5 Decision Trees

Decision trees are flexible learners that work well for classification, regression, and multi-output problems. A single tree can capture intricate relationships in the data with minimal preprocessing. Each tree also acts as the core building block of ensemble methods such as Random Forests, which are among the most reliable models in modern practice. One limitation is size: a fully grown tree can become quite large, so pruning or depth limits are often applied to control memory use and reduce overfitting.

In this chapter, we will explore the essentials of Decision Trees, starting with their training, visualization, and prediction processes. We will then look into the CART (Classification and Regression Trees) training algorithm, which Scikit-Learn utilizes for constructing Decision Trees. Additionally, we will examine how to regulate the complexity of Decision Trees and adapt them for regression tasks. The chapter concludes by addressing some inherent limitations of Decision Trees.

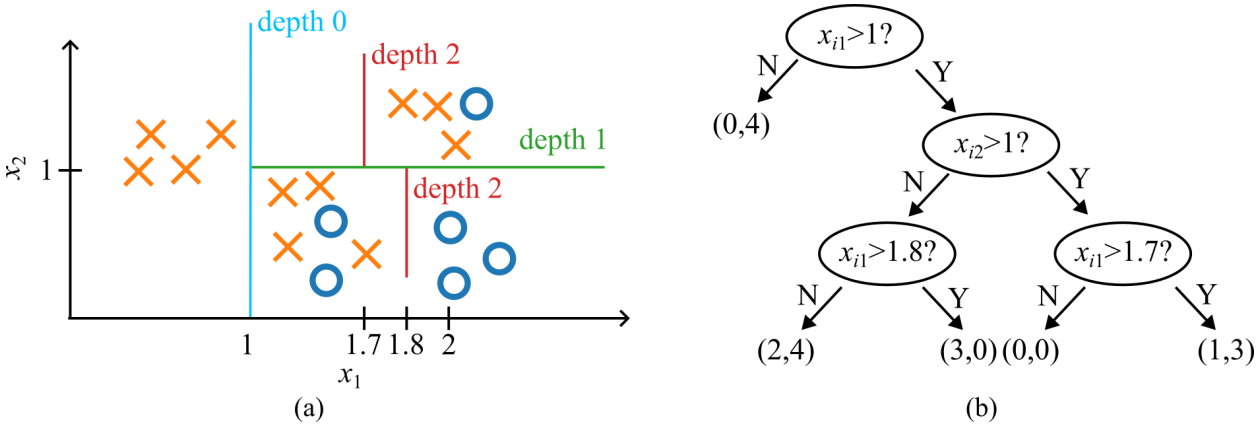


Figure 5.1: The basic connect of a decision tree, showing (a) how a decision tree is built, and (b) the developed decision tree.

5.1 Decision Tree Classification

Figure 5.2 shows the regions formed by the decision tree. The solid vertical line marks the root split at depth 0, where petal length equals 2.45 cm. All samples on the left of this line belong solely to Iris setosa, so no further division is needed there. The right half still mixes species, so the depth-1 right child splits again at petal width 1.75 cm, indicated by the dashed line. With `max_depth=2` the tree stops at this point. If `max_depth` were increased to 3, the two depth-2 leaves would introduce additional boundaries, drawn as dotted lines.

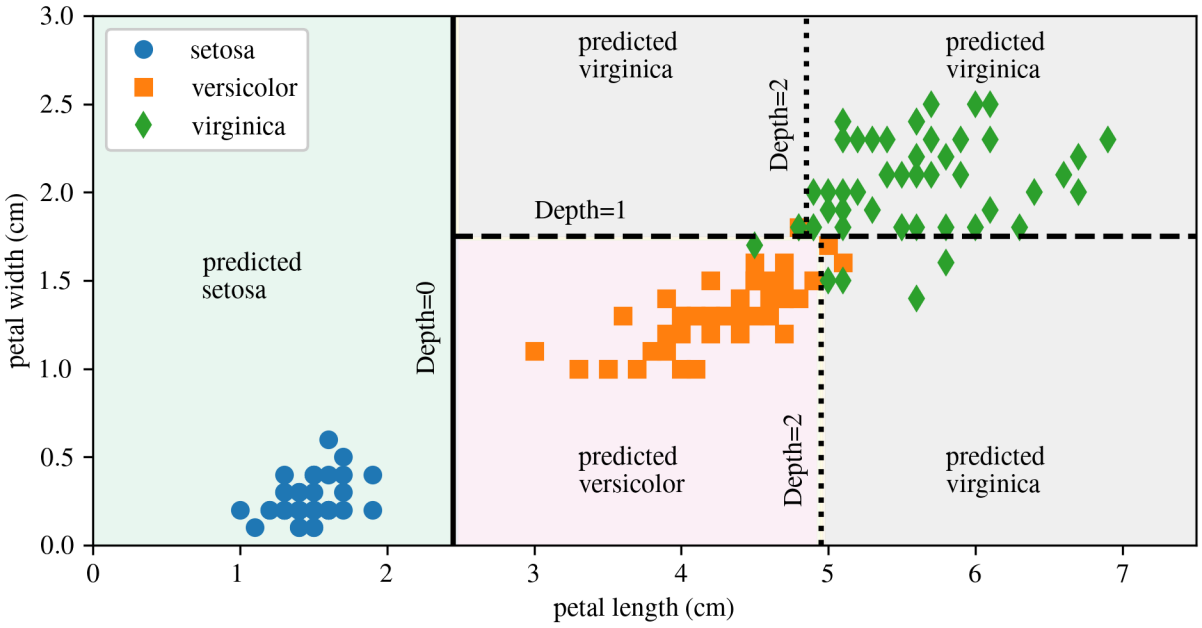


Figure 5.2: Decision Tree decision boundaries.

A decision tree for the Iris Dataset looks like Figure 5.3. Let us examine how the Decision Tree processes predictions. Suppose you come across an iris and need to classify it. Start at the root node (depth=0), which asks whether the flower's petal length is less than 2.45 cm. If the answer is yes, move to the root's left child (depth=1). This child is a leaf, so no further questions follow. The class stored in that leaf is Iris-setosa, and the tree therefore labels the sample as setosa.

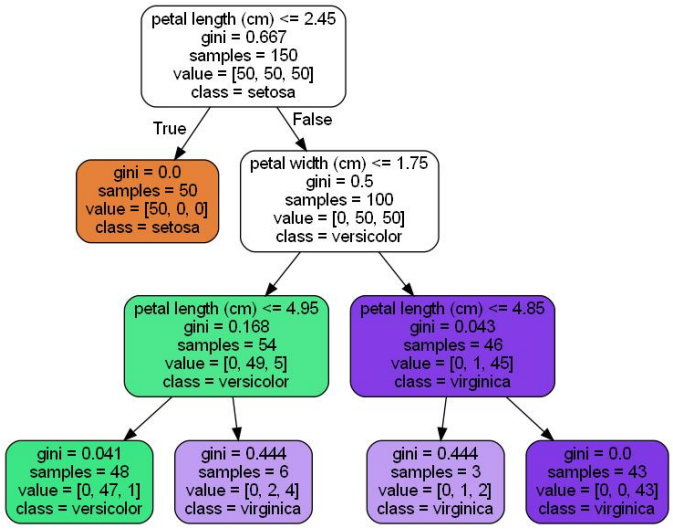


Figure 5.3: Decision tree for the Iris Dataset trained using the CART algorithms.

In Figure 5.3, each node reports the split criterion, Gini impurity, sample count, and class distribution. Fill colors denote the predicted species; orange for Setosa, green for Versicolor, and

purple for Virginica. Shade intensity conveys confidence: darker hues indicate purer (more certain) leaves, whereas white represents complete uncertainty. Consider a second iris whose petal length is greater than 2.45 cm. From the root you move to its right child (depth 1). This internal node asks a new question: is the petal width less than 1.75 cm? If yes, the flower is classified as Iris-versicolor (depth=2, left leaf). If no, it is labeled Iris-virginica (depth=2, right leaf). The decision path is clear and easy to follow.

The attributes of a node include:

- *class*: the class label stored in the leaf.
- *samples*: the number of training instances that reach the node. For example, 100 flowers have petal length >2.45 cm (depth1, right); among them, 54 also have petal width < 1.75 cm (depth2, left).
- *value*: a three-element vector giving the count of instances from each species at the node. The bottom-right leaf, for instance, contains 0 Iris-setosa, 1 Iris-versicolor, and 45 Iris-virginica.
- *gini*: the Gini impurity. A node is *pure* when $gini = 0$, meaning all samples belong to the same class. The depth-1 left node, which holds only Iris-setosa, is pure.

The Gini impurity G_i for the i^{th} node is computed using

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2, \tag{5.1}$$

where $p_{i,k}$ is the proportion of class k instances among the training instances at the i^{th} node. For instance, the Gini score for the depth-2 left node is calculated as follows: $1-(0/54)^2-(49/54)^2-(5/54)^2 \approx 0.168$. We will discuss an alternative impurity measure later.

5.1.1 Class Probability Estimation

A Decision Tree can estimate the likelihood that a sample belongs to class k. The algorithm traces the sample's path to its leaf node and then reports the fraction of training instances of class k found in that leaf. Consider a flower with petals 5cm long and 1.5cm wide. This sample reaches the depth-2 left leaf, which contains 54 training flowers: 0 *Iris*-setosa, 49 *Iris*-versicolor, and 5 *Iris*-virginica. The tree therefore assigns probabilities of 0%, 90.7%, and 9.3% to the three classes, respectively. When asked for a class label, it selects *Iris*-versicolor, the class with the highest probability.

Interestingly, the same estimated probabilities apply throughout the bottom-right rectangle of Figure 5.2, even in a scenario where the petal dimensions are 6 cm by 1.5 cm where one might intuitively expect an Iris-Virginica classification.

Example 5.1 Decision Tree Classifier
This example uses Scikit-Learn's `DecisionTreeClassifier` to classify iris species based on petal features. The decision process is visualized using Graphviz, showing the structure of

learned rules and how decisions are made based on feature thresholds. This online viewer is the easiest way to do that https://dreampuf.github.io/GraphvizOnline/

5.2 The CART Training Algorithm

Scikit-Learn trains decision trees with the Classification and Regression Tree (CART) algorithm[a], a process often called "growing" the tree. At each node the algorithm splits the local training set into two parts by selecting a feature k and a threshold t_k, for example "petal length ≤ 2.45 cm". It evaluates every candidate pair (k, t_k) and chooses the one that yields the purest child nodes, judging purity by a weighted combination of each child's impurity measure and sample count.

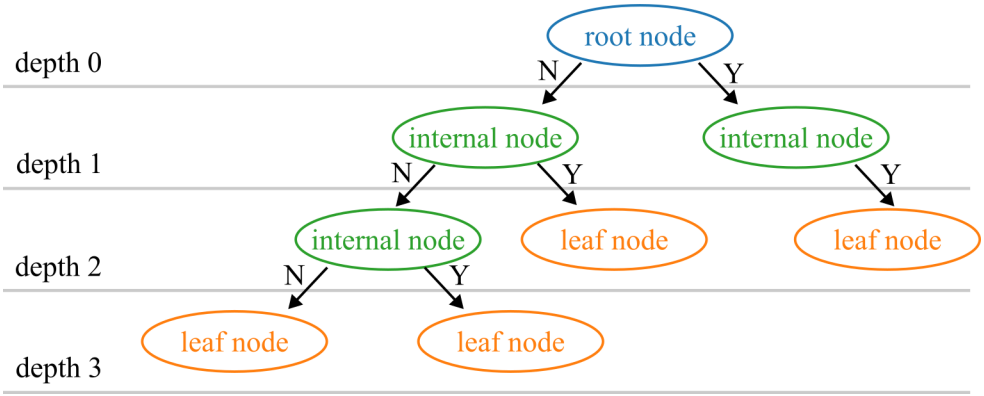


Figure 5.4: Illustration of the CART algorithm process.

The objective function it aims to minimize during this process is

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}. \tag{5.2}$$

In this equation, $G_{\text{left/right}}$ measures the impurity of the left/right subset, and $m_{\text{left/right}}$ is the number of instances in the left/right subset. After the initial split, the algorithm continues to divide the resulting subsets and their subsequent divisions recursively. This recursive process halts when it reaches the predefined maximum depth (`max_depth`) set through a hyperparameter, or if no further impurity-reducing splits can be found.

Unlike linear models that assume a specific data structure, Decision Trees impose few assumptions on their initial data. If not appropriately constrained, a Decision Tree can intricately conform to the training data, leading to overfitting. This model type is described as nonparametric, not due to a lack of parameters, which it can have in abundance, but because the parameters are not predetermined before training, allowing the model's structure to freely mirror the data intricacies. Conversely, a parametric model, like a linear model, has a fixed number of parameters, limiting its flexibility but also minimizing overfitting risks while potentially increasing underfitting risks.

To mitigate overfitting in Decision Trees, it is essential to control the model's freedom during training through regularization. The regularization hyperparameters vary by the algorithm, but

[a]Breiman, Leo. Classification and regression trees. Routledge, 2017

typically, the tree's maximum depth can be restricted. In Scikit-Learn, this is managed by the `max_depth` hyperparameter, which is unlimited by default. Lowering `max_depth` helps regularize the model, thereby reducing overfitting likelihood.

Other parameters in Scikit-Learn's DecisionTreeClassifier also influence the tree's structure:

- `min_samples_split`: The minimum number of samples required to split a node.
- `min_samples_leaf`: The minimum number of samples a leaf node must have.
- `min_weight_fraction_leaf`: Similar to `min_samples_leaf`, but expressed as a fraction of total weighted instances.
- `max_leaf_nodes`: The maximum number of leaf nodes.
- `max_features`: The maximum number of features evaluated for splitting at each node.

Adjusting these parameters by increasing `min_*` values or decreasing `max_*` values will help regularize the model.

Figure 5.5 illustrates two Decision Trees trained on the moons dataset: one on the left with default hyperparameters (unrestricted) and another on the right with `min_samples_leaf=4`. The left model appears to be overfitting, whereas the right model, with its restrictions, likely offers better generalization.

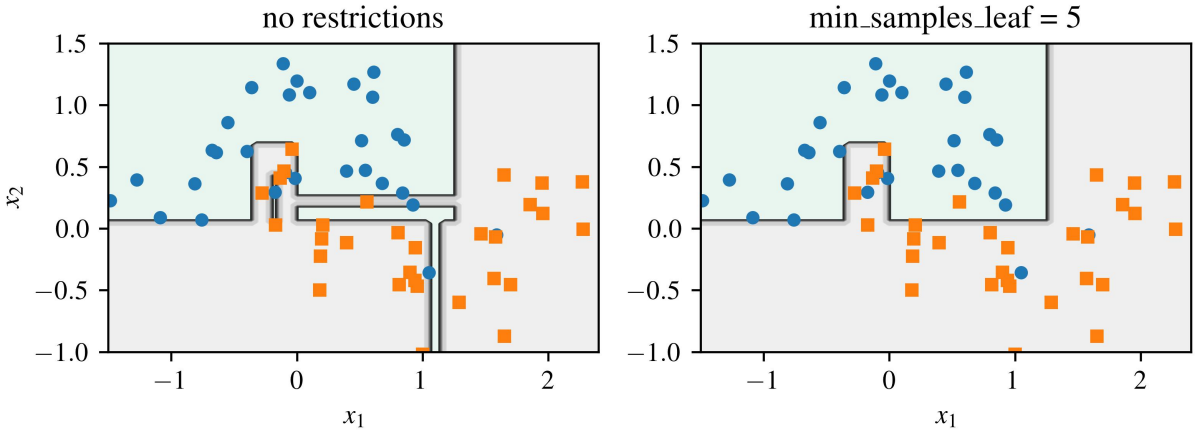


Figure 5.5: Regularization effects using `min_samples_leaf`

5.2.1 Computational Complexity

To predict with a Decision Tree the algorithm follows a path from the root to a leaf. Because most trained trees are roughly balanced, this path crosses about

$$T_{\text{predict}} = O\big(\log_2 m\big), \tag{5.3}$$

nodes, where m is the number of training samples. Each node tests a single feature, so the prediction cost in equation 5.3 is unaffected by the total feature count and is therefore very fast.

Training is costlier. At every split the algorithm scans each candidate feature (or the subset limited by `max_features`) over all samples that reach the node. For n input features the total work across all levels is

$$T_{\text{train}} = O\big(n \times m \log m\big), \tag{5.4}$$

because the tree has roughly $\log_2 m$ layers and each layer processes a shrinking fraction of the m samples.

When the dataset contains only a few thousand samples, enabling `presort=True` in Scikit?Learn can shorten training time, but for larger datasets the presorting step turns into a bottleneck and the standard (unsorted) approach is quicker.

5.2.2 Entropy verse Gini Impurity

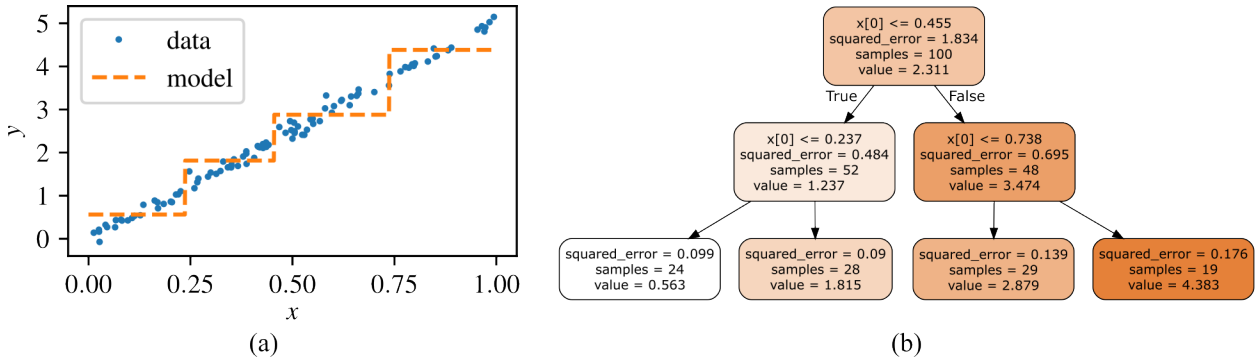
Decision Trees in sklearn use the Gini impurity as the default criterion for node purity, entropy can also be utilized by setting the criterion hyperparameter to "entropy". Originally a thermodynamic concept representing molecular disorder, entropy reaches zero when molecules are in a complete state of rest and order. This concept was later adopted in information theory, introduced by Shannon, to quantify the average informational content in messages entropy is zero when all messages are identical. In the realm of Machine Learning, entropy serves as a measure of impurity: it is zero in a dataset if all its elements belong to a single class. The entropy for the i^{th} node is defined as

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2(p_{i,k}). \tag{5.5}$$

The choice between Gini impurity and entropy often results in negligible differences, producing similar trees. Gini impurity has a slight computational advantage and is thus the default choice. However, it tends to separate the most frequent class into a distinct branch, whereas entropy generally yields more balanced trees.

5.3 Decision Tree Regression

Decision Trees are not limited to classification tasks; they can also be adapted for regression. To illustrate, we use Scikit-Learn's `DecisionTreeRegressor` to train a regression tree on a noisy linear dataset with `max_depth=2`. The structure of this tree is displayed in Figure 5.6. In contract to the color scheme for classification, the change in color shades here relate to the predicted value of y.



(a) (b)

Figure 5.6: A regression model developed using a Decision Tree, showing the: (a) model superimposed over noisy data, and (b) the decision tree developed for the task.

The workings of a regression tree mirror those of a classification tree, except that each leaf holds a continuous prediction instead of a class label. To obtain a prediction for a sample with $x_1 = 0.6$, trace the path from the root to its leaf; that leaf outputs 0.1106, which is the average target value among the 110 training instances that reach it. For those instances the mean squared error is 0.0151.

The predictions from this model are visualized in Figure 5.7, with results shown for trees of depth 2 and 3. The deeper tree partitions the input space into more regions, with each region's predicted value being the average target value of the instances it encompasses. The model attempts to organize the regions such that the instances within each are as close as possible to their predicted value.

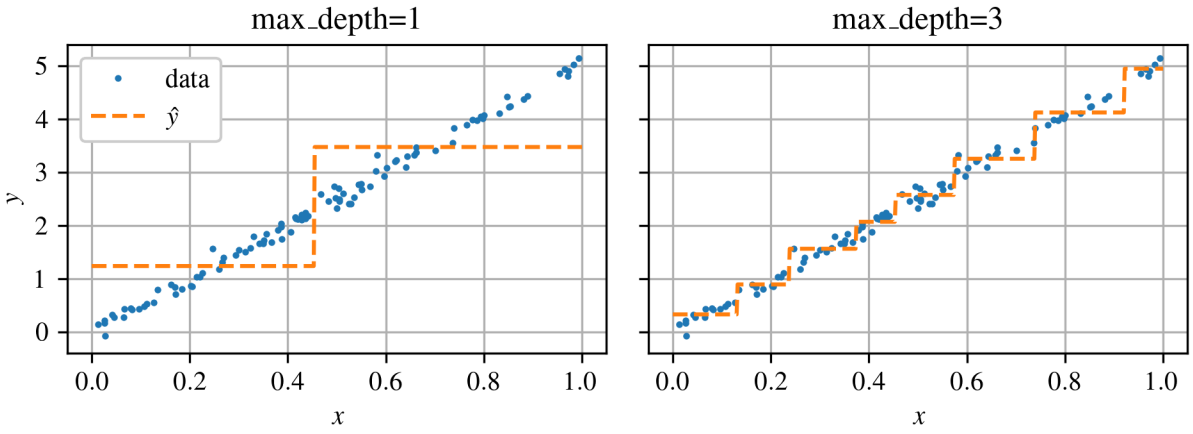


Figure 5.7: Comparison of predictions from two Decision Tree regression models with varying depths.

The CART algorithm for regression trees aims to minimize the MSE when splitting the training set, similar to how it minimizes impurity in classification tasks. The cost function minimized by the algorithm is represented as

$$J(k,t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}}. \tag{5.6}$$

Knowing that,

$$\text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)} \right)^2 \tag{5.7}$$

and

$$\hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)}. \tag{5.8}$$

Similar to classification, Decision Trees for regression can overfit if not properly regularized. Without regularization, the predictions, as depicted on the left of Figure 5.8, can fit the training data excessively. By setting `min_samples_leaf` to 15, a more generalized model is achieved, as shown on the right in the same figure.

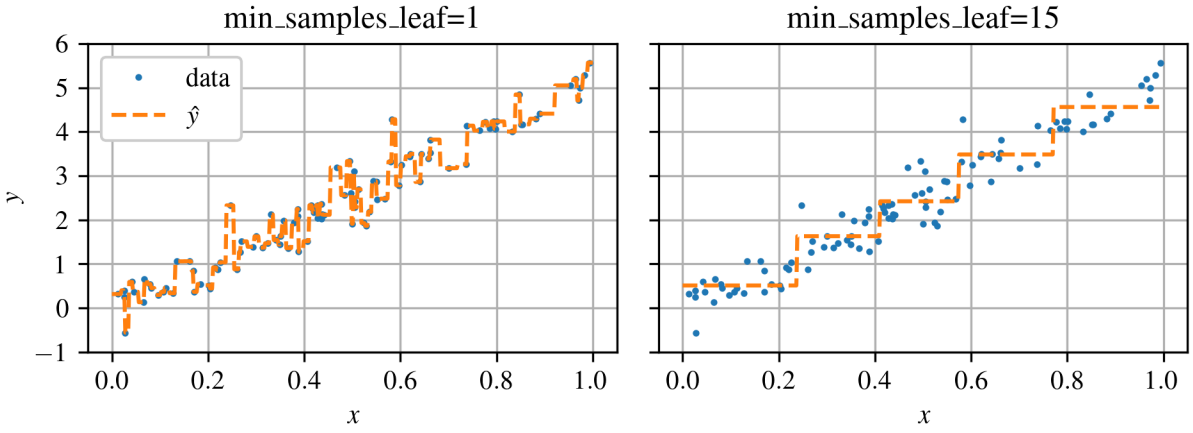


Figure 5.8: Impact of regularization through setting the minimum number of samples in a leaf on a Decision Tree regression model.

Example 5.2 Decision Tree Regression
This example uses Scikit-Learn's `DecisionTreeRegressor` to fit a non-linear relationship between input and output data. A tree of depth 4 is trained on noisy linear data and visualized using Graphviz to show how the regression model partitions the input space.

5.4 Instability

While Decision Trees offer simplicity, interpretability, versatility, and power, they come with certain drawbacks. A notable limitation is their preference for orthogonal decision boundaries, which makes them highly sensitive to the orientation of the data. For instance, Figure 5.9 illustrates how a simple linearly separable dataset can be perfectly split by a Decision Tree in its original alignment, whereas a 45° rotation results in a convoluted decision boundary. Despite perfect fits to their respective training sets, the rotated tree's model is likely to perform poorly on unseen data. Utilizing Principal Component Analysis (PCA) can often mitigate this issue by reorienting the data more suitably.

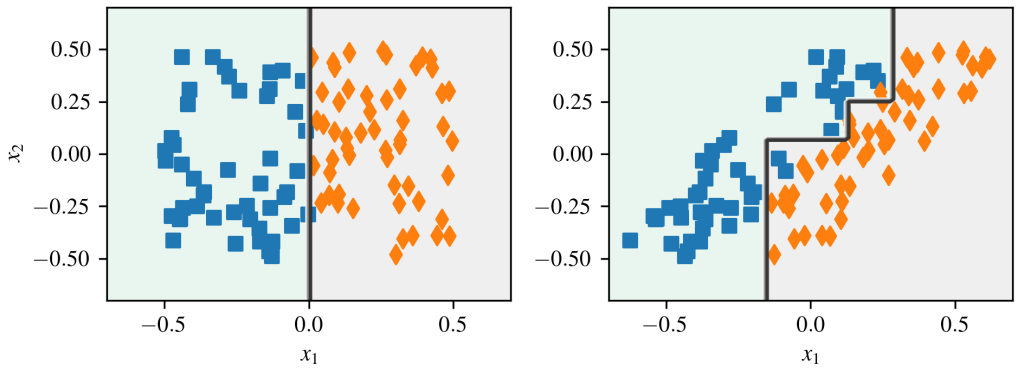


Figure 5.9: Sensitivity to training set rotation. Decision trees create a single clean split on the original data (left), but after a 45° rotation they must build a jagged, multi-step boundary; illustrating their sensitivity to feature orientation.

More broadly, Decision Trees are sensitive to small variations in training data. For instance, changing the seed of the random number generator can lead to a substantially different model, as shown in Figure 5.10. This variability is partly due to the stochastic nature of the Scikit-Learn's training algorithms; different models may result from the same data unless the `random_state` hyperparameter is fixed.

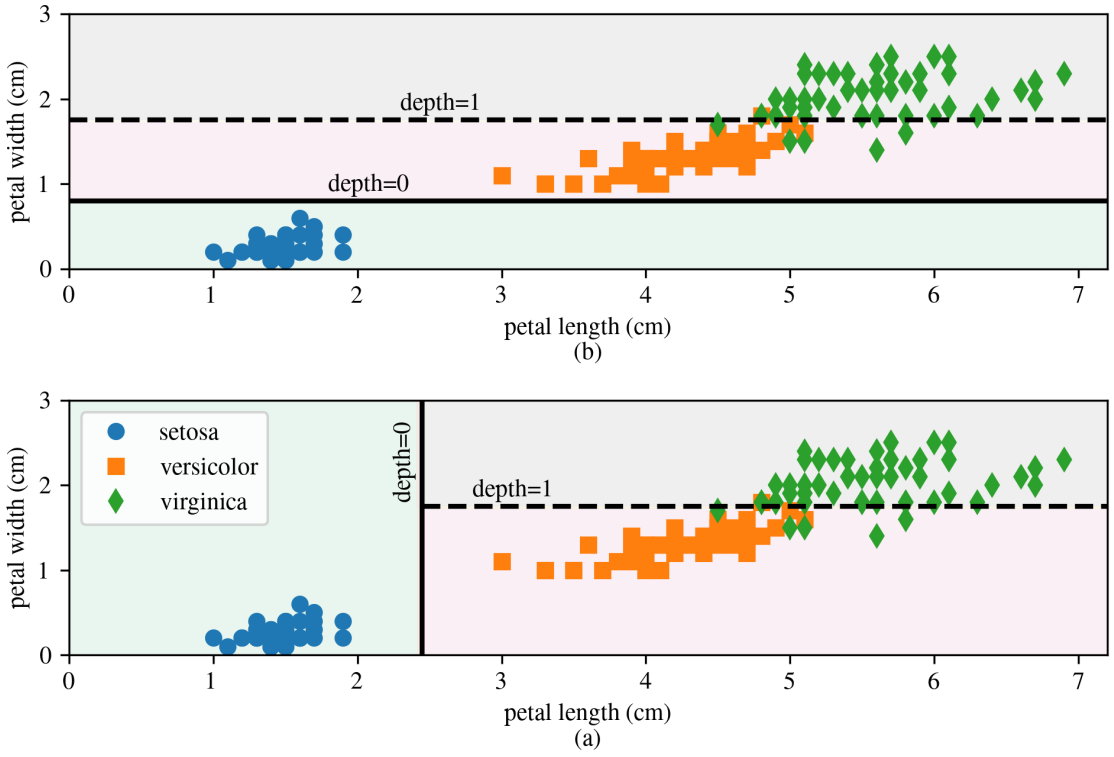


Figure 5.10: Decision Tree Sensitivity to initial conditions, showing: (a) random number generator seeded with "1", and: (b) random number generator seeded with "2". [a]

5.5 Random Forest

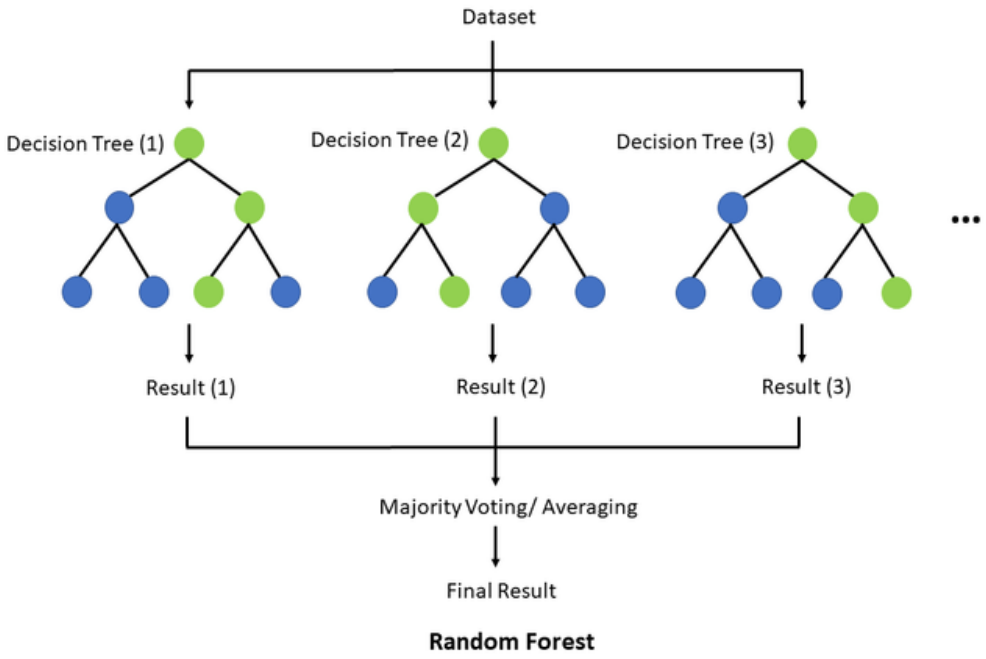


Figure 5.11: Random forest with majority voting. [a]

Random Forests reduce the high variance of individual decision trees by building many trees on bootstrap samples of the training data and then averaging their outputs. Each tree is grown with a random subset of input features, so the ensemble decorrelates the trees and improves generalisation. For classification problems the forest predicts the class that receives the majority vote, while for regression it returns the mean of the trees' numeric predictions. This bagging strategy limits overfitting and usually yields better performance than a single tree, although it often falls short of the accuracy achieved by gradient-boosted ensembles. Model quality still depends on the data's size, noise level, and feature interactions, and on hyperparameters such as the number of trees, the maximum depth, and the number of features considered at each split.

[a] "2" is the author's preferred random number seed.
[a] TseKiChun, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons