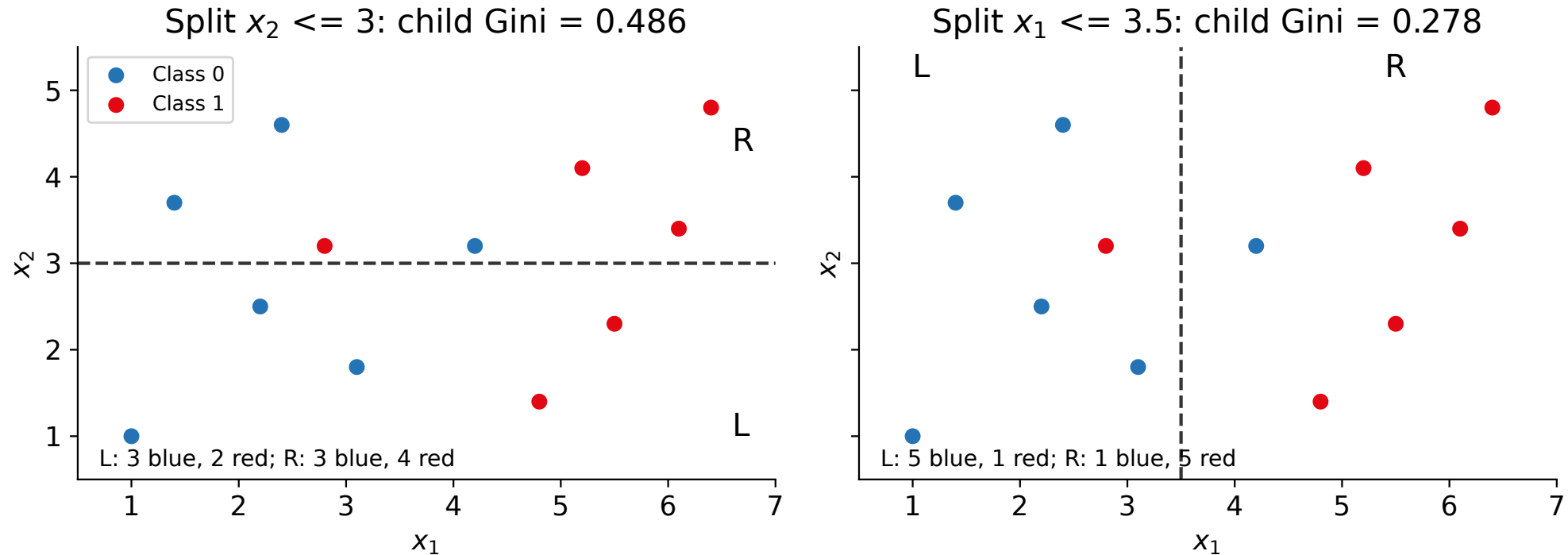
What does a leaf predict?

A leaf summarizes the training observations that reach it.

Task	Leaf prediction
Classification	Class proportions $p_k = n_k/n$; predict the largest one
Regression, squared loss	Mean target in the leaf
Regression, absolute loss	Median target in the leaf

Example: 3 class-0 and 1 class-1 observations give $\hat{p} = (0.75, 0.25)$ and predicted class 0.

A useful split makes the children less mixed



Same observations, two candidate cuts. Which creates more homogeneous groups?

Impurity measures class mixing inside a group.

Learning one split

At node t , choose a variable j and a threshold s .

1. Sort the distinct observed values of each candidate variable.
2. Consider thresholds between consecutive values.
3. Reject splits that violate constraints, such as `min_samples_leaf`.
4. Compare the remaining splits.

Each candidate creates **two child groups**:

$$L = \{i \in t : x_{ij} \leq s\}, \quad R = \{i \in t : x_{ij} > s\}.$$

L = left child; R = right child. Next: score how mixed they are.

Score a split by its children

Impurity $I(t)$ measures how mixed the labels are in a node.

A pure node contains one class; a mixed node contains several classes.

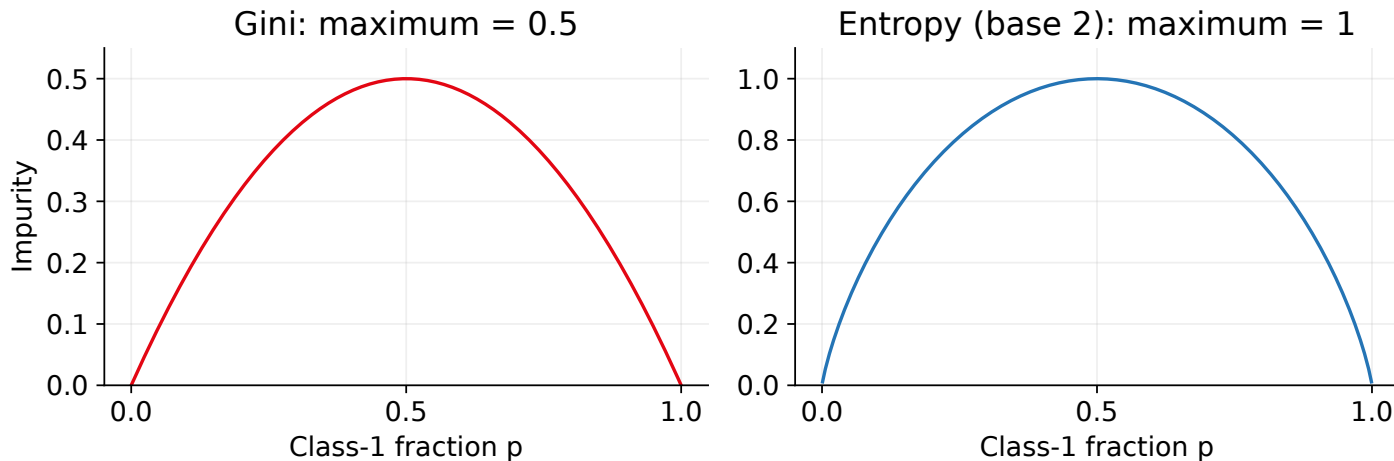
For a split into children L, R , with $n_t = n_L + n_R$:

$$Q(j, s) = \frac{n_L}{n_t} I(L) + \frac{n_R}{n_t} I(R), \quad \Delta I = I(t) - Q(j, s).$$

- Minimize the **average impurity after splitting**, Q .
- Equivalently, maximize the **impurity decrease**, ΔI .
- Weight by child size: a tiny pure child is not enough.

Classification impurity: how mixed is a node?

$$I_{\text{Gini}} = 1 - \sum_k p_k^2, \quad I_{\text{entropy}} = - \sum_k p_k \log_2 p_k.$$



Binary example: $(p_0, p_1) = (1, 0)$ gives impurity **0**.

A 50/50 mix gives **Gini 0.5**, but **entropy 1** (base 2).

The maximum depends on the criterion; impurity is not an error rate.

Classification criteria in scikit-learn

```
DecisionTreeClassifier(criterion=...)
```

Value	Quantity used to select the split
"gini" (default)	Gini impurity decrease
"entropy"	Shannon entropy decrease / information gain
"log_loss"	Same Shannon entropy criterion

- Gini and entropy can rank candidate splits differently.
- `entropy` and `log_loss` name the same split criterion.
- Classification error itself is too insensitive to many useful changes.

Why not minimize classification errors alone?

Suppose a node contains **100 observations: 80 class 1, 20 class 0**.

A candidate split isolates 60 class-1 observations:

Group	Class 1 / class 0	Majority-class errors	Gini
Parent	80 / 20	20	0.32
Left child	60 / 0	0	0
Right child	20 / 20	20	0.5

Errors stay at 20/100, so error alone sees no improvement.

But weighted Gini drops from **0.32 to 0.20**:

$$Q = (60/100) \times 0 + (40/100) \times 0.5 = 0.20.$$

The split has isolated a large **pure group**.

A root split we can calculate

Eight observations, sorted by their measured sensor signal (arbitrary units):

Signal x	12	19	31	44	58	73	91	110
y	0	0	1	0	1	1	1	1

The root contains 3 zeros and 5 ones:

$$I(t) = 1 - (3/8)^2 - (5/8)^2 = 0.46875.$$

For $x \leq 51$: left labels (0, 0, 1, 0); right labels (1, 1, 1, 1).

Check-up 1: which candidate is better?

Useful formulas: $I = 1 - \sum_k p_k^2$, $Q = (n_L/n_t)I(L) + (n_R/n_t)I(R)$, and $\Delta I = I(t) - Q$.

The root impurity is 0.46875. Compare:

Threshold	Left labels	Right labels
$s = 25$	(0, 0)	(1, 0, 1, 1, 1, 1)
$s = 51$	(0, 0, 1, 0)	(1, 1, 1, 1)

Compute each child's Gini, then Q and the gain. Which split is better?

Check-up 1: answer

Threshold	$I(L)$	$I(R)$	Q	Gain $I(t) - Q$
25	0	$5/18 \simeq 0.27778$	0.20833	0.26042
51	0.375	0	0.1875	0.28125

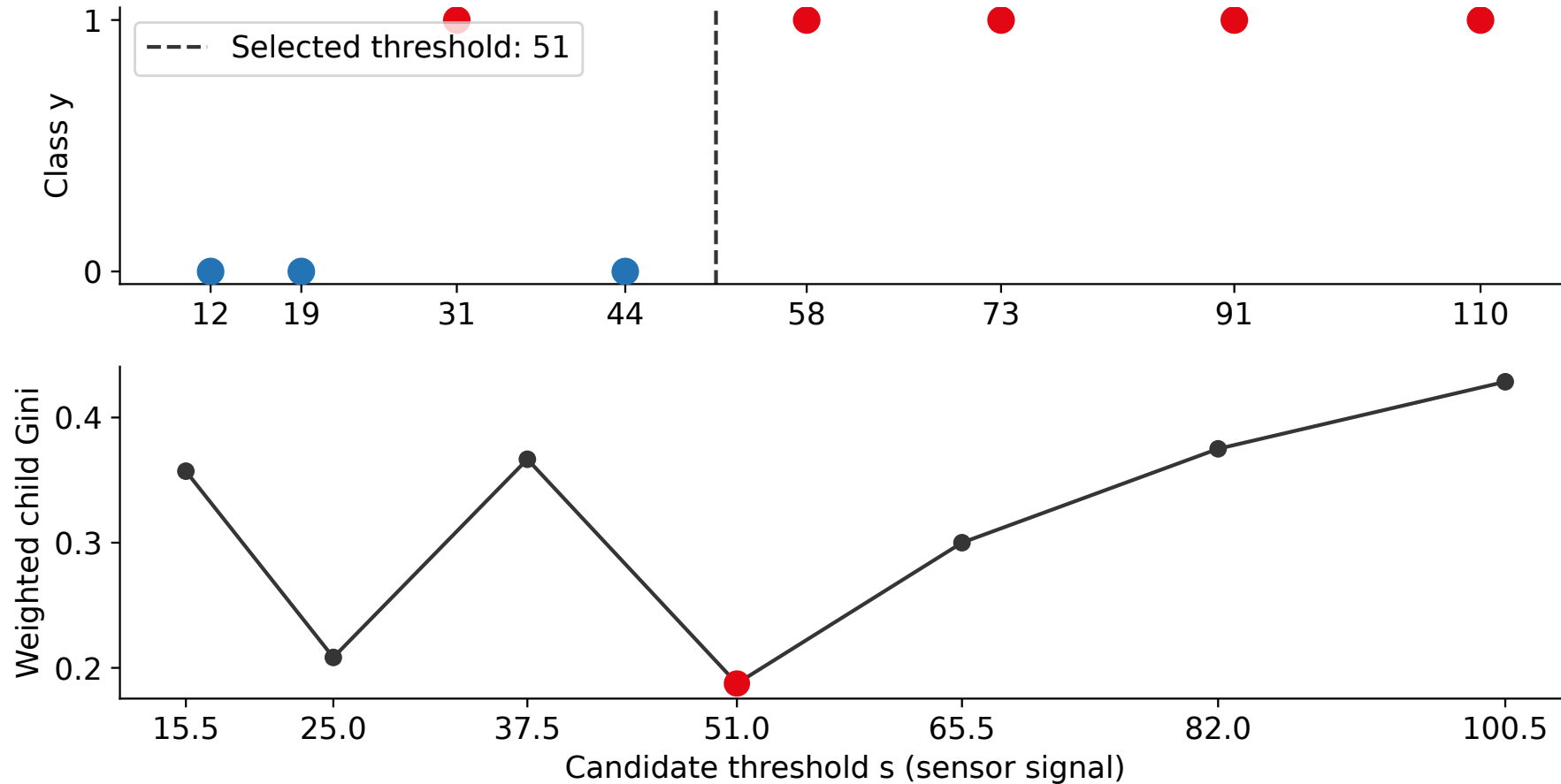
For $s = 25$: $I(R) = 1 - (5/6)^2 - (1/6)^2 = 5/18$.

$$Q_{25} = \frac{2}{8} \times 0 + \frac{6}{8} \times \frac{5}{18} = 0.20833,$$

$$Q_{51} = \frac{4}{8} \times 0.375 + \frac{4}{8} \times 0 = 0.1875.$$

Choose **51** among these candidates.

Search all thresholds, then recurse



After selecting the split, repeat the same procedure **inside each child**.

Recursive tree-growing algorithm

Algorithm 1 · Grow a decision tree

Input: observations S , depth h · **Output:** a subtree

```
1 function Grow( $S, h$ )
2    $c \leftarrow \text{LeafPrediction}(S)$ 
3   if Stop( $S, h$ ) then return Leaf( $c$ )
4    $A \leftarrow \text{AdmissibleSplits}(S)$ 
5   if  $A$  is empty then return Leaf( $c$ )
6    $(j^*, s^*) \leftarrow \text{split in } A \text{ with minimum weighted child impurity}$ 
7    $(L, R) \leftarrow \text{Partition}(S, j^*, s^*)$ 
8    $T_L \leftarrow \text{Grow}(L, h + 1)$ 
9    $T_R \leftarrow \text{Grow}(R, h + 1)$ 
10  return Node( $j^*, s^*, T_L, T_R$ )
11 end function
```

Greedy: the best split is chosen locally; recursion builds its descendants.

Regression: replace class mixing by target dispersion

With squared loss, a leaf predicts its target mean $\bar{y}_t$:

$$I(t) = \frac{1}{n_t} \sum_{i \in t} (y_i - \bar{y}_t)^2.$$

For a candidate split, minimize

$$Q = \frac{n_L}{n_t} I(L) + \frac{n_R}{n_t} I(R) = \frac{\text{SSE}_L + \text{SSE}_R}{n_t}.$$

The same search mechanism now selects **variance reduction**.

Regression: a split reduces squared error

A node has target values (1, 3, 7, 9). Its mean is 5:

$$I(t) = \frac{(1 - 5)^2 + (3 - 5)^2 + (7 - 5)^2 + (9 - 5)^2}{4} = 10.$$

A candidate split produces:

Child	Targets	Leaf prediction	Impurity
Left	(1, 3)	Mean = 2	1
Right	(7, 9)	Mean = 8	1

$$Q = \frac{2}{4} \times 1 + \frac{2}{4} \times 1 = 1, \quad \Delta I = 10 - 1 = 9.$$

New observations receive **2 or 8**, depending on their leaf.

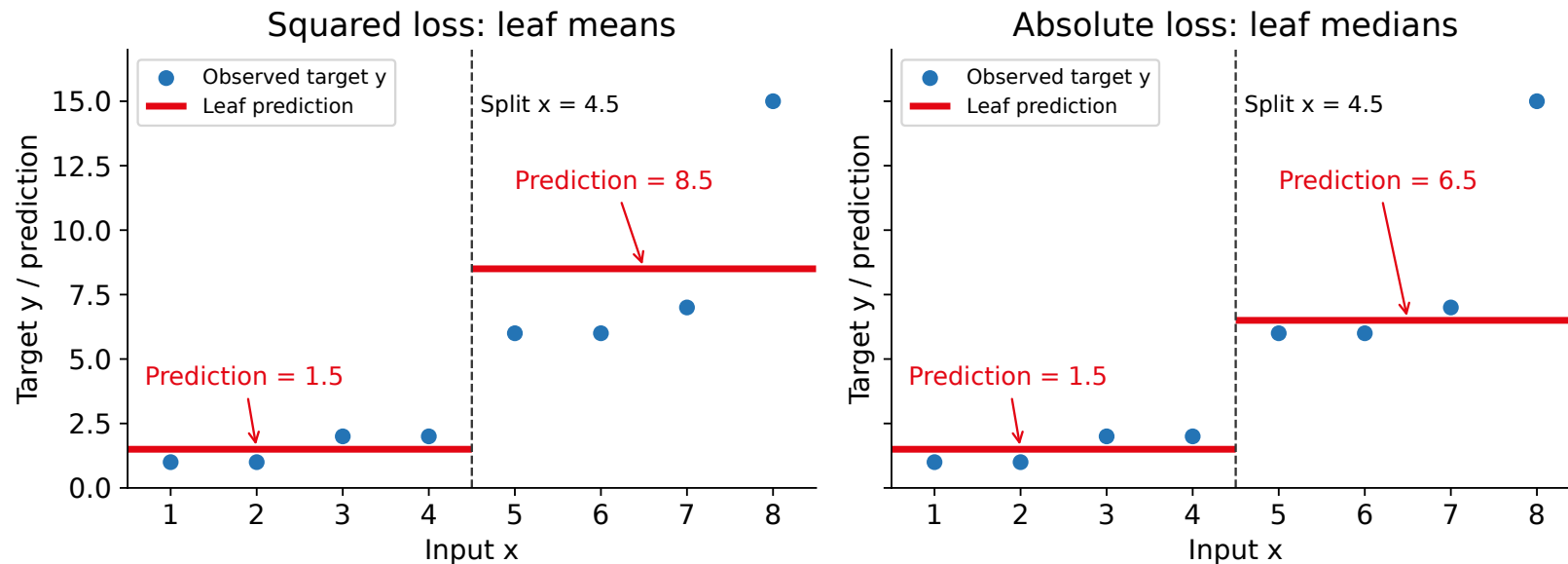
Regression criteria in scikit-learn

```
DecisionTreeRegressor(criterion=...)
```

Value	Split criterion	Leaf prediction
"squared_error" (default)	Mean squared error / variance	Mean
"friedman_mse"	MSE with Friedman's improvement score	Mean
"absolute_error"	Mean absolute deviation	Median
"poisson"	Half mean Poisson deviance reduction	Mean

Poisson example: predict the number of incidents per site per month, when exposure is comparable. It emphasizes a count-oriented deviance. Targets must be nonnegative, with a positive total sum; predictions may be non-integer.

Leaf means and medians produce different predictions



Blue dots: observed targets. **Red segments:** the value predicted by each leaf. The dashed line is the **fixed input split** $x = 4.5$.

At $x_{\text{new}} = 6.5$: the mean predicts **8.5**; the median predicts **6.5**.

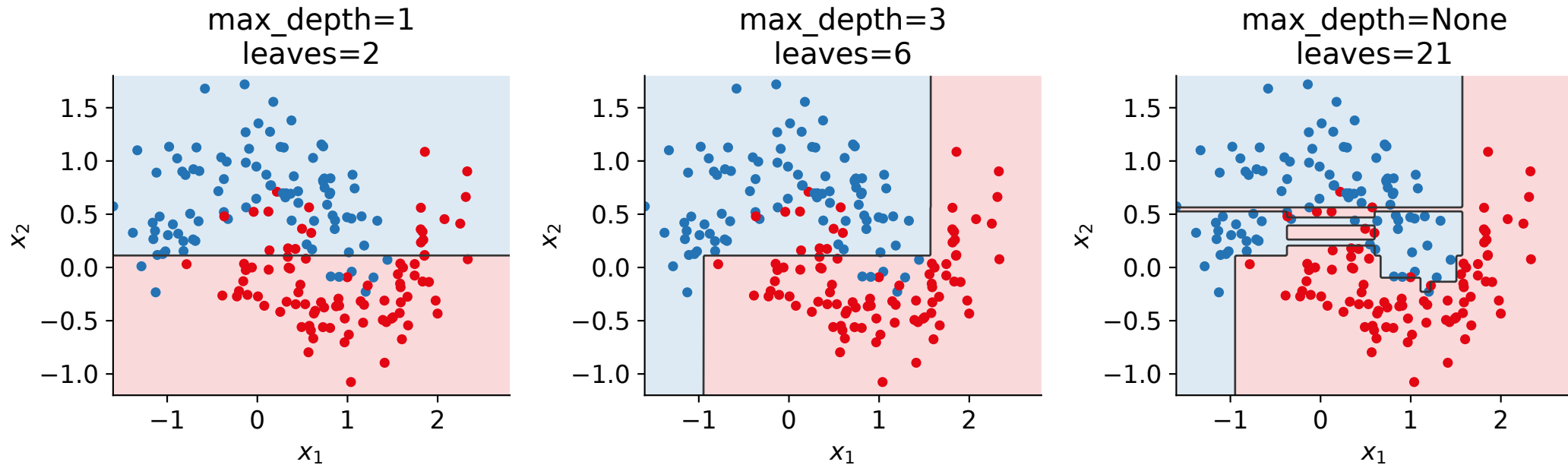
When does a tree stop growing?

With sklearn named parameters

Control	Effect
<code>max_depth</code>	Limit the longest root-to-leaf path
<code>min_samples_split</code>	Minimum node size before considering a split
<code>min_samples_leaf</code>	Minimum size of each resulting leaf
<code>max_leaf_nodes</code>	Limit the total number of leaves
<code>min_impurity_decrease</code>	Require a sufficiently large weighted gain

A pure node or a node with no admissible split also stops.

Depth controls the learned partition



Same training observations; only `max_depth` changes.

A more detailed training boundary is not automatically a better model.

Pruning after growing: cost-complexity

An alternative is to remove branches from a grown tree.

$$R_\alpha(T) = \underbrace{\sum_{\ell \in \text{leaves}(T)} \frac{n_\ell}{N} I(\ell)}_{\text{weighted leaf impurity}} + \alpha \underbrace{|\text{leaves}(T)|}_{\text{tree size}}.$$

- In scikit-learn: `ccp_alpha` controls this trade-off.
- Larger values favour smaller subtrees.
- Select it on development data, not on test performance.

Check-up 2: can this node split?

A node contains 6 observations. A candidate split gives children of sizes 1 and 5.

The model uses:

```
min_samples_split=6  
min_samples_leaf=2
```

1. Is the parent large enough to consider splitting?
2. Is this candidate split admissible?
3. What changes with `max_depth=1` ?

Check-up 2: answer

1. **Yes:** 6 observations satisfy `min_samples_split=6` .
2. **No:** the child of size 1 violates `min_samples_leaf=2` .
3. With `max_depth=1` , only the root may split; its children are leaves.

The algorithm must satisfy **all** applicable constraints.
An attractive impurity gain cannot override them.

Fit and inspect a tree

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

tree = DecisionTreeClassifier(
    criterion="gini", max_depth=3,
    min_samples_leaf=5, random_state=42,
)
tree.fit(X_dev, y_dev)
plot_tree(tree, feature_names=feature_names, filled=True)

probabilities = tree.predict_proba(X_new)
predictions = tree.predict(X_new)
```

The impurity criterion trains the tree; a validation metric evaluates it.

How a tree computes feature importance

For feature j , add the weighted gains of nodes that split on j :

$$a_j = \sum_{t: j(t)=j} \frac{n_t}{N} \Delta I_t, \quad \text{importance}_j = \frac{a_j}{\sum_r a_r}.$$

Feature	Weighted gains (illustrative)	Normalized importance
x_1	$0.18 + 0.06 = 0.24$	0.75
x_2	0.08	0.25

It measures **training impurity reduction attributed to a feature**, not accuracy gain or a causal effect.

From one tree to an ensemble

One deep tree is flexible, but an early split can change a large part of its prediction rule.

To build a useful ensemble:

1. Train **several useful trees**.
2. Make them **different** so they do not repeat the same errors.
3. Combine their outputs for each new observation.

Copying the same tree 200 times changes nothing.

First source of diversity: different resampled training observations.

Bootstrap: sample observations with replacement

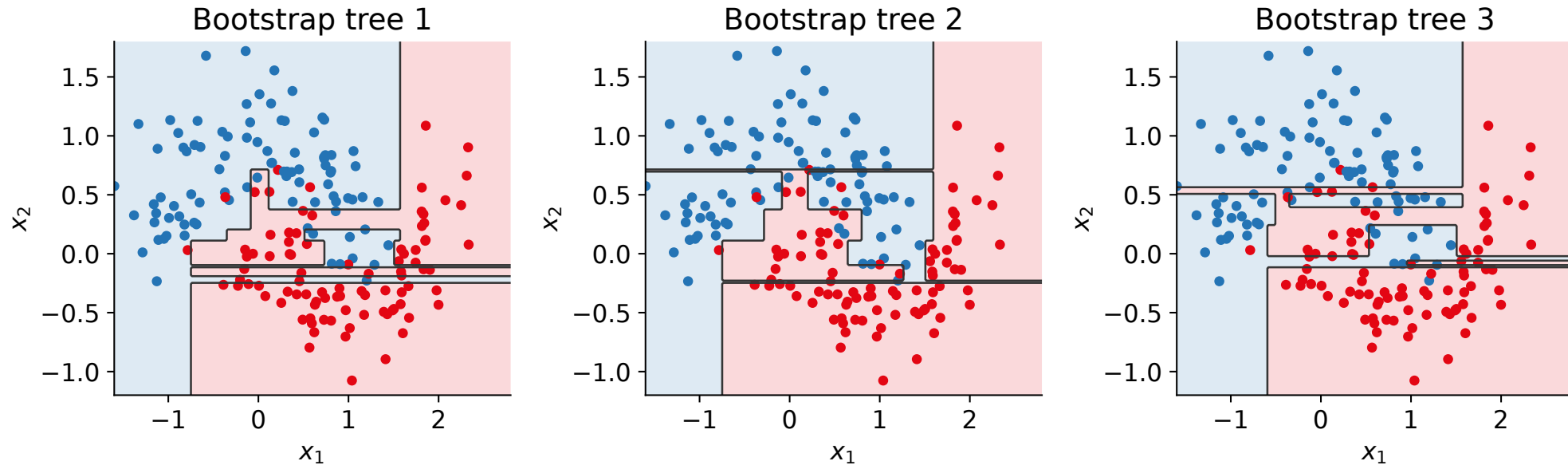
Imagine drawing an observation's index from a bag:

1. Draw an index and copy the corresponding observation.
2. **Put the index back in the bag**, so it can be drawn again.
3. Repeat N times to build one training sample of size N .

```
Original indices:  0 1 2 3 4 5  
One bootstrap:    2 2 5 0 5 1
```

The original data are unchanged. Here 2 and 5 are repeated; 3 and 4 are omitted.

Why combine trees?



Same original dataset; each tree learns from a different bootstrap sample.

Greedy growth amplifies changes: a different root changes the data reaching every descendant.

Bagging: average unstable predictors

Bootstrap aggregating: learn B trees on different bootstrap samples.

For a new input, query every tree and combine their predictions.

Regression example: predictions (4, 6, 5) give the average **5**.

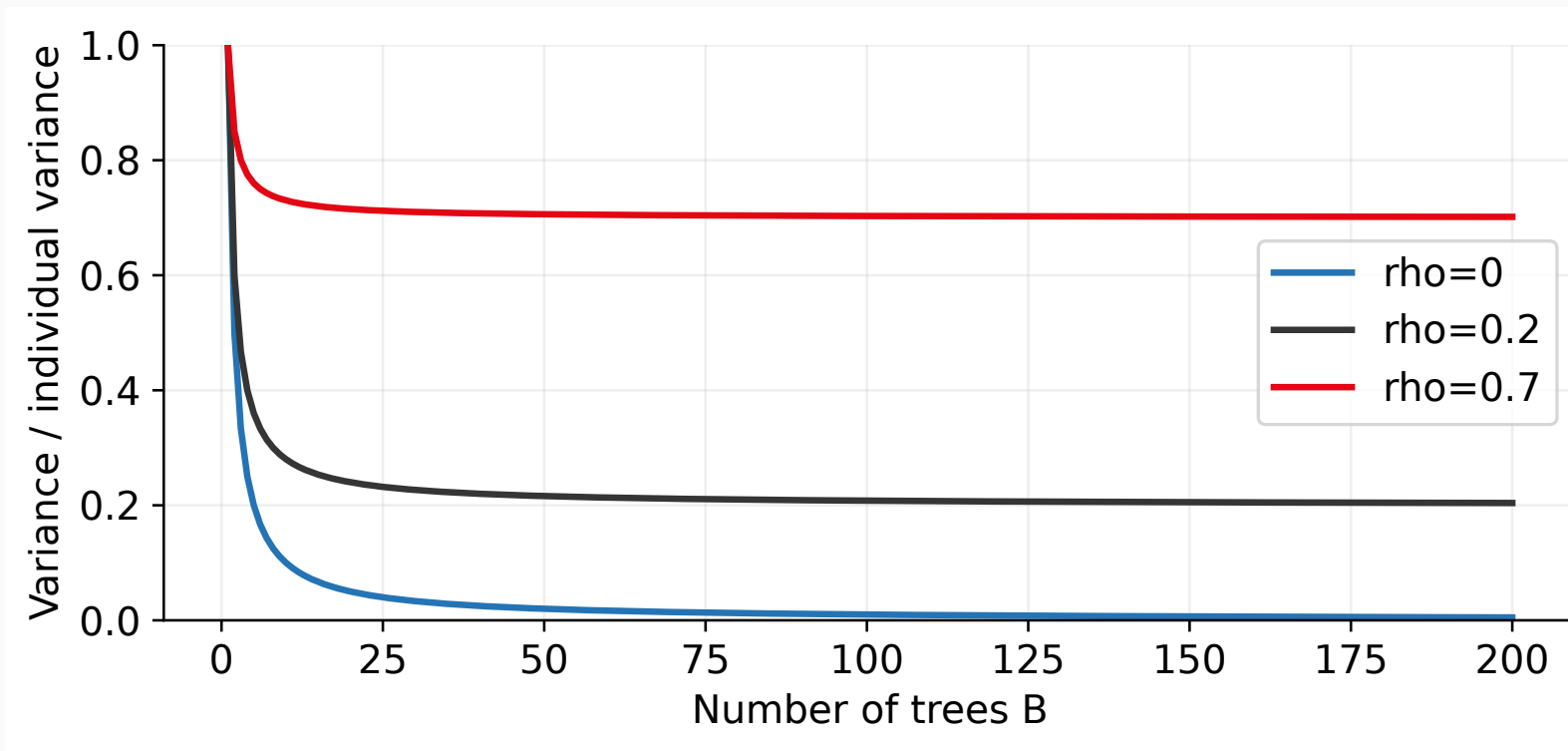
Classification combines class predictions or probabilities (detailed next).

Under an illustrative equal-variance/equal-correlation model:

$$\text{Var}(\bar{f}) = \sigma^2 \left(\rho + \frac{1 - \rho}{B} \right).$$

More useful trees + less correlated errors → more stable predictions.

More trees cannot remove all correlation



This theoretical illustration is not a measured learning curve.
Reducing correlation matters as well as increasing the tree count.

Random forests: randomize rows and candidate features

For each tree:

1. Draw a bootstrap sample of the training observations.
2. At **each node**, randomly select candidate features.
3. Among those features, learn the best threshold by impurity decrease.
4. Grow the tree using its stopping constraints.
5. Aggregate the fitted trees.

The tree uses the **same split criterion** as before.

Which features and thresholds are considered?

Estimator / parameter	Split search at a node
Tree, <code>splitter="best"</code>	Best admissible threshold among candidate features
Tree, <code>splitter="random"</code>	Best among randomly proposed thresholds
Random forest	Best thresholds, with randomized candidate features

`max_features` controls how many candidate features are considered.

- Classifier forest default: `"sqrt"`.
- Regressor forest default: `1.0` — all features.

Smaller subsets can reduce correlation, but also weaken individual trees.

Forest predictions in scikit-learn

Classification averages the trees' **class probabilities**:

$$\hat{p}_k(x) = \frac{1}{B} \sum_{b=1}^B \hat{p}_{b,k}(x), \quad \hat{y}(x) = \arg \max_k \hat{p}_k(x).$$

This can differ from a majority vote of individual tree labels.

Regression averages the trees' numerical predictions:

$$\hat{y}(x) = \frac{1}{B} \sum_b \hat{y}_b(x).$$

Check-up 3: combine probabilities, not just labels

For one observation, three trees predict:

Tree	Probability of class 1	Individual predicted class
1	0.51	1
2	0.51	1
3	0.00	0

What does a hard majority vote predict?

What does scikit-learn's probability averaging predict?

Check-up 3: answer

Hard vote: two trees predict class 1, so class **1** wins.

Probability average:

$$\hat{p}_1 = \frac{0.51 + 0.51 + 0}{3} = 0.34, \quad \hat{p}_0 = 0.66.$$

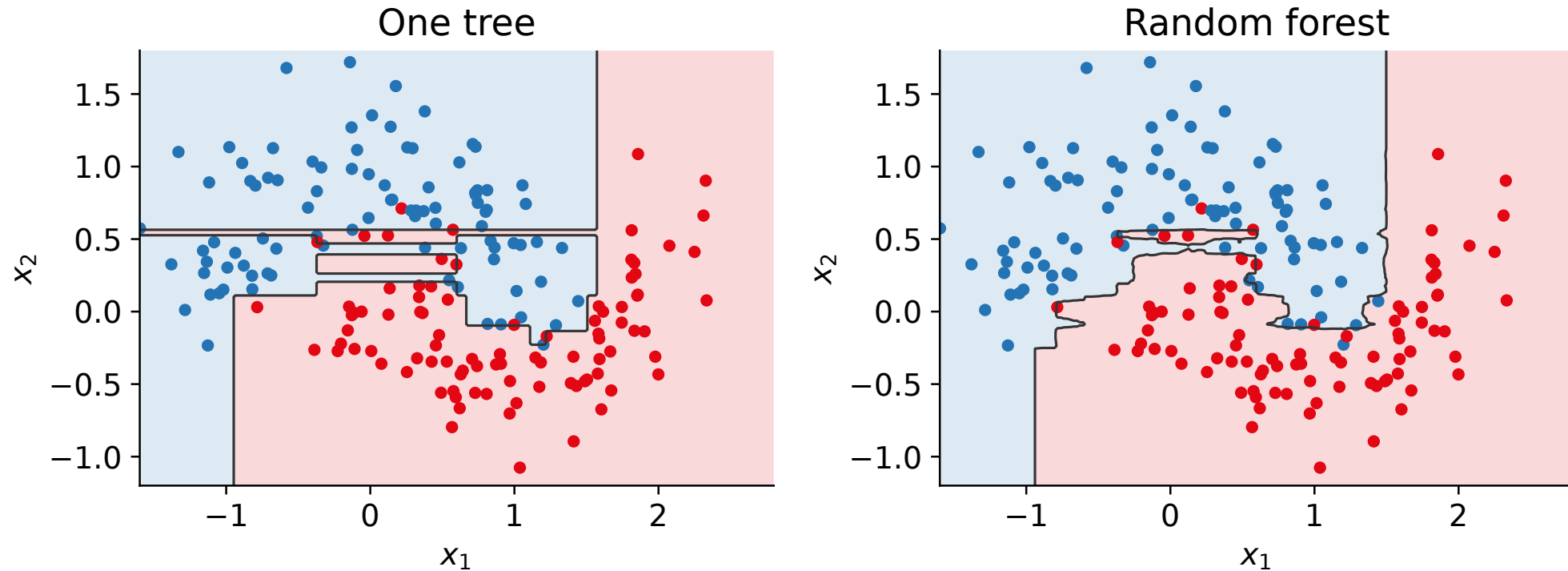
Scikit-learn predicts class **0**.

`RandomForestClassifier` always averages probabilities: **no voting parameter**.

For an explicit choice across fitted-model types, use

`VotingClassifier(voting="soft")` or `voting="hard"` (a different estimator).

A forest can stabilize a complex boundary



Same training observations. Each tree can be complex; the ensemble averages their predictions.

Validation is still needed to assess generalization.

Out-of-bag evaluation

Purpose: estimate how the forest predicts observations omitted during fitting.

For each training observation i :

1. Keep only trees whose bootstrap sample omitted i .
2. Average their predictions for x_i , then compare with y_i .

During tree fitting	After fitting
Impurity decrease chooses the splits	OOB predictions give an evaluation score

`bootstrap=True, oob_score=True` computes `forest.oob_score_`.

It does not change the learned splits. You may compare settings using OOB, but keep an independent final test set.

Fit a forest with explicit parameters

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(
    n_estimators=200, criterion="gini",
    max_features="sqrt", min_samples_leaf=2,
    bootstrap=True, oob_score=True,
    random_state=42, n_jobs=-1,
)
forest.fit(X_dev, y_dev)
print(forest.oob_score_) # OOB accuracy, not test accuracy
probabilities = forest.predict_proba(X_new)
```

The following slide explains each parameter.

Read the forest parameters

Parameter in the example	Meaning
<code>n_estimators=200</code>	Train and combine 200 trees
<code>criterion="gini"</code>	Select splits by Gini decrease
<code>max_features="sqrt"</code>	Random candidate-feature subset at each node
<code>min_samples_leaf=2</code>	Keep at least 2 training observations per leaf
<code>bootstrap=True</code>	Sample training rows with replacement
<code>oob_score=True</code>	Evaluate each observation with trees that omitted it
<code>random_state=42</code>	Reproduce the random draws
<code>n_jobs=-1</code>	Use available CPU cores

These are illustrative settings, not universally optimal values.

Choose the model on development data

Question	Parameters to investigate
Are the trees too detailed?	<code>max_depth</code> , <code>min_samples_leaf</code>
Are the trees too similar?	<code>max_features</code>
Is aggregation stable enough?	<code>n_estimators</code>
How much resampling?	<code>bootstrap</code> , <code>max_samples</code>

Use a task-appropriate metric and the same CV folds for comparisons.

Keep preprocessing inside the evaluation protocol.

Feature importance: useful, but not causal

`feature_importances_` sums weighted impurity decreases attributed to each feature, then normalizes.

- It describes how the fitted model used its features.
- Many candidate thresholds can favour a feature in this measure.
- Correlated features can share or substitute importance.

An importance score is **not a causal effect**.

Next lecture: evaluate importance by permutation on held-out data.

Training cost and prediction cost

Operation	Useful cost model for dense data
Search a node with n_t observations, q candidate features	Sort + scan: about $O(qn_t \log n_t)$
Predict with one tree of depth h	$O(h)$ comparisons
Predict with B trees	$O(Bh)$ for similar depths

Training repeats the split search over many nodes and trees.

Depth, leaf count, implementation and sparsity matter.

A forest needs no $N \times N$ Gram matrix.

Results to retain

1. A tree learns **variables, thresholds and leaf predictions**.
2. Each split reduces **size-weighted child impurity**.
3. The task determines the criterion: class mixing or target loss.
4. Greedy growth needs complexity control and validation.
5. Random forests combine bootstrap samples and randomized feature subsets.
6. Their benefit comes from aggregating useful, diverse predictors.

Next: boosting builds trees sequentially to improve the current ensemble.

References and practical preparation

- [scikit-learn: decision trees and mathematical formulation](#)
- [Classification criteria and parameters](#)
- [Regression criteria and parameters](#)
- [Forests, bagging and OOB evaluation](#)

TP7: fit and visualize a tree, control its size, compare with a random forest and inspect errors.