CHAPTER 6

Decision Trees

Chapter 6 introduces **Decision Trees** as flexible models that can handle classification, regression, and even multioutput tasks, and explains how they are trained, interpreted, regularized, and where they are weak.

Basics and visualization

A Decision Tree recursively splits the feature space into regions using simple threshold tests (e.g., "petal length ≤ 2.45 cm?") until reaching leaf nodes that make predictions. The chapter trains a DecisionTreeClassifier on the iris dataset using petal length and width, limits depth with max_depth=2, and visualizes it via export_graphviz and Graphviz into a readable tree diagram. Each node shows: the splitting rule, the **Gini impurity**, the number of samples, and the class counts; leaves output a predicted class based on the majority label in that node.

Predictions and class probabilities

To classify a new instance, the tree starts at the root and follows the feature tests down to a leaf; the leaf's majority class is the prediction. Trees require **almost no preprocessing** (no scaling or centering) and handle feature interactions by splitting on different features at different depths. A tree can estimate **class probabilities** by returning the fraction of training samples of each class in the leaf where the instance lands, e.g., a leaf with 0 setosa, 49 versicolor, 5 virginica gives probabilities $[0, 49/54, 5/54]$. This makes Decision Trees highly interpretable **white-box** models, in contrast to black-box ensembles or deep nets.

CART algorithm and complexity

Scikit-Learn uses the **CART** (Classification and Regression Trees) algorithm, which greedily picks at each node the feature k and threshold t_k that minimize a weighted impurity measure of the two resulting child subsets. For classification, the CART cost is

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

where G can be **Gini impurity**

$$G_i = 1 - \sum_k p_{i,k}^2$$

or **entropy**

$$H_i = -\sum_k p_{i,k} \log_2 p_{i,k}$$

The algorithm continues splitting recursively until stopping criteria are met (e.g., max_depth, min_samples_split, min_samples_leaf, max_leaf_nodes). Training a tree has time complexity approximately $O(n \cdot m \log m)$ (n features, m instances), while prediction is $O(\log m)$ thanks to tree depth scaling like $\log_2 m$.

Gini vs entropy and regularization

DecisionTreeClassifier uses **Gini impurity by default**, but setting criterion="entropy" switches to entropy; in practice they yield similar trees. Gini tends to isolate the most frequent class a bit faster, while entropy may produce slightly more balanced trees, but the difference is usually minor. Unconstrained trees are **nonparametric** and easily overfit; regularization relies on limiting depth and growth:

- max_depth to cap tree depth
- min_samples_split, min_samples_leaf, min_weight_fraction_leaf to require enough data for splits/leaves
- max_leaf_nodes to bound the number of leaves
  Tighter limits (smaller depth, larger minimum samples, fewer leaves) reduce variance and combat overfitting.

Regression trees and limitations

For regression, DecisionTreeRegressor predicts a numeric value in each leaf, typically the **mean target** of its samples, minimizing **MSE** instead of impurity. The CART criterion becomes a weighted sum of child MSEs, and the resulting prediction function is piecewise constant, forming horizontal segments across regions of the feature space. Regression trees overfit easily as well; controlling max_depth or min_samples_leaf produces smoother, more generalizable predictions.

The chapter also highlights **instability** and axis-aligned splits as key limitations: small changes in the training set can drastically change the learned tree, and purely axis-aligned splits make trees sensitive to feature rotation, potentially leading to convoluted boundaries that generalize poorly. These issues motivate **ensembles of trees** such as Random Forests and Gradient Boosting, discussed in Chapter 7.

.