Decision Trees

Introduction

Outline

- **Decision Trees vs. Regression**
- Tree Building and Pruning Process
- Classification Trees vs. Regression Trees
- Application of Trees

Regression

- Regression models are well-tested, extensively studied, robust and work even with moderate violations of the assumptions
- However, with large number of variables regression becomes difficult to use when
    - Relationships are non-linear
    - Variables interact

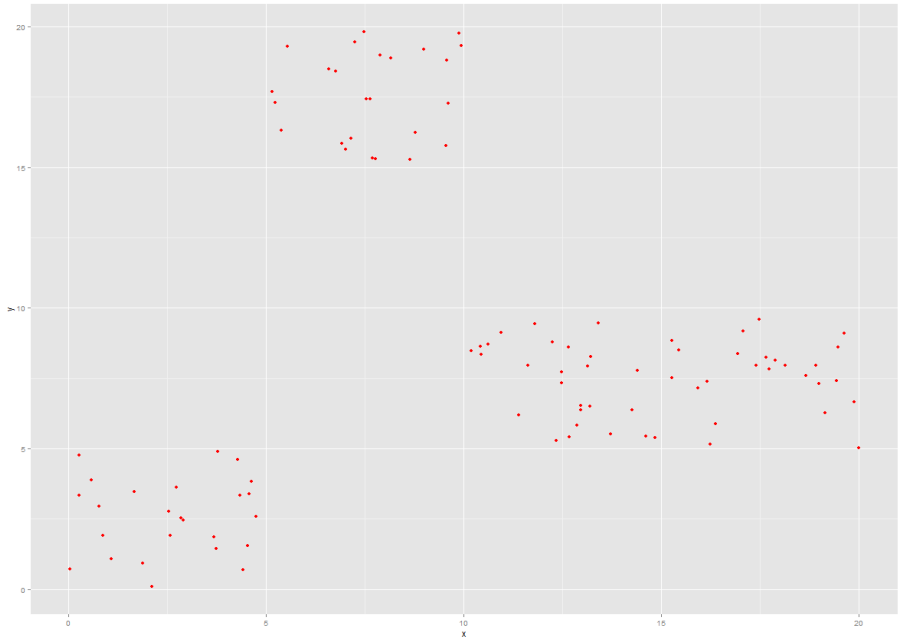
Decision Trees

- Trees are particularly useful when
    - Using large number of independent variables and
    - When there are likely to be non-linear relationships and interactions amongst variables.
    - And when interpretability of results is important
- At the same time, trees often
    - Perform poorly in predicting relative to the best supervised learning approaches
    - Tend to over-fit the data. Fortunately, this can be remedied by cross-validation.

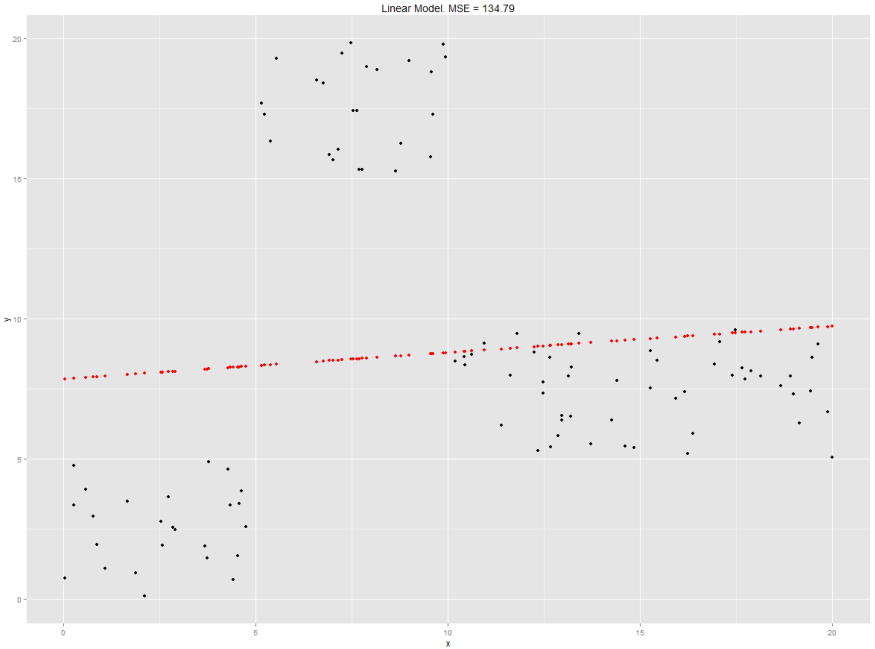
Decision Trees

- These involve stratifying or segmenting the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods.
- These are also referred to as CART (Classification and Regression Tree) after the family of models used for tree-algorithms
- They can be used for metric (prediction problem) and non-metric (classification problem) outcomes

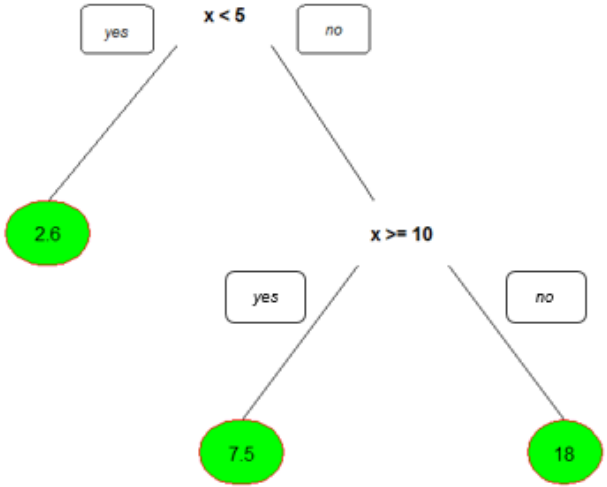
Scatter Plot of Simulated Data



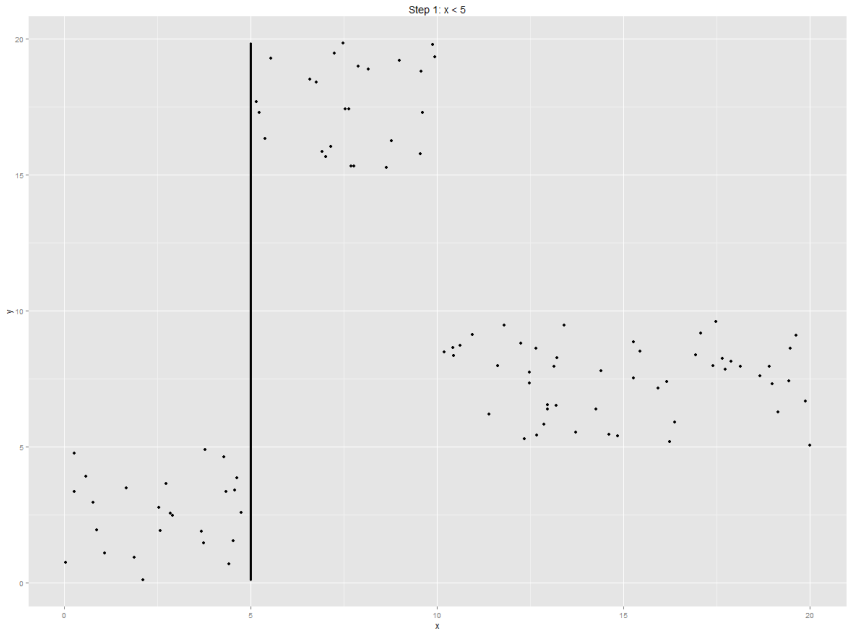
Fitting a Linear Model



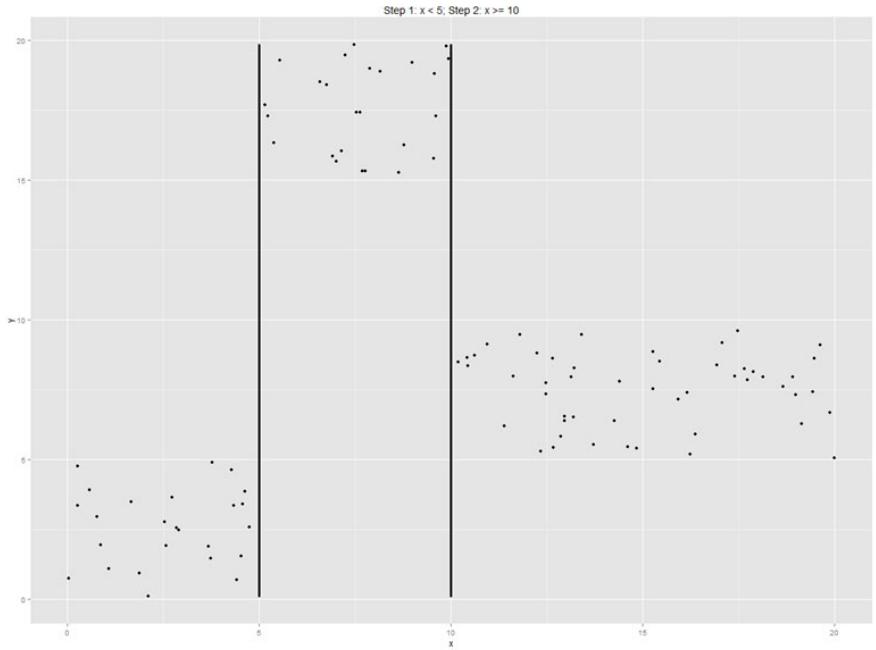
Tree Model Plot



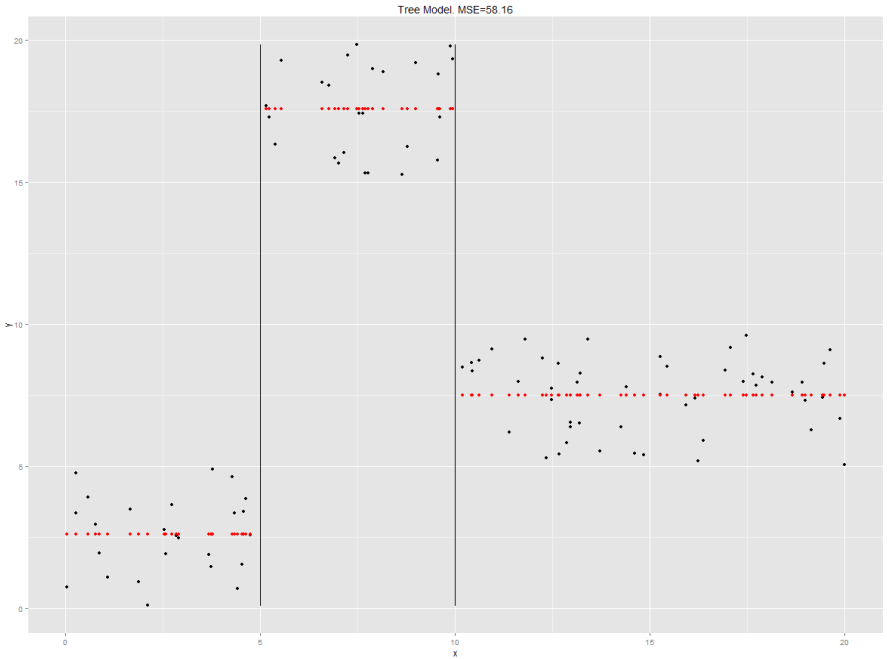
Fitting Process



Fitting Process



Fitting a Tree



Linear Model Vs Tree

Terminology

- Trees are drawn as an inverted tree
- The first split point is called the Root Node
- Nodes below the Root Node are interior nodes and are linked to the root node by Branches
- The end nodes are called Leaves

Outline

- Decision Trees vs. Regression
- **Tree Building and Pruning Process**
- Classification Trees vs. Regression Trees
- Application of Trees

Tree Building Process

- We divide the predictor (independent variables) space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
- For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j .

Tree Building Process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize a function of MSE, given by

Equation 6-4. CART cost function for regression

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

- $\hat{y}_{node}$ is the mean response for the training observations within the specific box (node).

Tree Building Process

- We first select the predictor Xj (or call it k) and the cutpoint such that splitting the predictor space into the regions $\{X|k < t_k\}$ and $\{X|k \geq t_k\}$ leads to the greatest possible reduction in $J(k,t_k)$.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the $J(k,t_k)$ within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the $J(k,t_k)$. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Tree Building Process

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a top-down, greedy approach that is known as recursive binary splitting.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

Pruning a Tree

- The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. Why?
- A smaller tree with fewer splits (that is, fewer regions R1, . . . , R_J) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the J due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too short-sighted: a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in J later on.

Pruning a Tree

- A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree
- Cost complexity pruning — also known as weakest link pruning — is used to do this
- We consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree T ⊂ T0 such that

$$\sum_{m=1}^{|T|} \quad \sum_{i \in Rj} (yi - \hat{y}_{node})^2 + \alpha|T|$$

- is as small as possible. Here |T| indicates the number of terminal nodes of the tree T, Rm is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{node}$ is the mean of the training observations in Rm.

Choosing the Best Tree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: Tree Algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. For each k = 1, . . . , K:

    a) Repeat Steps 1 and 2 on the 1…K−1th fraction of the training data, excluding the kth fold.
    b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.
    c) Average the results, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Regularization hyperparameters

- Hyper parameters that you can adjust to regularize your decision tree in Scikit-Learn to avoid overfitting:

  - max_depth
  - min_samples_split (the minimum number of a samples a node must have before it can e split)
  - min_samples_leaf (the minimum number of samples a leaf node must have)
  - min_weight_fraction_leaf (same as min_samples_leaf but expressed as a fraction of the total number of weighted instances)
  - max_leaf_nodes (the maximum number of leaf nodes)
  - max_features (the maximum number of features that are evaluated for splitting at each node)

Regression Tree Example - 1

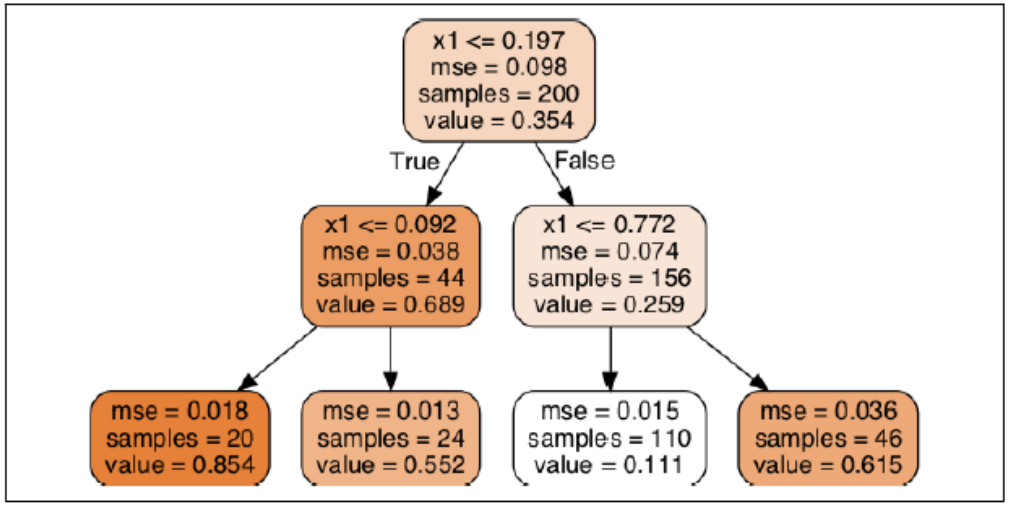


Figure 6-4. A Decision Tree for regression

```python
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2)
tree_reg.fit(X, y)
```

Regression Tree Example - 2

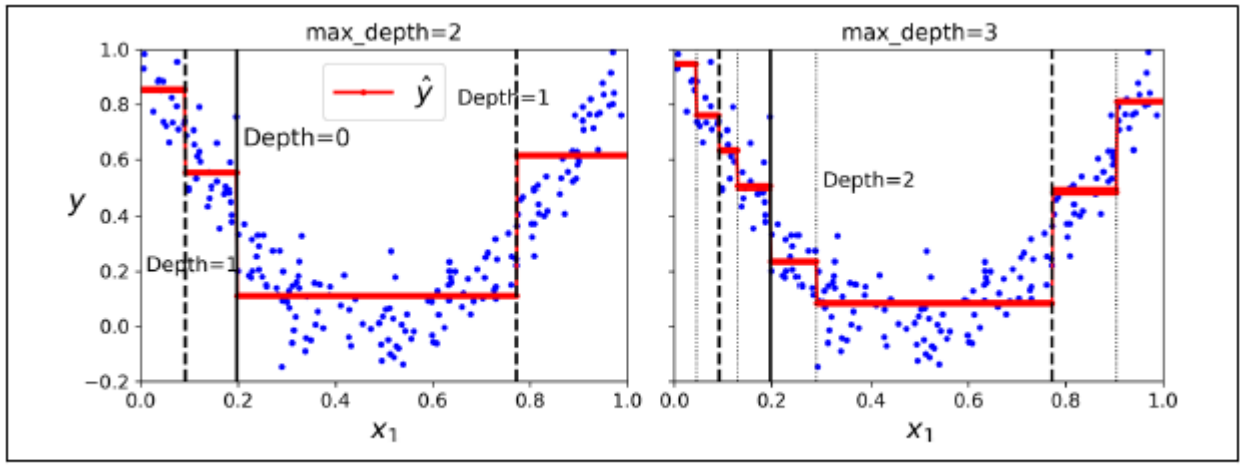


Figure 6-5. Predictions of two Decision Tree regression models

Regression Tree Example - 3

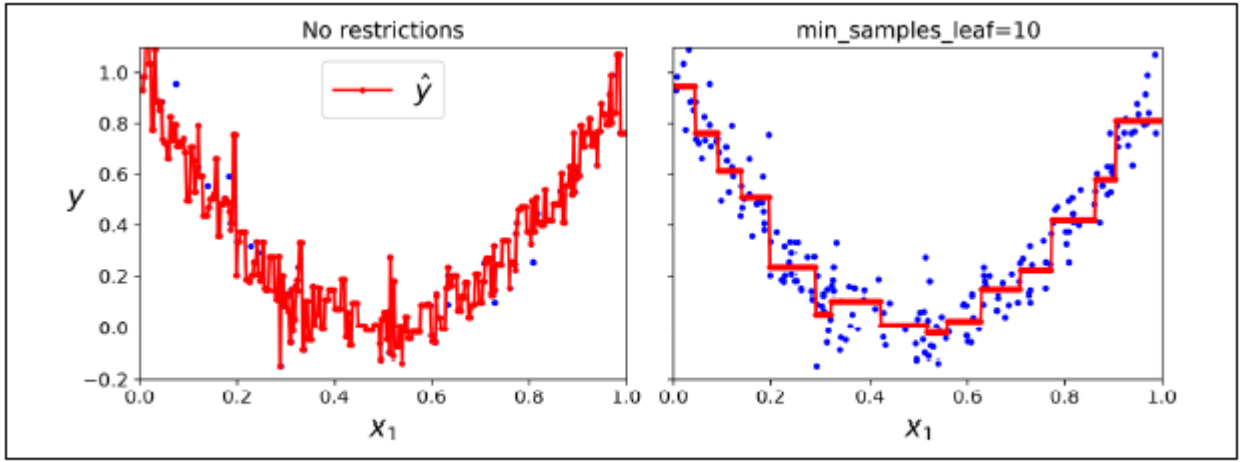


Figure 6-6. Regularizing a Decision Tree regressor

Outline

- Decision Trees vs. Regression
- Tree Building and Pruning Process
- **Classification Trees** vs. Regression Trees
- **Application of Trees**

Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

Classification Tree Algorithm

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, MSE cannot be used as a criterion for making the binary splits
- The cost function is defined as below:

Equation 6-2. CART cost function for classification

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

where $\begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset.} \end{cases}$

Classification Tree Algorithm

- The Gini index/Gini impurity is defined by

Equation 6-1. Gini impurity

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

In this equation:

- $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node.

- A measure of total variance across the K classes. The Gini index takes on a small value if all of the $P_{i,k}$ are close to one.
- For this reason the Gini index is referred to as a measure of node purity — a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is cross-entropy, given by

Equation 6-3. Entropy

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

- It turns out that the Gini index and the cross-entropy are very similar numerically.

Classification Tree Example - 1

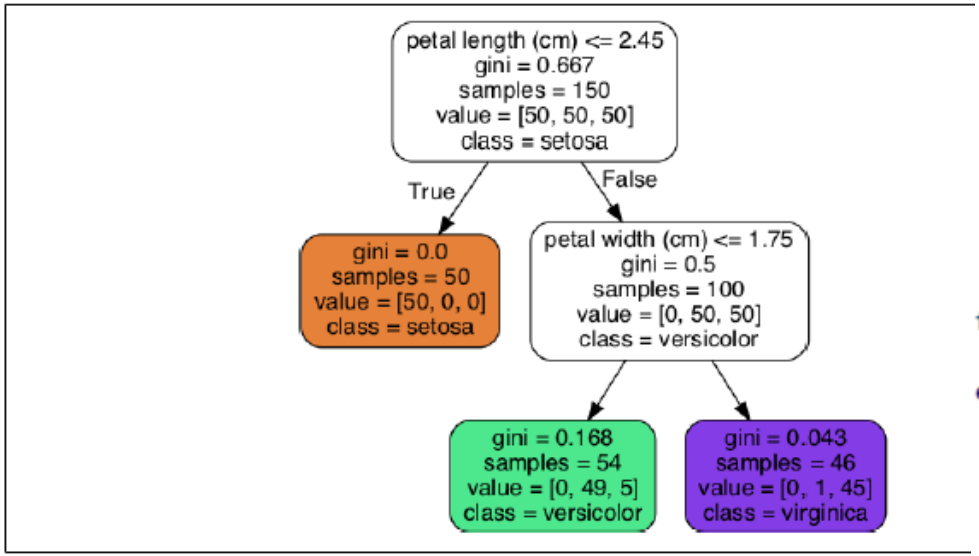


Figure 6-1. Iris Decision Tree

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)
```

```
from sklearn.tree import export_graphviz

export_graphviz(
        tree_clf,
        out_file=image_path("iris_tree.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )
```

Classification Tree Example - 2

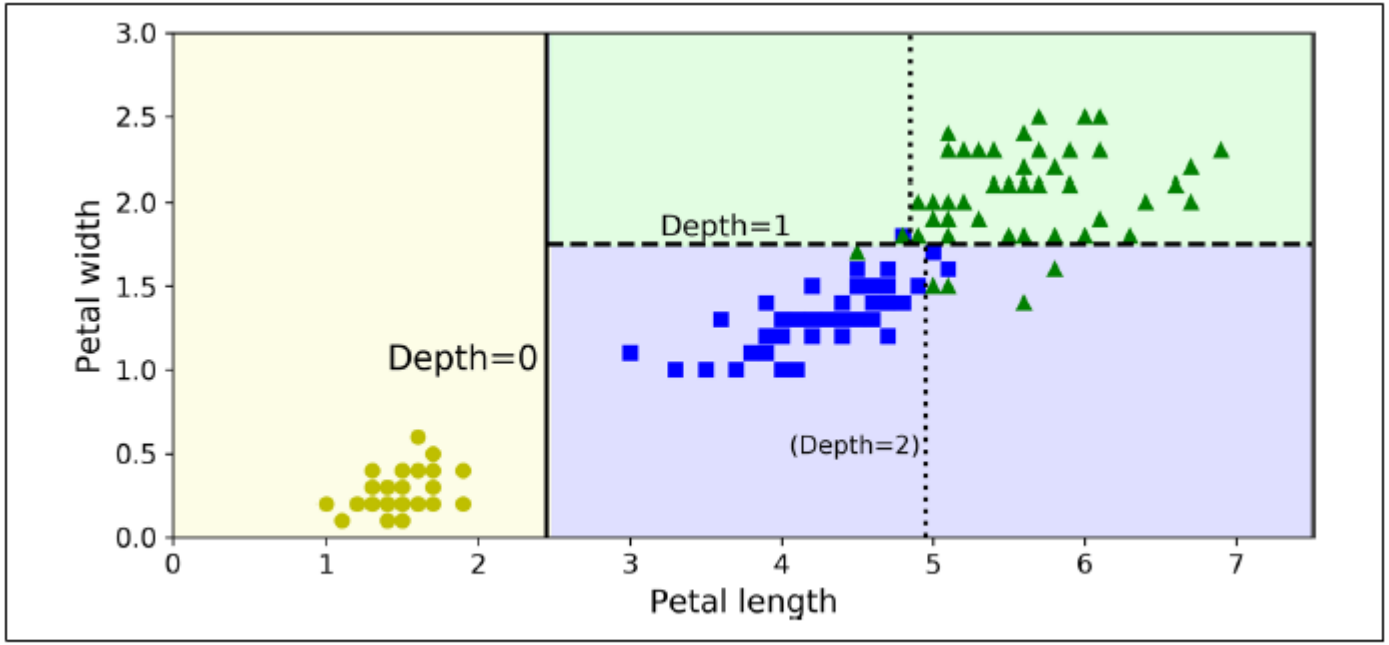


Figure 6-2. Decision Tree decision boundaries

Classification Tree Example - 3

- Regularization example

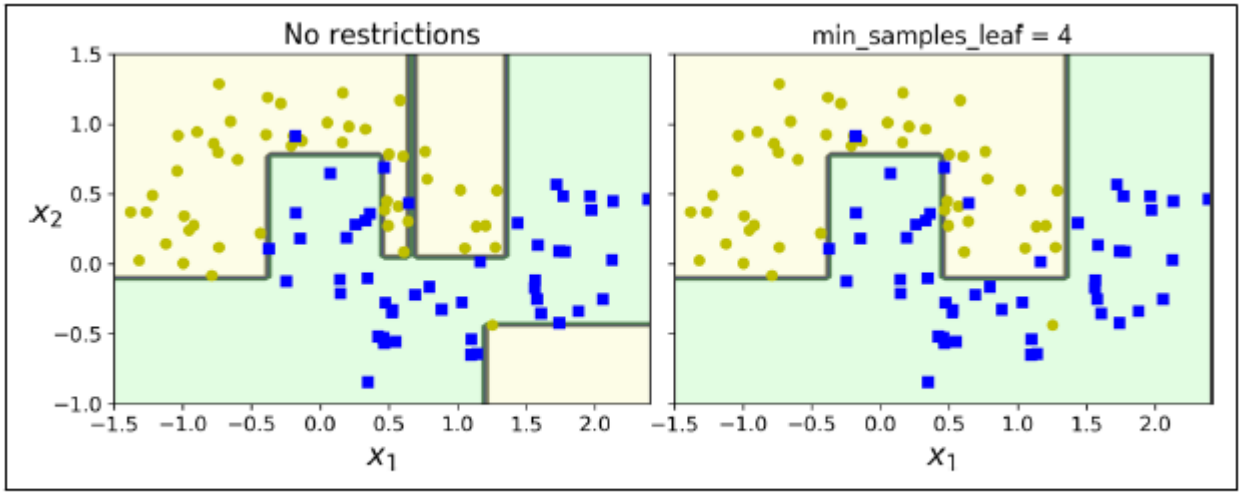


Figure 6-3. Regularization using `min_samples_leaf`

Advantages and Disadvantages of Trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.
- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other prediction and classification approaches. However, the predictive performance of trees can be substantially improved by aggregating many decision trees.

Summary

- Trees are seldom the best models
- But, they are the simplest to understand
- Trees can be greatly improved by tuning complexity and considering an ensemble of trees