2025-07-14 23:16

tags :

- Machine Learning
- Hands on ML - book

Chapter 6 - Hands on

Training and Visualizing a Decision Tree

- A **DecisionTreeClassifier** is trained on the Iris dataset using two features: *petal length* and *petal width*.
- Example :

```
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_iris, y_iris)`
```

- Visualization:
    - Use `export_graphviz()` to create a `.dot` file for the tree.
    - Display in a Jupyter notebook using `graphviz.Source.from_file()`.

```
export_graphviz( tree_clf, out_file="iris_tree.dot",
                 feature_names=["petal length (cm)", "petal width (cm)"],
                 class_names=iris.target_names, rounded=True, filled=True
                 )

from graphviz import Source
Source.from_file("iris_tree.dot")
```

- Graphviz is an open source graph visualization software package. It also includes a dot command-line tool to convert .dot files to a variety of formats, such as PDF or PNG.

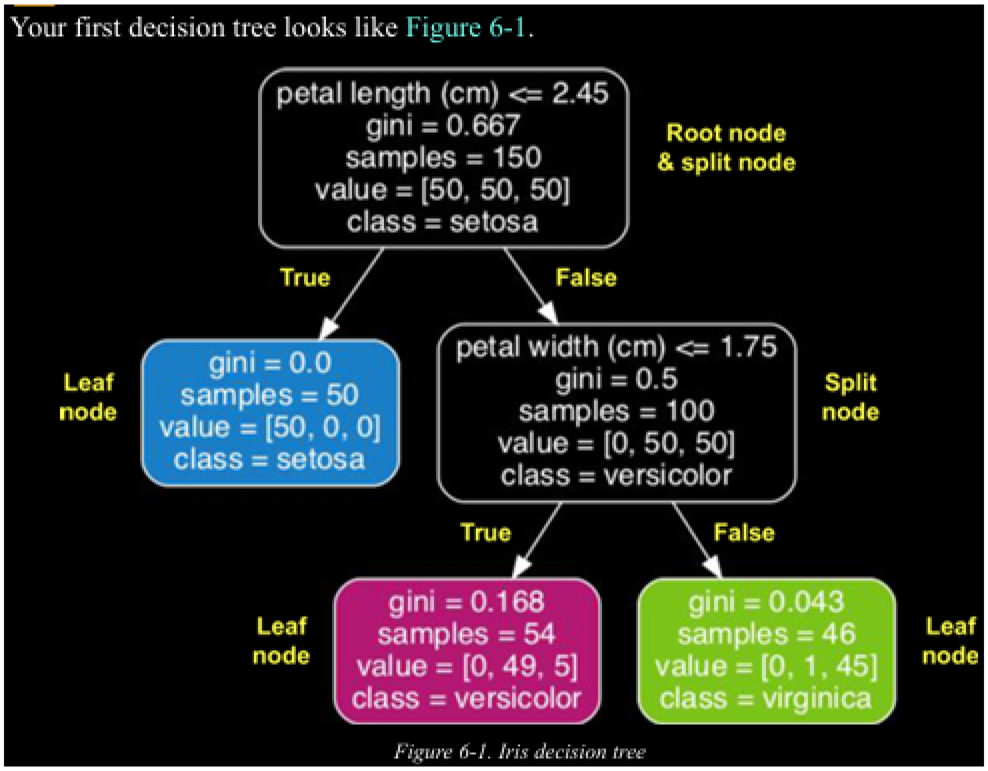
- 
  Your first decision tree looks like Figure 6-1.

  Figure 6-1. Iris decision tree

Making Predictions

- To classify a flower, start at the root and follow the splits:
  - If **petal length ≤ 2.45 cm** → predict *Iris setosa* (leaf node).
  - Else → check **petal width ≤ 1.75 cm**:
    - If yes → predict *Iris versicolor*.
    - If no → predict *Iris virginica*.
- **Advantages**:
  - No need for feature scaling or centering; trees work directly on raw data.- To classify a flower, start at the root and follow the splits:
  - If **petal length ≤ 2.45 cm** → predict *Iris setosa* (leaf node).
  - Else → check **petal width ≤ 1.75 cm**:
    - If yes → predict *Iris versicolor*.
    - If no → predict *Iris virginica*.
- **Key Node Attributes**:
  - `samples` : Number of training instances at the node.
  - `value` : Count of instances per class at the node.
  - `gini` : Measures impurity (0 = pure).
    - Example: Gini impurity at a node:
      $G = 1 - \sum_{k=1}^{n} (p_{i,k})^2$
      where $p_{i,k} = ratio$ of class k in node i.
- Decision Tree Boundaries

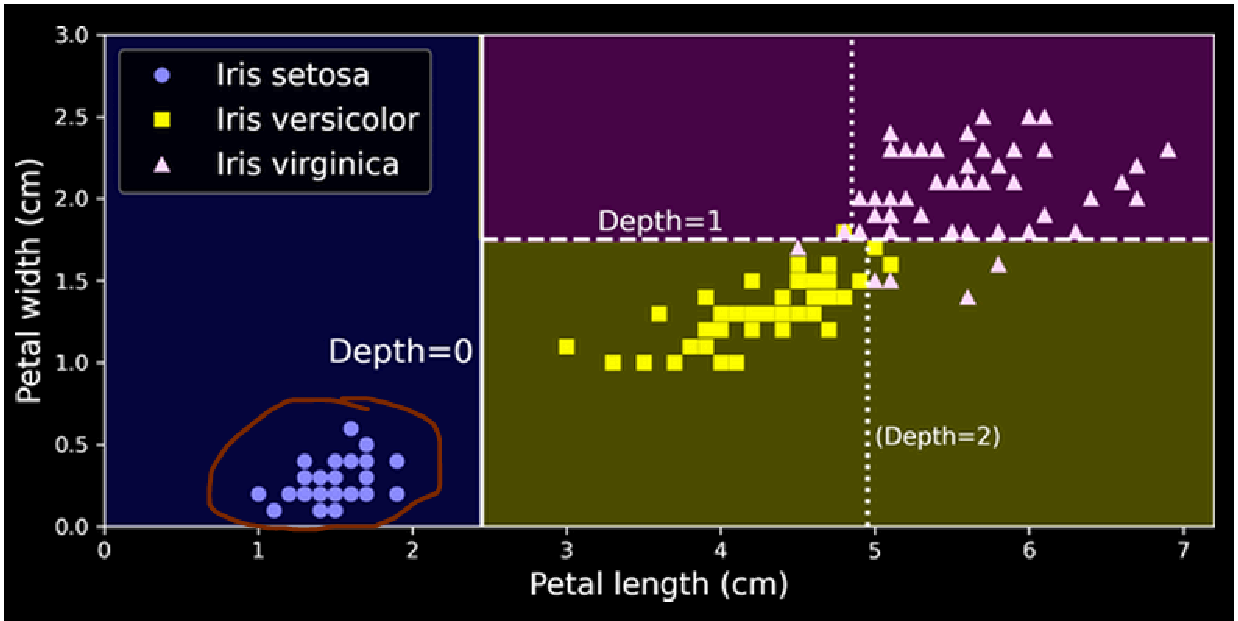
- 

Figure 6-2. Decision tree decision boundaries
- **CART algorithm** (used in Scikit-Learn) creates **binary trees**: each split produces **two branches** (yes/no decisions).
- At **depth 0**, the root splits data at *petal length = 2.45 cm*.
  - Left region is **pure** (*Iris setosa*) → no further splits.
  - Right region is **impure** → splits again at *petal width = 1.75 cm* (depth 1).
- With `max_depth=2`, the tree stops here. Increasing `max_depth` would add more splits and decision boundaries.
- **White Box vs Black Box Models**

- **Decision trees** are *white box models*: their decisions are **intuitive and interpretable**.
  - **Random forests & neural networks** are *black box models*: they often make accurate predictions but are **hard to interpret**.
  - **Interpretable ML** is an emerging field to help explain complex models' decisions (important for fairness and transparency).

Estimating Class Probabilities

- A decision tree can estimate the probability that an instance belongs to class *k*:
  1. Traverse the tree to find the **leaf node** for the instance.
  2. Compute the **ratio of class *k* instances** in that node.
- **Example**: For a flower with petals *(5 cm, 1.5 cm)* →
  - Probabilities:
    - *Iris setosa*: **0%** (0/54)
    - *Iris versicolor*: **90.7%** (49/54)
    - *Iris virginica*: **9.3%** (5/54)
  - Prediction: *Iris versicolor* (highest probability).

```
tree_clf.predict_proba([[5, 1.5]]) # Output: [[0.    , 0.907, 0.093]]
tree_clf.predict([[5, 1.5]]) # Output: [1]`
```

```

```

- ⚠ Limitation: Probabilities are **constant within each region**, even if some points seem closer to another class.

The CART Training Algorithm

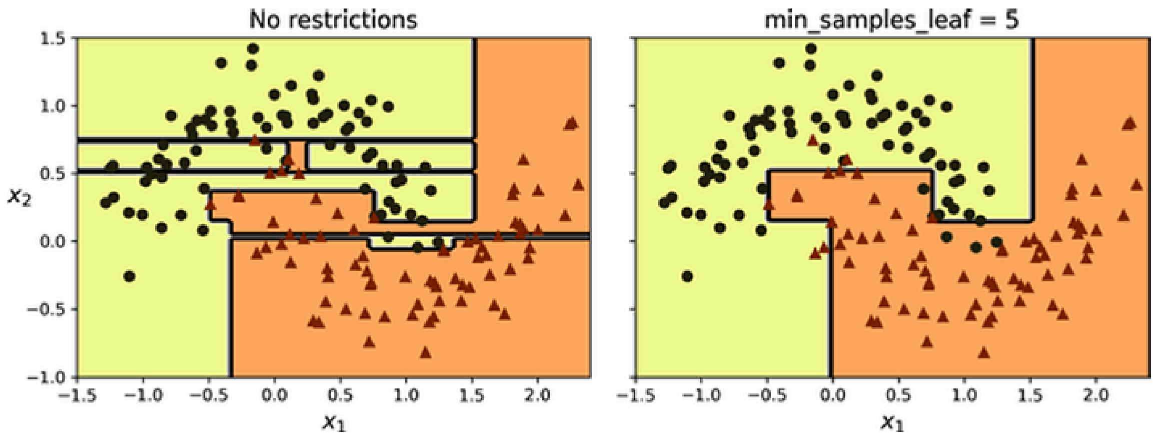
- Scikit-Learn uses **CART (Classification and Regression Tree)** to build decision trees.
- At each step, it:
    1. Splits the data into two subsets using a feature `k` and threshold `t` (e.g., *petal length ≤ 2.45 cm*).
    2. Chooses (`k, t`) that minimizes the **cost function**:
       $J(k,t) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$
        - $G_{\text{left/right}}$: impurity of the subsets.
        - $m_{\text{left/right}}$t: number of instances in each subset.
- **Algorithm Behavior**
    - **Recursively splits** the dataset based on best `(k, t)` pairs.
    - Stops when:
        - Maximum depth is reached (`max_depth`)
        - No split reduces impurity
        - Other stopping conditions triggered:
            - `min_samples_split`
            - `min_samples_leaf`
            - `min_weight_fraction_leaf`
            - `max_leaf_nodes`
- ⚠️ **Limitations**
    - CART is a **greedy algorithm**: it looks for the best immediate split without considering future impact.
    - **Not optimal**: finding the globally best tree is **NP-complete** and computationally infeasible.
    - So CART aims for a **"reasonably good" tree**, not the best possible one.

Regularization Hyperparameters

- **Decision Trees** are **nonparametric models**:

- They don't have a fixed number of parameters.
- They adapt closely to the training data, which can lead to **overfitting**.

- To reduce overfitting, **regularization** limits the tree's flexibility.
- **Key Regularization Hyperparameters (Scikit-Learn):**
  - `max_depth` : Maximum depth of the tree (default = unlimited).
  - `max_features` : Max features to consider for splits at each node.
  - `max_leaf_nodes` : Limits number of leaf nodes.
  - `min_samples_split` : Minimum samples required to split a node.
  - `min_samples_leaf` : Minimum samples required to create a leaf node.
  - `min_weight_fraction_leaf` : Like `min_samples_leaf` but as a fraction of total weights.
- **Tip**: Increasing `min_*` or reducing `max_*` → more regularization.
- **Pruning** (used in some algorithms):
  - Start with a fully grown tree, then **remove unnecessary nodes**.
  - A node is pruned if its improvement is **not statistically significant** (tested with p-values, e.g., chi-squared test).
  - Pruning stops when all such nodes are deleted.
- **Testing Regularization on the Moons Dataset**
  - Two decision trees were trained:
    1. **Unregularized** (default settings).
    2. **Regularized** with `min_samples_leaf=5` .
  - **Result (Figure 6-3):**



    - *Figure 6-3. Decision boundaries of an unregularized tree (left) and a regularized tree (right)*
    - The unregularized tree **overfits** the training data (complex decision boundary).
    - The regularized tree produces a **simpler boundary** and generalizes better.
  - **Test Set Accuracy:**
    - Unregularized tree: **89.8%**
    - Regularized tree: **92%** ✅

- **Key takeaway:** Regularization (e.g., `min_samples_leaf`) improves model generalization by reducing overfitting.

Regression

- Decision trees can also perform **regression tasks** using Scikit-Learn's `DecisionTreeRegressor`.
- Example: A **noisy quadratic dataset** is created, and a regression tree is trained with `max_depth=2`.

```
tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X_quad, y_quad)`
```

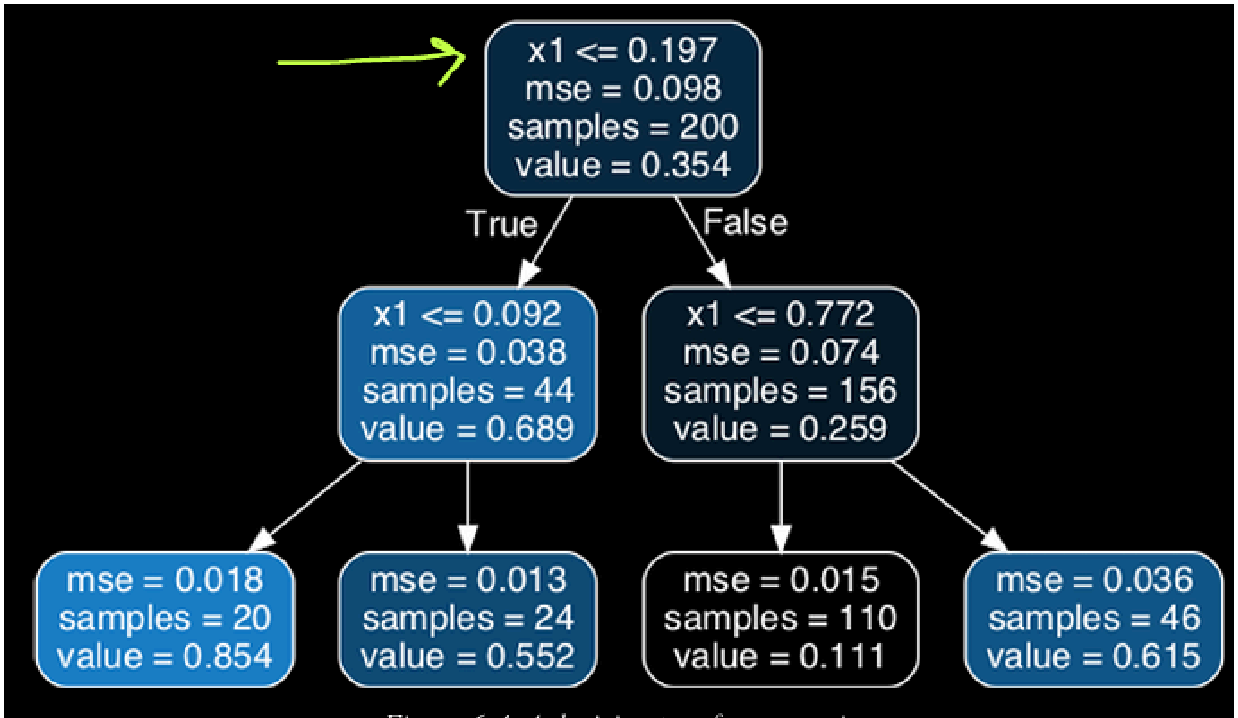
- In the resulting tree (Figure 6-4), the splits are similar to those in classification, but instead of predicting classes, each **leaf node predicts a numerical value**.
  - 

Figure 6-4. A decision tree for regression
- **How Prediction Works:**
  - For a new instance x=0.2:
    1. The root node asks if $x \leq 0.197$. Since $x = 0.2 > 0.197$, the algorithm moves to the **right child node**.
    2. Next, it asks if $x \leq 0.772$. Since $x = 0.2 < 0.772$, it moves to the **left child node.**
    3. This leaf predicts value=0.111, which is the **average target value** of the 110 training instances in that region.
  - The **Mean Squared Error (MSE)** of this prediction over those 110 instances is 0.015.
- **CART for Regression**

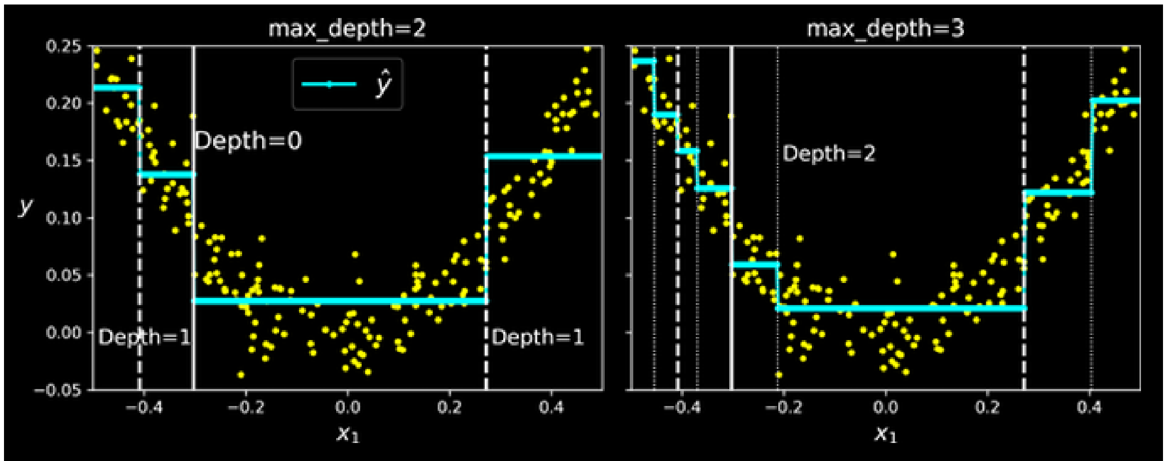
- The CART algorithm works similarly as in classification, but instead of minimizing impurity, it **minimizes MSE** at each split:
  - 

    Figure 6-5. Predictions of two decision tree regression models

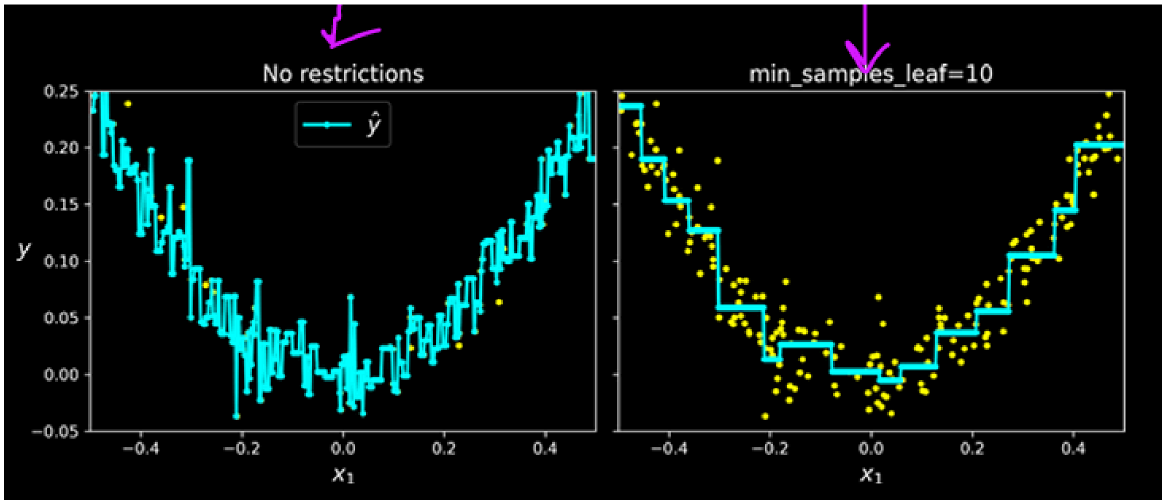
  $J(k, t_k) = \frac{m_{\text{left}}}{m} MSE_{\text{left}} + \frac{m_{\text{right}}}{m} MSE_{\text{right}}$
  where:
    - $MSE_{node} = \frac{1}{m_{node}} \sum_{i \in node} (y^{(i)} - \hat{y}_{node})^2$
    - $\hat{y}_{node}$: Mean target value in the node.
- ⚠️ **Overfitting in Regression Trees:**
  - Like in classification, regression trees are prone to **overfitting**, especially without regularization.
  - Example (Figure 6-6):
  - 

    Figure 6-6. Predictions of an unregularized regression tree (left) and a regularized tree (right)
    - **Unregularized tree** (left): fits training data too closely, leading to poor generalization.
    - **Regularized tree** (right): setting `min_samples_leaf=10` produces **smoother predictions** that generalize better.

Sensitivity to Axis Orientation

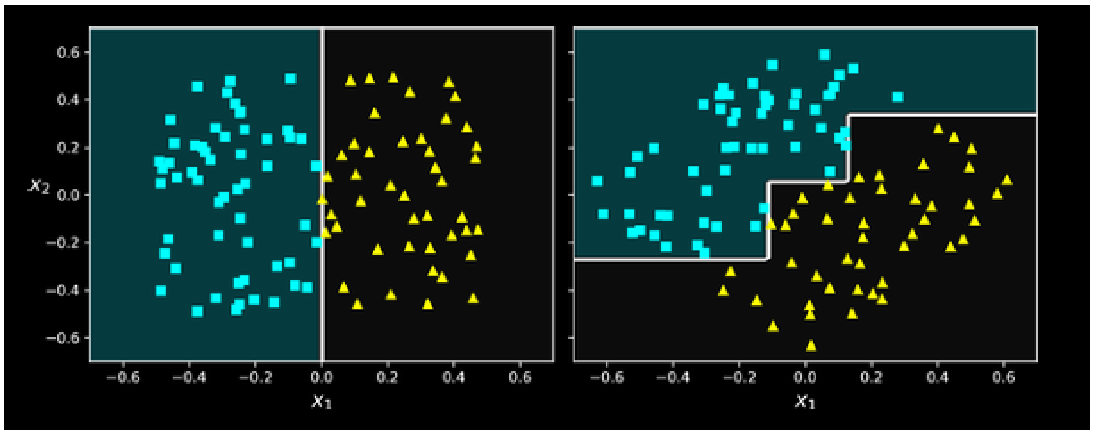
- **Strengths of Decision Trees**:
  - Easy to understand and interpret.
  - Simple to use, versatile, and powerful.
- **Limitation: Sensitivity to Orientation**
  - Decision trees naturally create **orthogonal decision boundaries** (splits perpendicular to feature axes).
  - This makes them **sensitive to the dataset's orientation**:
    - 📊 *Example (Figure 6-7)*:
      - 

      Figure 6-7. Sensitivity to training set rotation
      - On the **left**, the dataset is aligned with the axes → the tree splits cleanly.
      - On the **right**, the same dataset rotated by 45° results in a **complex and jagged decision boundary**, even though both trees perfectly fit the data.
    - ⚠ The rotated tree is more likely to **overfit** and generalize poorly.
- **Solution: Scaling + PCA**
  - To mitigate orientation sensitivity:
    1. **Scale the data** using `StandardScaler`.
    2. Apply **Principal Component Analysis (PCA)** to rotate the dataset and reduce feature correlations.
  - *Example (Figure 6-8)*:

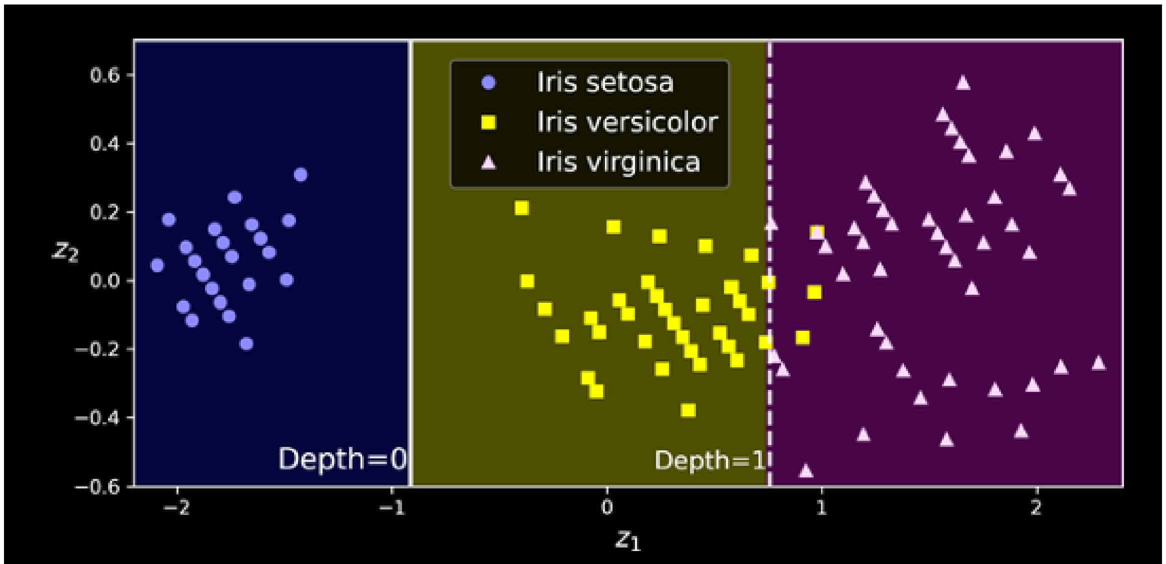
- 

Figure 6-8. A tree's decision boundaries on the scaled and PCA-rotated iris dataset
- After scaling and PCA rotation, the Decision Tree fits the dataset well using only a single principal component (z1z_1z1), a linear combination of petal length and width.

  - Code snippet for pipeline:

    ```
    from sklearn.decomposition import PCA
    from sklearn.pipeline import make_pipeline
    from sklearn.preprocessing import StandardScaler

    pca_pipeline = make_pipeline(StandardScaler(), PCA())
    X_iris_rotated = pca_pipeline.fit_transform(X_iris)

    tree_clf_pca = DecisionTreeClassifier(max_depth=2, random_state=42)
    tree_clf_pca.fit(X_iris_rotated, y_iris)`
    ```

- **Key Takeaway:**
  - While decision trees are powerful, their reliance on axis-aligned splits can lead to poor performance on rotated datasets.
  - Preprocessing with **scaling and PCA** can improve their ability to create simpler, more generalizable decision boundaries.

Decision Trees Have a High Variance

- **Decision Trees Have High Variance**
  - A key limitation of decision trees is their **high variance**:
    - Small changes in the **training data** or **hyperparameters** can result in very different tree structures.

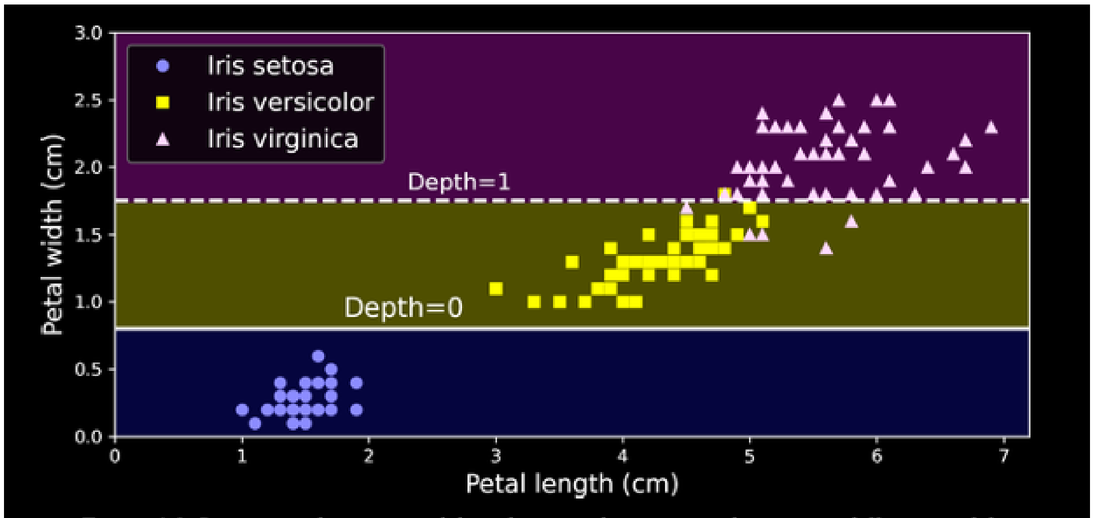
- Even retraining on the **same dataset** may produce a different tree because Scikit-Learn's training algorithm is **stochastic** (it randomly selects which features to evaluate at each node).
- Example: *Figure 6-9* shows a retrained tree that looks very different from the earlier tree in *Figure 6-2*.
  - 

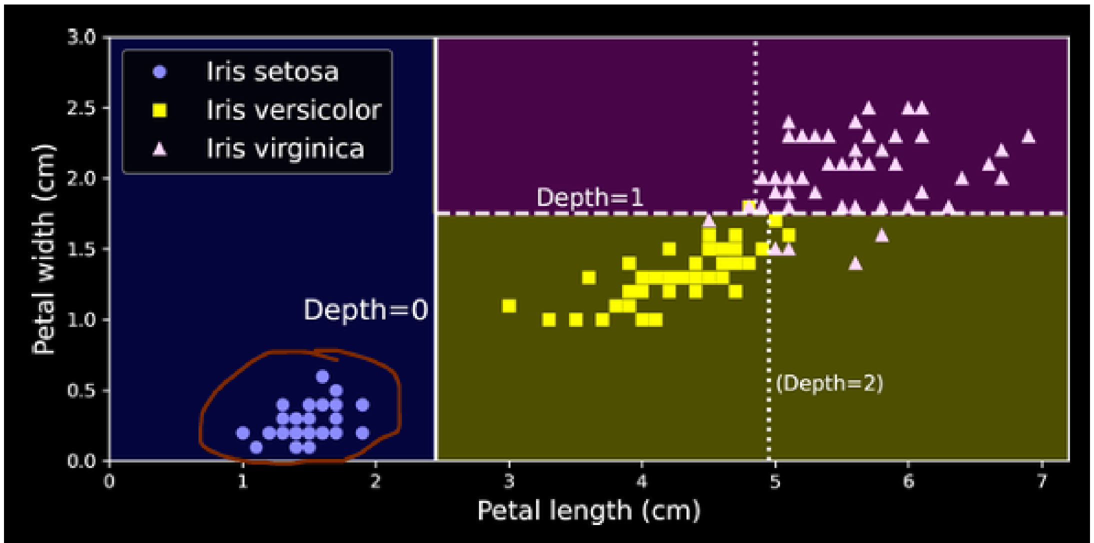
    Figure 6-9. Retraining the same model on the same data may produce a very different model
  - 

    Figure 6-2. Decision tree decision boundaries
- This variability can lead to **instability** in predictions.

- **Solution: Reduce Variance with Ensembles**
  - A powerful way to address high variance is to **combine multiple trees** into an ensemble.
  - Averaging predictions from many trees reduces variance significantly.
  - ✅ Such an ensemble is called a **Random Forest**, which combines the predictions of multiple trees for better stability and performance.
    📖 *Random Forests* will be covered in detail in **Chapter 7**.

Resources :

-

Related notes :

-

References :

- **Internal :**
  -
  -
  -
- **External :**
  - [hegab videos]()
  - [the book]()
  - [the notebook]()
  -