Pruning and Overfitting

The Problem: Overfitting in Decision Trees

- **Unpruned trees**: Can grow very deep and complex
- **Perfect training accuracy**: Each leaf contains single training example
- **But**: Poor generalization to new data
- **Symptoms**:
  - High training accuracy, low test accuracy
  - Very deep trees with many leaves
  - Rules that are too specific to training data
- **Solution**: Pruning to control model complexity

Pre-pruning (Early Stopping)

Stop growing tree before it becomes too complex:

- **Maximum depth**: Limit tree depth (e.g., max_depth $= 5$)
- **Minimum samples per split**: Don't split if node has $<$ N samples
- **Minimum samples per leaf**: Ensure each leaf has $\geq$ M samples
- **Maximum features**: Consider only subset of features at each split
- **Minimum impurity decrease**: Only split if improvement $>$ threshold

Advantages: Simple, computationally efficient
Disadvantages: May stop too early, miss good splits later

Post-pruning (Tree Simplification)

Grow full tree, then remove unnecessary branches:

- **Algorithm**:
  1. Grow complete tree on training data
  2. Use validation set to evaluate subtree performance
  3. Remove branches that don't improve validation accuracy
  4. Repeat until no beneficial removals remain
- **Cost Complexity Pruning**: Minimize Error $+ \alpha \times$ Tree Size
- **Advantages**: More thorough, can recover from early stopping mistakes
- **Disadvantages**: More computationally expensive

Cost Complexity Pruning Algorithm

Systematic approach to find optimal tree size:

- **Cost function**: $R_\alpha(T) = R(T) + \alpha|T|$
  - $R(T)$: Total impurity (training error)
  - $|T|$: Number of leaves
  - α: Complexity penalty parameter
- **Weakest Link**: At each pruning step, compute:

$$g(t) = \frac{R(t) - R(T_t)}{|T_t| - 1}$$

  - $g(t)$: The α value at which subtree rooted at node t should be pruned
  - $R(t)$: Impurity of node t, treating it as leaf node
  - $R(T_t)$: Total impurity of subtree rooted at node t
  - $|T_t|$: Number of leaves in subtree rooted at node t

Cost Complexity Pruning: Algorithm Steps

Iterative pruning process:

- **Process**:
  1. Start with full tree ($\alpha = 0$)
  2. Compute $g(t)$ for all internal nodes
  3. Prune node with smallest $g(t)$ (weakest link)
  4. Repeat until only root remains
  5. Use cross-validation to select optimal α

Bias-Variance Trade-off in Trees

- **Unpruned trees**:
  - Low bias (can fit complex patterns)
  - High variance (sensitive to training data changes)
  - Prone to overfitting
- **Heavily pruned trees**:
  - High bias (may miss important patterns)
  - Low variance (more stable predictions)
  - Risk of underfitting
- **Optimal pruning**: Balances bias and variance
- **Cross-validation**: Essential for finding this balance

Practical Pruning Guidelines

- **Start simple**: Begin with restrictive pre-pruning parameters
- **Cross-validation**: Always use CV to select pruning parameters
- **Validation curves**: Plot training/validation error vs. tree complexity
- **Common parameters** (sklearn):
  - `max_depth`: Start with 3-10
  - `min_samples_split`: Try 10-100
  - `min_samples_leaf`: Try 5-50
  - `ccp_alpha`: Use for cost complexity pruning
- **Domain knowledge**: Consider interpretability requirements