# Decision Tree Classifier: Implementation Analysis & Results

## Implementation Approach

The decision tree was implemented from scratch using two fundamental classes:

- **Node class**: Represents each node of the tree, storing the split feature index, threshold, left and right children, and—at leaves—the predicted class. Leaf status is checked via an `is_leaf_node` method.
- **DecisionTreeClassifier class**: Handles tree construction and prediction. The `fit` method initiates recursive tree building. The `_grow_tree` method recursively:
  - Checks stopping criteria (`max_depth`, minimum samples, or purity).
  - Finds the best split using the `_best_split` method, which iterates through all features and unique thresholds, selecting the split with the highest information gain.
  - Constructs left and right child nodes accordingly.

Split quality is evaluated using Gini impurity (or entropy, if implemented) and information gain, aligning each decision node with the most informative split available.

## Overfitting Analysis: The Role of max_depth

The model was evaluated with different values of `max_depth`, recording training and testing accuracy.

| max_depth | Training Accuracy | Testing Accuracy |
|---|---|---|
| 1 | 0.6667 | 0.6333 |
| 2 | 0.9500 | 0.9333 |
| 3 | 0.9583 | 0.9667 |
| 5 | 1.0000 | 0.9333 |
| 10 | 1.0000 | 0.9000 |

As depth increases, training accuracy rises—eventually perfect (1.0)—but testing accuracy first rises then drops beyond a certain depth, illustrating overfitting. With shallow trees, both train and test accuracy are low (underfitting). As the tree deepens, the training set is learned in greater detail, but the gap between training and testing accuracy widens past an optimal point.

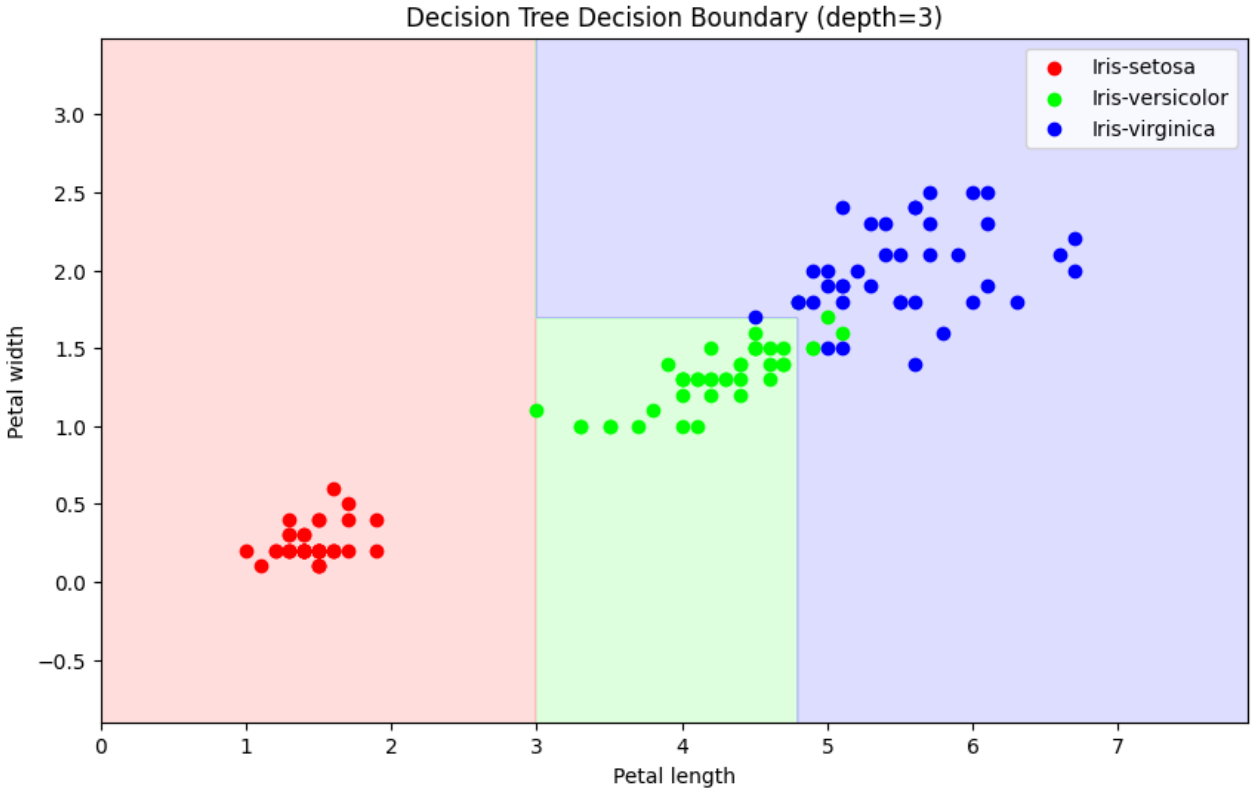
## Printed Tree Structure and Decision Path

### Tree Structure (Depth=3, Key Features):

```
Feature 2 < 3.000?
--> True:
 Predict: Iris-setosa
--> False:
 Feature 2 < 4.800?
 --> True:
  Feature 3 < 1.700?
  --> True:
   Predict: Iris-versicolor
  --> False:
   Predict: Iris-virginica
 --> False:
  Feature 3 < 1.800?
  --> True:
   Predict: Iris-virginica
  --> False:
   Predict: Iris-virginica
```

- Here, `Feature 2` is petal length, and `Feature 3` is petal width (zero-based indexing).

### Sample Decision Path Example:



- For a data point with petal length = 4.0 and petal width = 1.2:
    a. 4.0 < 3.0? No → Go to right branch.
    b. 4.0 < 4.8? Yes → Go to left branch.
    c. 1.2 < 1.7? Yes → Predict: Iris-versicolor.

### Analysis of Decision Boundary Plot

The decision boundary plot below shows how the decision tree partitions the 2D feature space (`petal_length` vs. `petal_width`) for three Iris classes:

- **Interpretation**: Each region corresponds to a leaf node's prediction, and the boundaries are axis-aligned, reflecting the tree's splits. Each class (color) occupies a "rectangle" in the plot. For example, all points to the left of petal length = 3.0 are classified as Iris-setosa, matching the first split in the tree.
- The hierarchical nature of splits can be seen: regions are carved according to recursive, rectangular thresholds, not diagonal or curved lines.

### Summary Table: Training vs. Testing Accuracy (Various Depths)

| max_depth | Training Accuracy | Testing Accuracy |
|---|---|---|
| 1 | 0.6667 | 0.6333 |
| 2 | 0.9500 | 0.9333 |
| 3 | 0.9583 | 0.9667 |
| 5 | 1.0000 | 0.9333 |
| 10 | 1.0000 | 0.9000 |

**Conclusion:**

This experiment demonstrates that careful tuning of tree depth yields the best generalization results. The printed

tree and decision boundary together make the classifier's logic transparent, and the overfitting pattern is quantitatively observed as the accuracy gap widens with a deeper tree.