Chapter 6

Decision Trees

Decision Trees are versatile machine learning algorithms capable of handling classification, regression, and even multioutput tasks. They can model complex relationships in data with little preprocessing, which makes them intuitive and powerful, but also prone to overfitting if left unconstrained. Decision Trees are also the building blocks of ensemble methods such as Random Forests, which significantly improve their performance and stability.

Training and Visualizing a Decision Tree

To understand how Decision Trees work, the chapter begins with a classification example using the Iris dataset. A DecisionTreeClassifier is trained using petal length and width as features, with a limited depth to keep the model simple and interpretable. The trained tree can be visualized using Graphviz, which reveals the hierarchical structure of decisions made by the model.

At each node, the tree asks a yes/no question about a feature threshold, gradually narrowing down the class prediction. This visualization makes Decision Trees particularly attractive, as their logic closely resembles human decision-making.

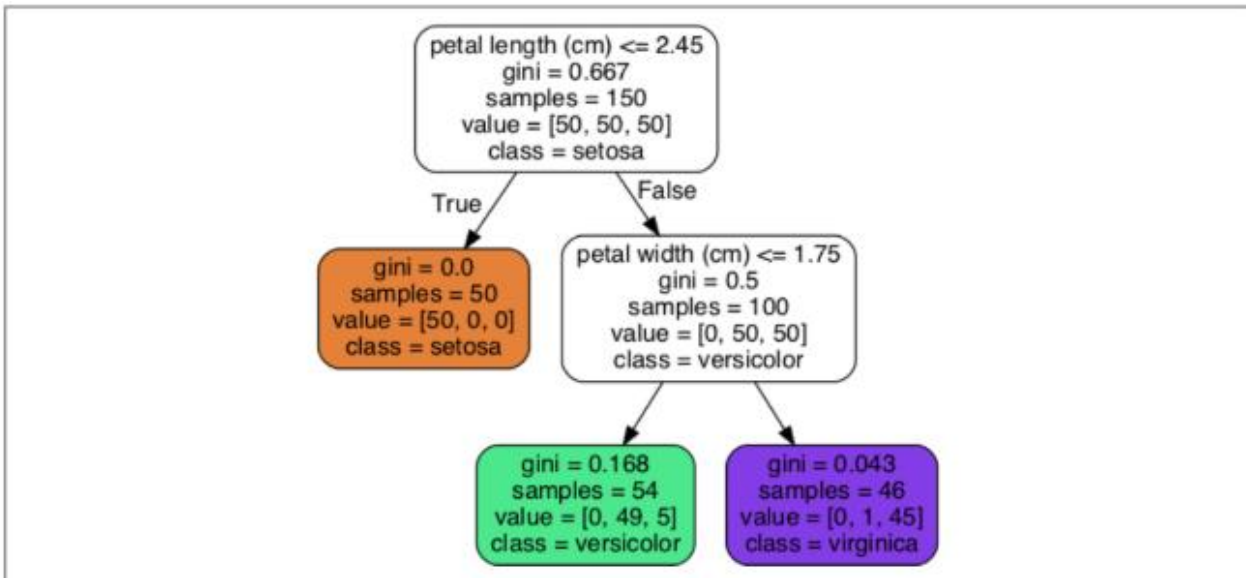


Figure 6-1. Iris Decision Tree

Making Predictions

Predictions in a Decision Tree are made by starting at the root node and following the decision rules down the tree until a leaf node is reached. Each internal node represents a condition on a feature, and each leaf node corresponds to a predicted class.

In the Iris example, the first split separates flowers based on petal length. If the condition is met, the tree immediately predicts *Iris setosa*. Otherwise, a second split based on petal width distinguishes between *Iris versicolor* and *Iris virginica*. This step-by-step traversal illustrates how Decision Trees partition the feature space into rectangular regions.

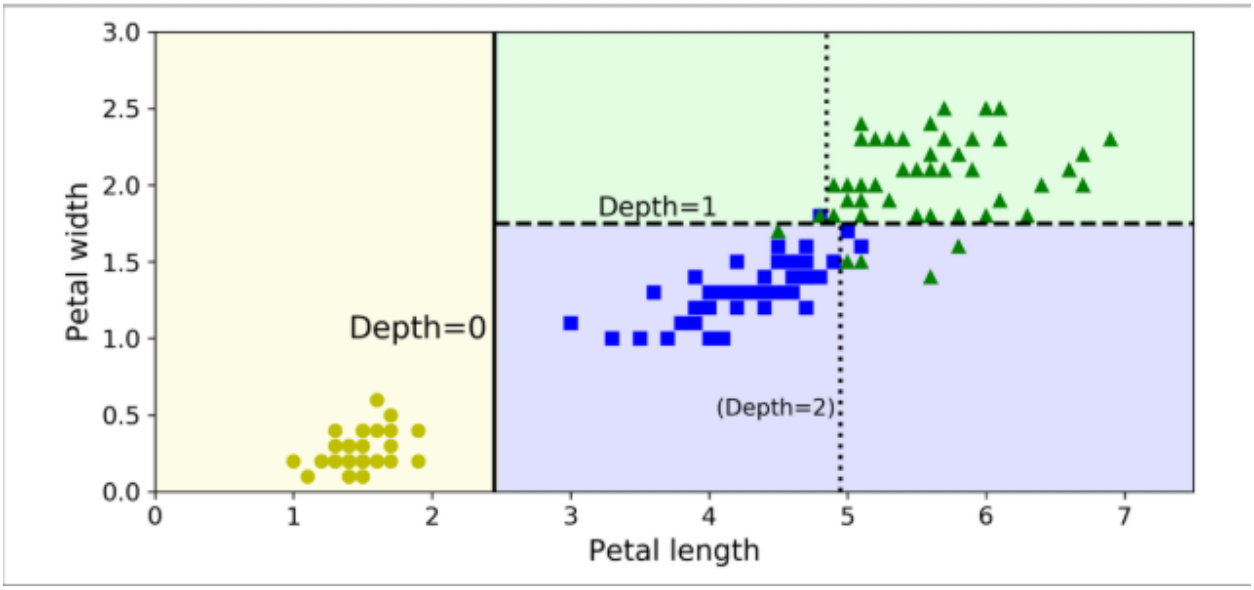


Figure 6-2. Decision Tree decision boundaries

Estimating Class Probabilities

Beyond class labels, Decision Trees can estimate class probabilities. Once an instance reaches a leaf node, the probability for each class is computed as the proportion of training samples of that class within the node. As a result, all instances falling into the same region of the feature space receive identical probability estimates, even if their exact feature values differ.

This probabilistic interpretation allows Decision Trees to be used in scenarios where uncertainty estimation is important, not just hard classification.

The CART Training Algorithm

Scikit-Learn uses the CART (Classification and Regression Tree) algorithm to train Decision Trees. CART builds trees by recursively selecting the feature and threshold that produce the purest child nodes, measured by a cost function based on impurity. For classification, impurity is typically measured using Gini impurity, which quantifies how mixed the classes are within a node.

The algorithm is greedy: it finds the best split at each step without considering future splits. While this makes training efficient, it also means the resulting tree is not guaranteed to be globally optimal. Finding the optimal Decision Tree is an NP-complete problem, which is why practical implementations rely on heuristics like CART.

Computational Complexity

Once trained, Decision Trees make predictions very efficiently. Because trees are usually balanced, prediction requires traversing a path whose length grows logarithmically with the number of training instances. This makes inference fast and independent of the number of features.

Training, however, is more computationally expensive. At each node, the algorithm evaluates multiple features and potential split points, leading to a training complexity that scales with both the number of features and the size of the dataset.

Gini Impurity vs. Entropy

Decision Trees can use either Gini impurity or entropy to measure node impurity. Both metrics aim to quantify how mixed the classes are within a node, and in practice they often produce very similar trees. Gini impurity is slightly faster to compute and is therefore the default choice in Scikit-Learn.

When differences do arise, Gini impurity tends to favor isolating the most frequent class earlier, while entropy often results in more balanced splits. However, these differences are usually minor compared to the impact of other hyperparameters.

Regularization Hyperparameters

Because Decision Trees can grow very deep and complex, they are highly susceptible to overfitting. Regularization is therefore essential. This is typically achieved by limiting the tree’s depth or by enforcing minimum sample requirements for splits and leaf nodes.

Hyperparameters such as max_depth, min_samples_leaf, and max_leaf_nodes restrict the flexibility of the tree, forcing it to learn broader patterns instead of memorizing the training data. Proper regularization significantly improves generalization performance.

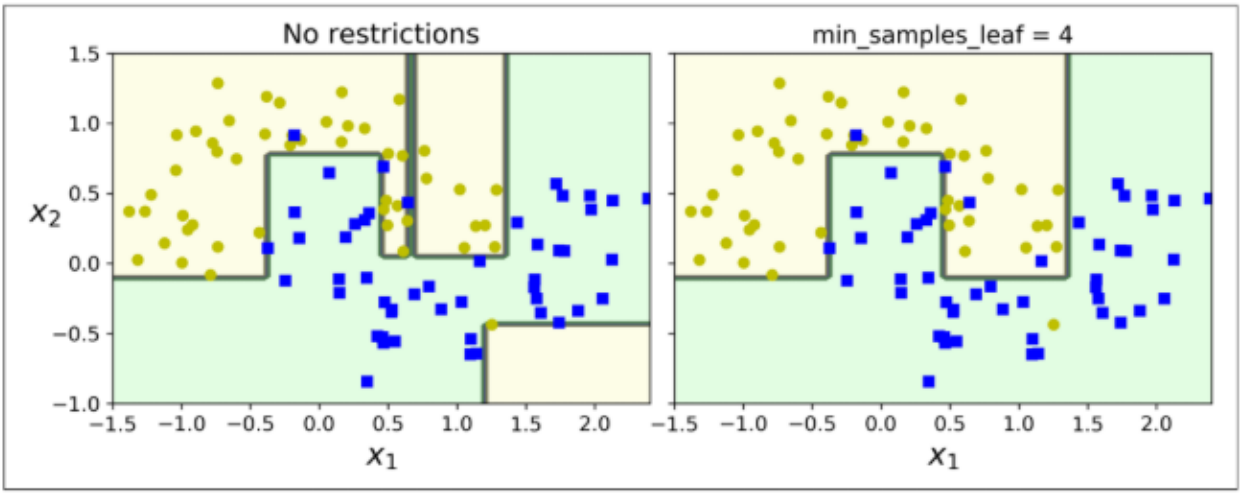


Figure 6-3. Regularization using min_samples_leaf

Regression with Decision Trees

Decision Trees can also be applied to regression tasks. Instead of predicting a class, each leaf node predicts a numerical value, usually the average of the target values of the samples in that node. The splitting criterion is modified accordingly: CART minimizes the mean squared error (MSE) rather than impurity.

As with classification, regression trees partition the feature space into regions with constant predictions. Deeper trees fit the training data more closely, while shallow trees produce smoother predictions.

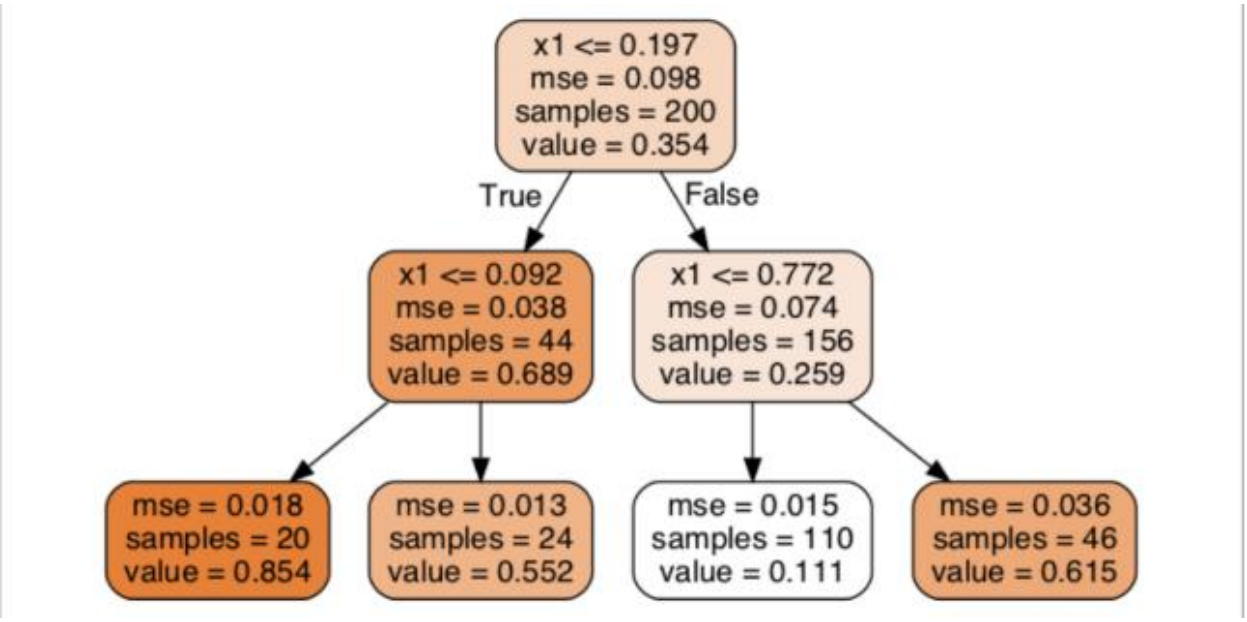


Figure 6-4. A Decision Tree for regression

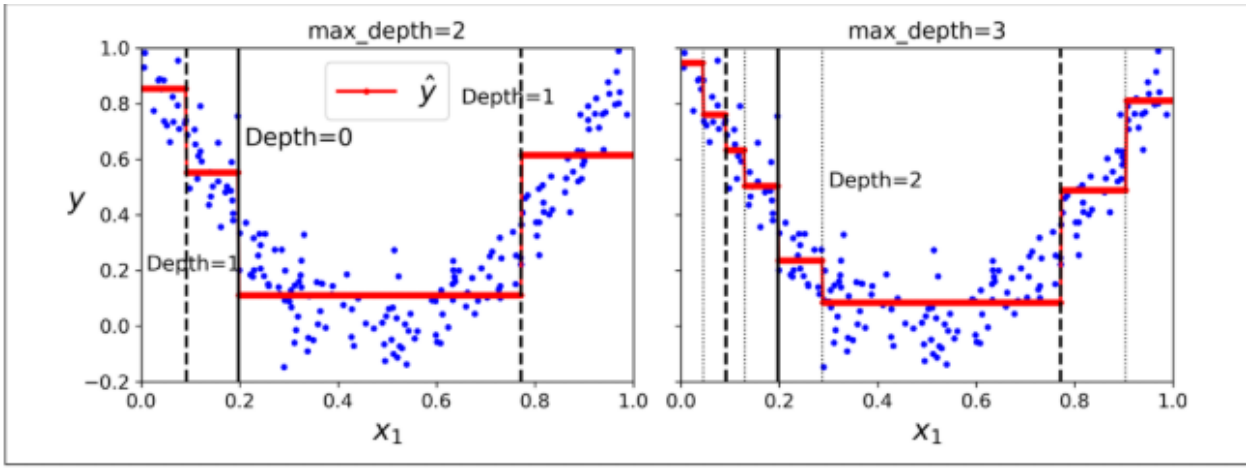
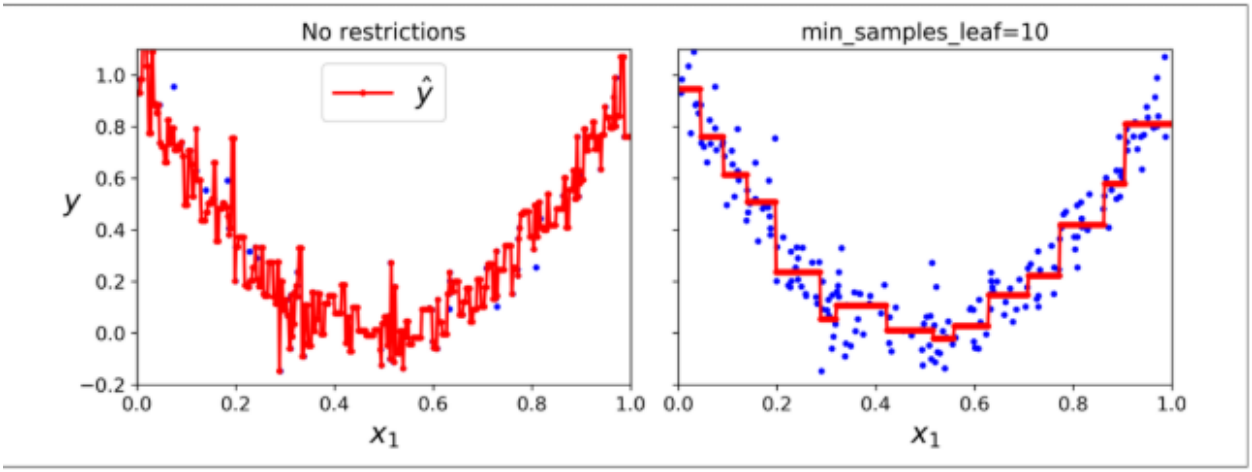


Figure 6-5. Predictions of two Decision Tree regression models



/Figure 6-6. Regularizing a Decision Tree regressor

Instability of Decision Trees

Despite their strengths, Decision Trees suffer from instability. Small changes in the training data can lead to very different tree structures, which makes their predictions sensitive to noise. They are also biased toward axis-aligned splits, meaning they struggle with rotated datasets.

Techniques such as feature preprocessing, dimensionality reduction, and ensemble methods like Random Forests are commonly used to mitigate these weaknesses and improve robustness.

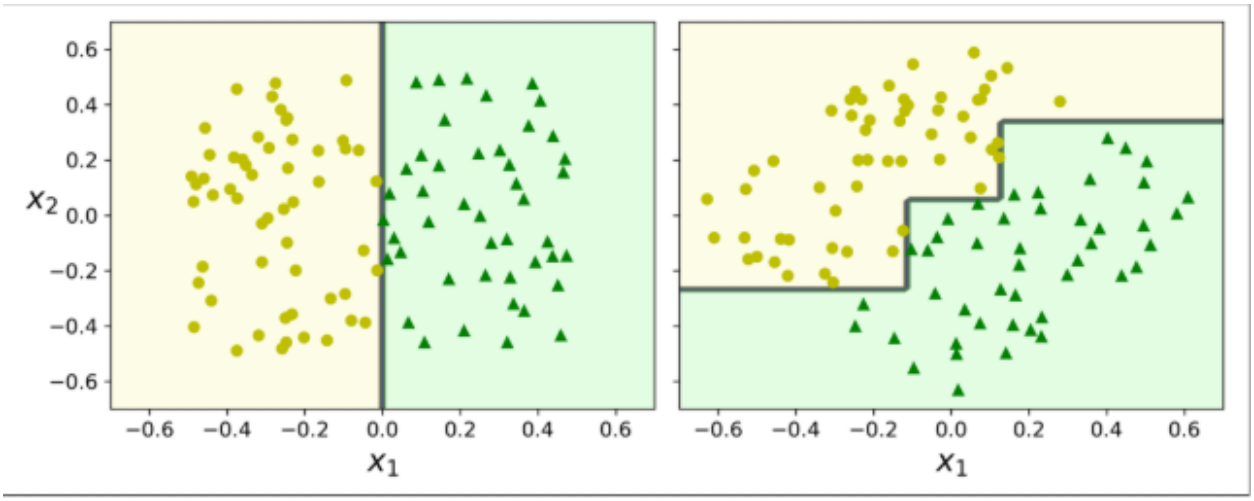


Figure 6-7. Sensitivity to training set rotation

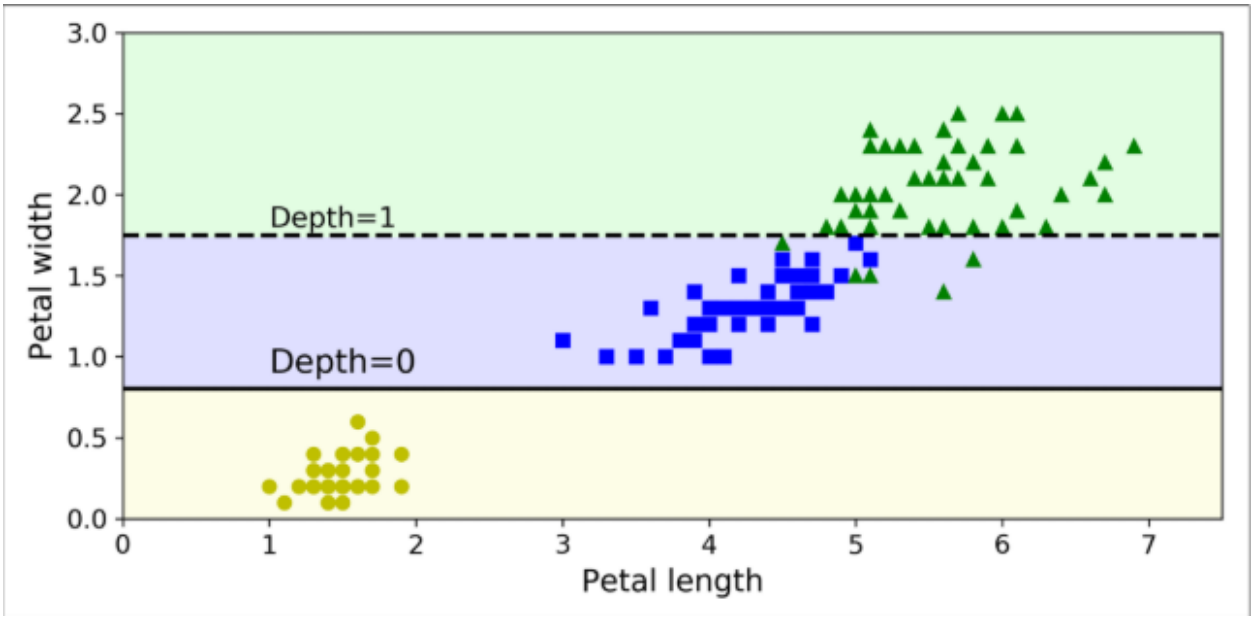


Figure 6-8. Sensitivity to training set details