Post Pruning And Pre Pruning

- **Pruning** is a technique used to simplify Decision Trees and prevent **overfitting**.

1. **Pre-Pruning (Early Stopping)**:
    - Stop the tree from growing too deep during the training process by setting constraints.
    - Bigger datasets
2. **Post-Pruning**:
    - Allow the tree to grow fully and then trim it back to remove unnecessary branches.
    - Smaller datasets

1. Pre-Pruning (Early Stopping)

Pre-pruning involves setting constraints **before** the tree is fully grown. These constraints prevent the tree from becoming too complex. Common constraints include:

Key Parameters for Pre-Pruning

1. `max_depth`:
    - Limits the maximum depth of the tree.
    - Example: `max_depth=3` means the tree can have at most 3 levels.
2. `min_samples_split`:
    - Specifies the minimum number of samples required to split a node.

- Example: `min_samples_split=10` means a node must have at least 10 samples to be split.

3. `min_samples_leaf`:
   - Specifies the minimum number of samples required to be at a leaf node.
   - Example: `min_samples_leaf=5` means each leaf node must have at least 5 samples.
4. `max_leaf_nodes`:
   - Limits the maximum number of leaf nodes in the tree.
   - Example: `max_leaf_nodes=10` means the tree can have at most 10 leaf nodes.
5. `min_impurity_decrease`: (IMP)
   - Stops splitting if the impurity decrease is below a certain threshold.
   - Example: `min_impurity_decrease=0.01` means a split must reduce impurity by at least 0.01

2. Post-Pruning

`tree.cost_complexity_pruning_path(X_train, y_train)`

Post-pruning involves growing the tree fully and then trimming it back to remove unnecessary branches. This is done by removing branches that contribute little to the model's performance.

How Post-Pruning Works

1. **Grow the Tree Fully**:
   - Allow the tree to grow without any constraints.
2. **Calculate `Cost-Complexity`**:
   - Use a metric called **Cost-Complexity Pruning** (also known as **Weakest Link Pruning**) to evaluate the trade-off between tree complexity and accuracy.

3. **Trim the Tree**:
    - Remove branches that do not significantly improve the model's performance.

Key Parameter for Post-Pruning

1. `ccp_alpha`:
    - Controls the amount of pruning.

> 💡
> - **A higher `ccp_alpha` results in more pruning.**

    - Example: `ccp_alpha=0.01` will prune more aggressively than `ccp_alpha=0.001`.

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X = data.data  # Features
y = data.target  # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
e=42)

# Create a Decision Tree model without pruning
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

# Make predictions before pruning
y_pred_before = tree.predict(X_test)
print("Accuracy before pruning:", accuracy_score(y_test, y_pred_before))
```

```
# Perform post-pruning using Cost-Complexity Pruning
path = tree.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas  # Alpha values for pruning

# Train a pruned tree
pruned_tree = DecisionTreeClassifier(ccp_alpha=ccp_alphas[-2])  # Use the s
econd-to-last alpha
pruned_tree.fit(X_train, y_train)

# Make predictions after pruning
y_pred_after = pruned_tree.predict(X_test)
print("Accuracy after pruning:", accuracy_score(y_test, y_pred_after))
```

```
Accuracy before pruning: 1.0
Accuracy after pruning: 0.7111111111111111
```

What is `cost_complexity_pruning_path`?

The `cost_complexity_pruning_path` method computes the **effective alphas** (values of `ccp_alpha`) that can be used for pruning. It returns two things:

1. `ccp_alphas`: A list of alpha values that can be used for pruning.
2. `impurities`: A list of impurity values corresponding to each alpha.

```
path.ccp_alphas
```

```
array([0.        , 0.00923521, 0.01269841, 0.01269841, 0.01847042,
       0.02705804, 0.25028684, 0.31210884])
```

- `tree.cost_complexity_pruning_path(X_train, y_train)`:
  - This method calculates the effective `ccp_alpha` values for pruning.

- It returns an object (`path`) that contains the list of `ccp_alphas` and corresponding impurities.

- **`ccp_alphas = path.ccp_alphas`**:
  - Extracts the list of alpha values from the `path` object.
  - These alpha values represent different levels of pruning, from no pruning (`ccp_alpha=0`) to maximum pruning.

```
pruned_tree = DecisionTreeClassifier(ccp_alpha=ccp_alphas[-2])  # Use the s
econd-to-last alpha
pruned_tree.fit(X_train, y_train)
```

- **`ccp_alphas[-2]`**:
  - The `ccp_alphas` list is sorted in ascending order, so the last value (`ccp_alphas[-1]`) corresponds to the most aggressive pruning (a tree with just one node).
  - The second-to-last value (`ccp_alphas[-2]`) is chosen because it represents a good balance between pruning and retaining useful splits.

- **`DecisionTreeClassifier(ccp_alpha=ccp_alphas[-2])`**:
  - Creates a new Decision Tree model with the chosen `ccp_alpha` value.
  - This value controls how much pruning will be applied to the tree.

- **`pruned_tree.fit(X_train, y_train)`**:
  - Trains the pruned tree on the training data.

Why Use `ccp_alphas[-2]`?

- The `ccp_alphas` list is sorted from smallest to largest.
  - The **first value** (`ccp_alphas[0]`) corresponds to **no pruning** (the full tree).

    - The **last value** (`ccp_alphas[-1]`) corresponds to **maximum pruning** (a tree with just one node).
- The **second-to-last value** (`ccp_alphas[-2]`) is often a good choice because:
    - It prunes the tree significantly but still retains useful splits.
    - It avoids overfitting while maintaining good accuracy.

Pre-Pruning vs. Post-Pruning

Aspect	Pre-Pruning	Post-Pruning
When It Happens	During tree construction.	After the tree is fully grown.
Complexity	Simpler to implement.	More complex to implement.
Control	Directly control tree growth.	Allows the tree to grow fully first.
Risk of Overfitting	May underfit if constraints are too strict.	Reduces overfitting by trimming the tree.
Flexibility	Less flexible.	More flexible.