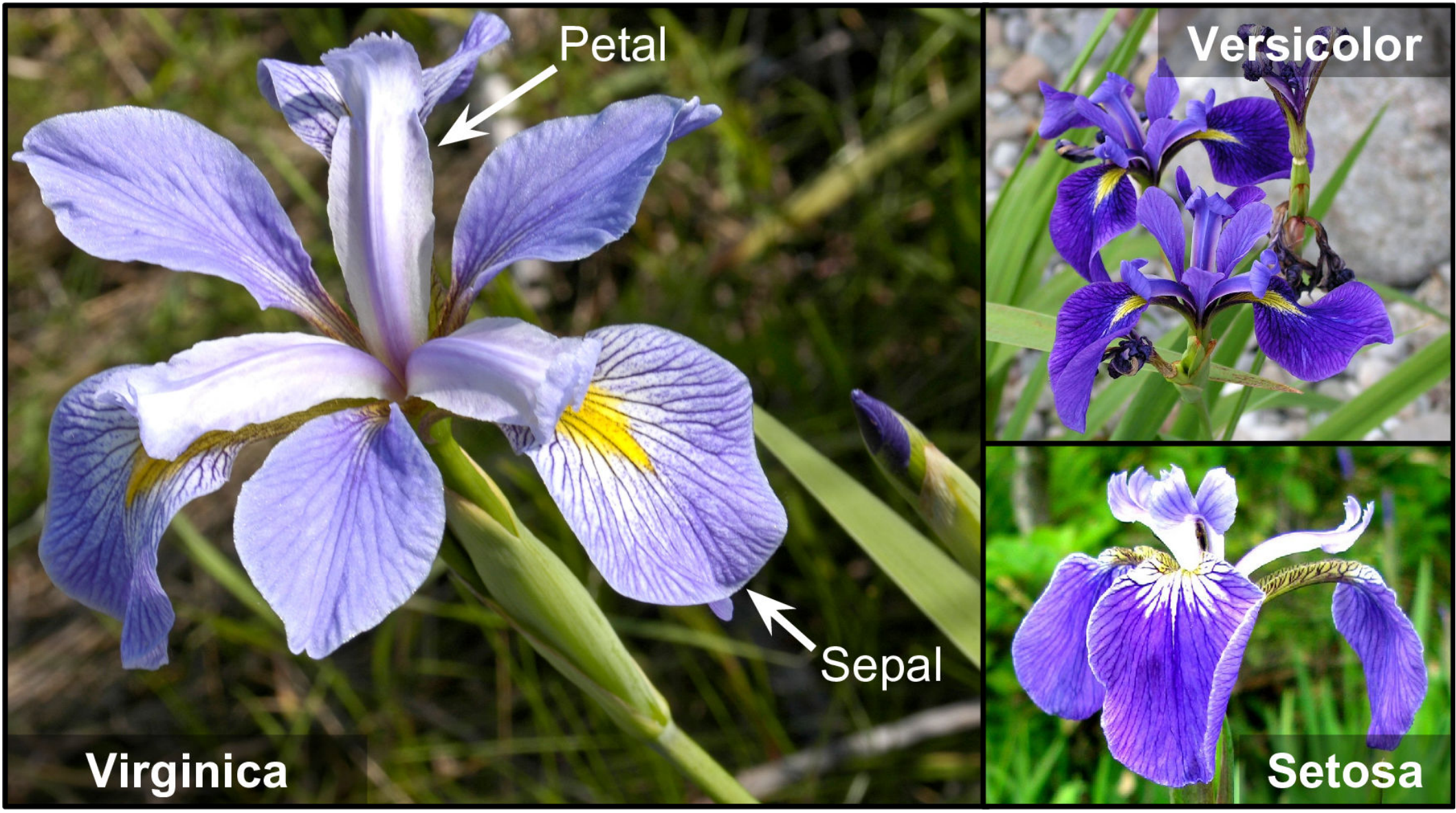
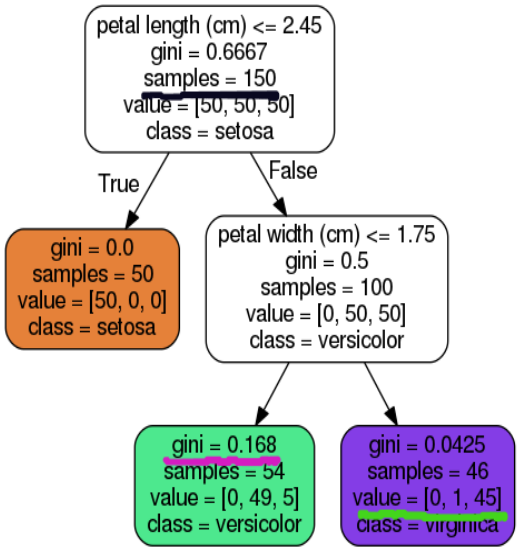
Decision Trees

Decision Tree:Data set



Decision Tree:Data set

```
iris = load_iris()
print("list(iris.keys()):",list(iris.keys()))
print("iris.DESCR:",iris.DESCR)

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

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
print("tree_clf:",tree_clf.fit(X, y))
export_graphviz(
        tree_clf,
        out_file=fl.image_path("iris_tree.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )
```



- sample attribute counts how many training instances it applies to.
- value attribute counts how many training instances each class of this applies to. 0 for IS, 1 for iris Versicolor and 45 iris virginica.
- Despite the criteria of PW <= 1.75, 45/46 labels were correct.
- $1 - (0/54)^2 - (49/54)^2 - (5/54)^2 = 0.168$

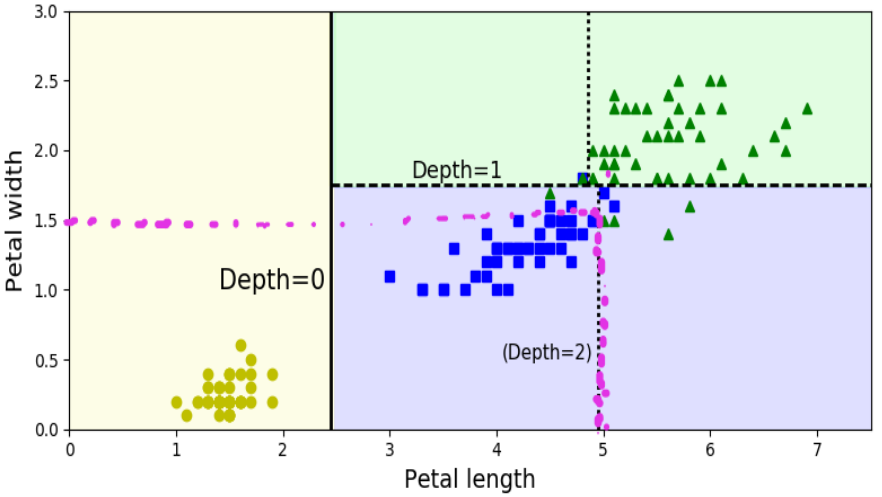
Decision Tree:Decision Boundary

```
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X, y)

print("DT predict_proba[5, 1.5]:",tree_clf.predict_proba([[5, 1.5]]))
print("DT predict[5, 1.5]:",tree_clf.predict([[5, 1.5]]))

def plot_decision_boundary(clf, X, y, axes=[0, 7.5, 0, 3], iris=True, legend=False, plot_training=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap, linewidth=10)
    if not iris:
        custom_cmap2 = ListedColormap(['#7d7d58','#4c4c7f','#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    if plot_training:
        plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", label="Iris-Setosa")
        plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", label="Iris-Versicolor")
        plt.plot(X[:, 0][y==2], X[:, 1][y==2], "g^", label="Iris-Virginica")
        plt.axis(axes)
    if iris:
        plt.xlabel("Petal length", fontsize=14)
        plt.ylabel("Petal width", fontsize=14)
    else:
        plt.xlabel(r"$x_1$", fontsize=18)
        plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
    if legend:
        plt.legend(loc="lower right", fontsize=14)
```

- Max depth was specified as 2. If it was 3 then right-side would have split along the dotted line.

```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/decisiontrees/decisionboundary.py
DT predict_proba[5, 1.5]: [[ 0.          0.90740741  0.09259259]]
DT predict[5, 1.5]: [1]
```

- It can also predict probability.
- For PL = 5 and PW of 1.5, it is Iris Versicolor.



Decision Tree:Model Interpretation

- White Box Versus Black Box
  - As you can see Decision Trees are fairly intuitive and their decisions are easy to interpret. Such models are often called **white box** models.
  - In contrast, as we will see, Random Forests or neural networks are generally considered **black box** models.
  - They make great predictions, and you can easily check the calculations that they performed to make these predictions; nevertheless, it is usually hard to explain in simple terms why the predictions were made.

Decision Tree:Training Algorithm

- Classification And Regression Tree (CART) algorithm to train Decision Trees (also called “growing” trees).
- The idea is really quite simple: the algorithm first splits the training set in two subsets using a single feature k and a threshold tk (e.g., “petal length ≤ 2.45 cm”). **How does it choose k and tk?**
- It searches for the pair (k, tk) that produces the purest subsets (weighted by their size).

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

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

Decision Tree:Training Algorithm

- It stops recursing once it reaches the maximum depth (defined by the max_depth hyperparameter), or if it cannot find a split that will reduce impurity.
- As you can see, the CART algorithm is a greedy algorithm: it greedily searches for an optimum split at the top level
- It does not check whether or not the split will lead to the lowest possible impurity several levels down.

Decision Tree:Cost Fuction

- Gini Impurity or Entropy?
  - By default, the Gini impurity measure is used, but you can select the entropy impurity measure instead by setting the criterion hyperparameter to "entropy".
  - a set’s entropy is zero when it contains instances of only one class.

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

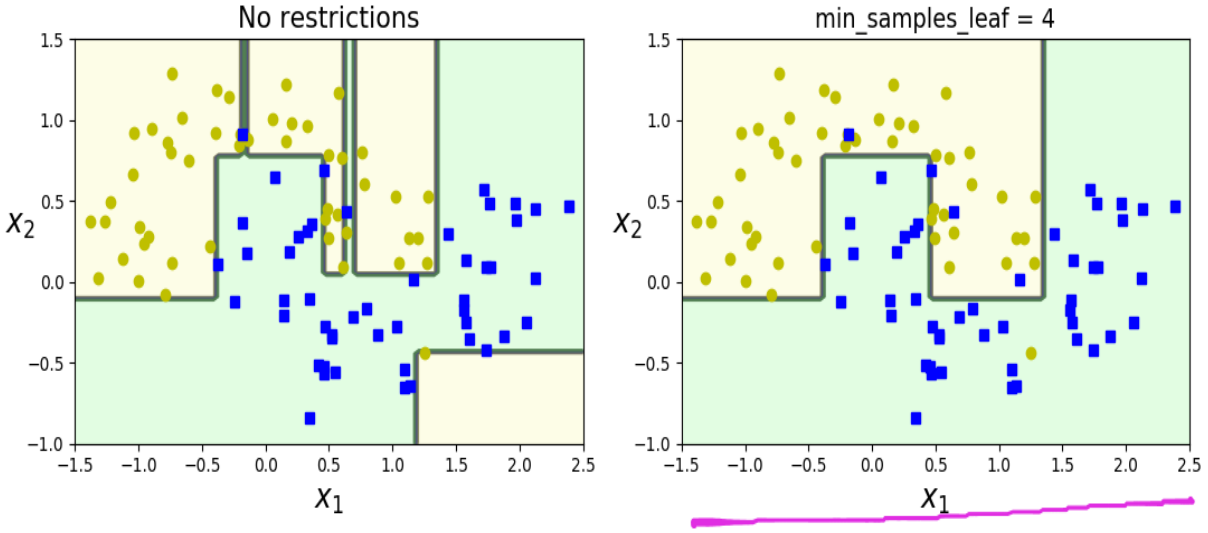
$$-\frac{49}{54}\log\left(\frac{49}{54}\right) - \frac{5}{54}\log\left(\frac{5}{54}\right) \approx 0.3\overline{1}.$$

Decision Tree:Cost Fuction

- Gini Impurity or Entropy?
  - The truth is, most of the time it does not make a big difference: they lead to similar trees. Gini impurity is slightly faster to compute, so it is a good default.
  - However, when they differ, Gini impurity tends to isolate the most frequent class in its own branch of the tree, while entropy tends to produce slightly more balanced trees.

Decision Tree:Overfitting and Regularization

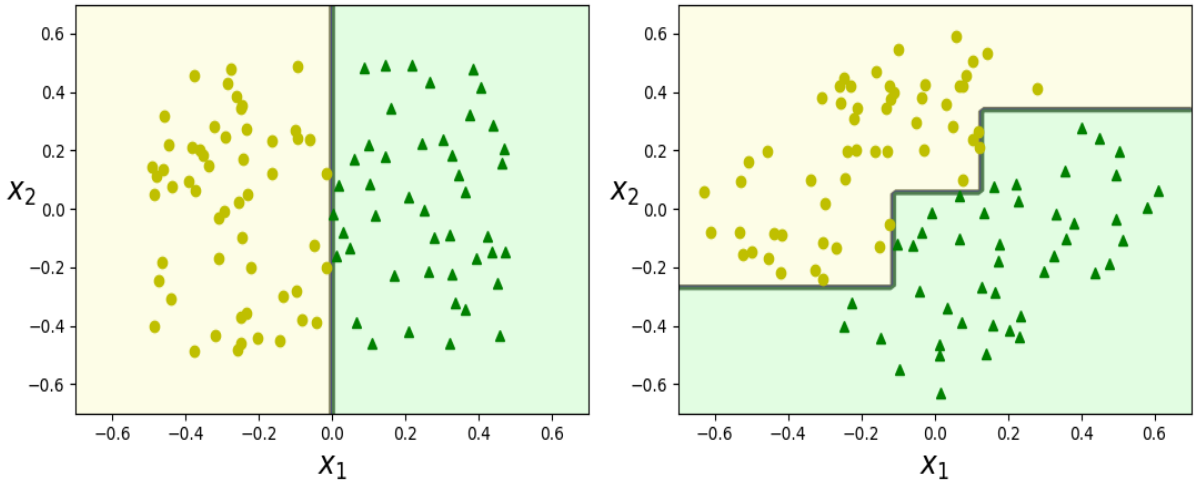
```
Xm, ym = make_moons(n_samples=100, noise=0.25, random_state=53)

deep_tree_clf1 = DecisionTreeClassifier(random_state=42)
deep_tree_clf2 = DecisionTreeClassifier(min_samples_leaf=4, random_state=42)
deep_tree_clf1.fit(Xm, ym)
deep_tree_clf2.fit(Xm, ym)

plt.figure(figsize=(11, 4))
plt.subplot(121)
plot_decision_boundary(deep_tree_clf1, Xm, ym, axes=[-1.5, 2.5, -1, 1.5], iris=False)
plt.title("No restrictions", fontsize=16)
plt.subplot(122)
plot_decision_boundary(deep_tree_clf2, Xm, ym, axes=[-1.5, 2.5, -1, 1.5], iris=False)
plt.title("min_samples_leaf = {}".format(deep_tree_clf2.min_samples_leaf), fontsize=14)

fl.save_fig("min_samples_leaf_plot")
plt.show()
```



- Minimum num of of samples a leaf must have.
- min_weight_fraction_leaf: same as above exept expressed as a fraction.
- max_features
- Increasing min_* and decreasing max_* will regularize the model.

Decision Tree:Overfitting and Regularization

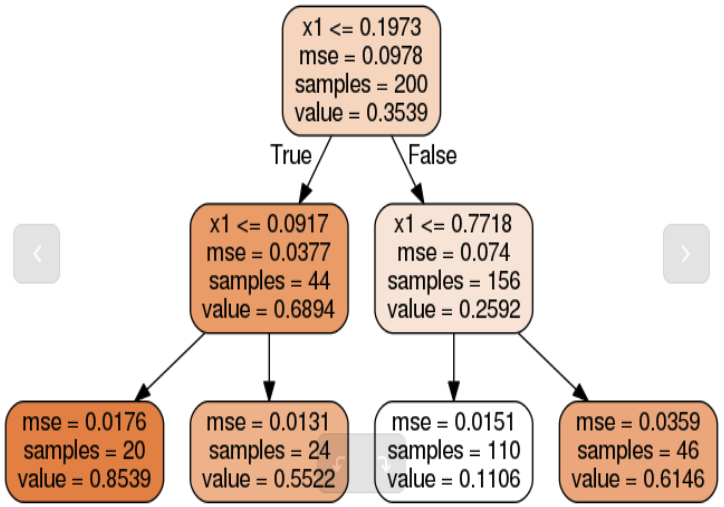
```
rnd.seed(6)
Xs = rnd.rand(100, 2) - 0.5
ys = (Xs[:, 0] > 0).astype(np.float32) * 2

angle = np.pi / 4
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
Xsr = Xs.dot(rotation_matrix)

tree_clf_s = DecisionTreeClassifier(random_state=42)
tree_clf_s.fit(Xs, ys)
tree_clf_sr = DecisionTreeClassifier(random_state=42)
tree_clf_sr.fit(Xsr, ys)

plt.figure(figsize=(11, 4))
plt.subplot(121)
plot_decision_boundary(tree_clf_s, Xs, ys, axes=[-0.7, 0.7, -0.7, 0.7], iris=False)
plt.subplot(122)
plot_decision_boundary(tree_clf_sr, Xsr, ys, axes=[-0.7, 0.7, -0.7, 0.7], iris=False)

fl.save_fig("sensitivity_to_rotation_plot")
plt.show()
```



- They like orthogonal decision boundaries.
- If the data is rotated by 45° it takes unnecessary convolutions
- Although both fit the data well. It is likely that the model on the right will not generalize well.
- One way to limit the problem is to use PCA.

Decision Tree:Regression

```python
from sklearn.tree import DecisionTreeRegressor

# Quadratic training set + noise
rnd.seed(42)
m = 200
X = rnd.rand(m, 1)
y = 4 * (X - 0.5) ** 2
y = y + rnd.randn(m, 1) / 10

tree_reg1 = DecisionTreeRegressor(random_state=42, max_depth=2)
tree_reg2 = DecisionTreeRegressor(random_state=42, max_depth=3)
tree_reg1.fit(X, y)
tree_reg2.fit(X, y)
```



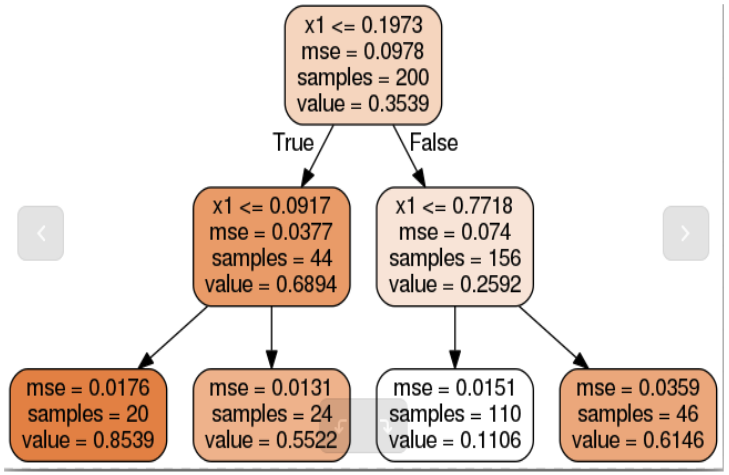
- If $x_1 = 0.6$, DTR will traverse the tree, starting at root and predict a value of 0.1106.
- The value is average target value of 110 instances associated with leaf node.
- This prediction result in a MSE of 0.0151 over 110 instances.

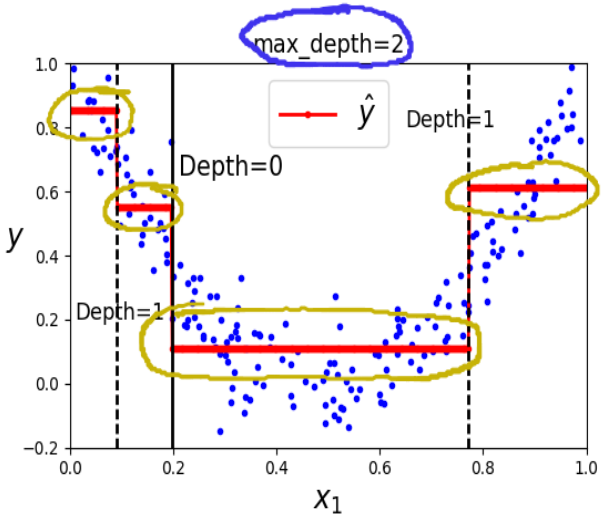
Decision Tree:Regression

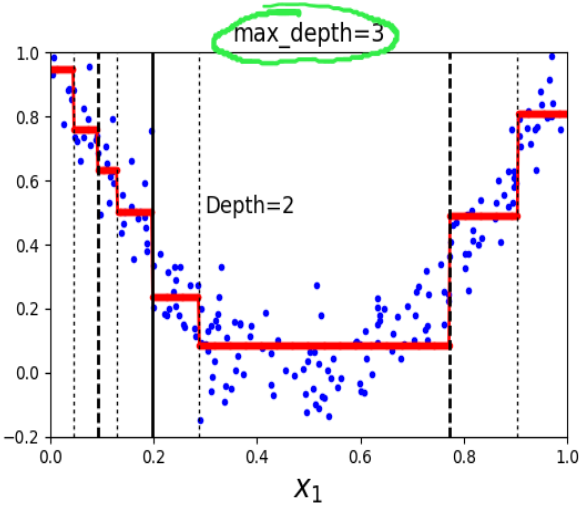
```
from sklearn.tree import DecisionTreeRegressor

# Quadratic training set + noise
rnd.seed(42)
m = 200
X = rnd.rand(m, 1)
y = 4 * (X - 0.5) ** 2
y = y + rnd.randn(m, 1) / 10

tree_reg1 = DecisionTreeRegressor(random_state=42, max_depth=2)
tree_reg2 = DecisionTreeRegressor(random_state=42, max_depth=3)
tree_reg1.fit(X, y)
tree_reg2.fit(X, y)
```

- Average value of instances in that leaf.
- Cost function for DTR (minimize MSE)

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



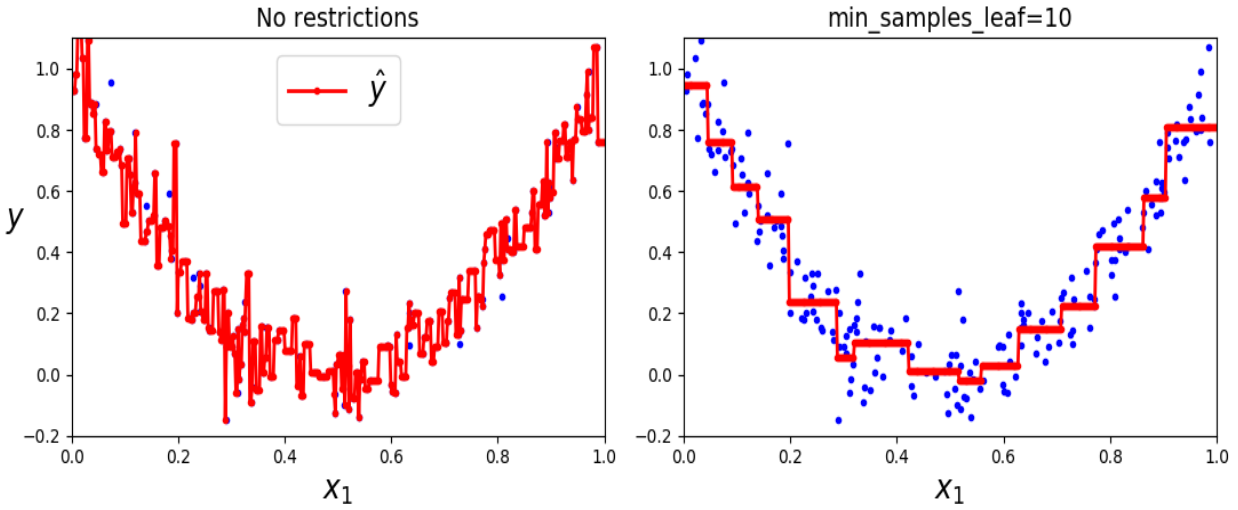




Decision Tree:Regression

```
from sklearn.tree import DecisionTreeRegressor
rnd.seed(42)
m = 200
X = rnd.rand(m, 1)
y = 4 * (X - 0.5) ** 2
y = y + rnd.randn(m, 1) / 10

tree_reg1 = DecisionTreeRegressor(random_state=42)
tree_reg2 = DecisionTreeRegressor(random_state=42, min_samples_leaf=10)
tree_reg1.fit(X, y)
tree_reg2.fit(X, y)
```

• Effect of regularization.



Decision Tree:Sensitivity

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

X[(X[:, 1]==X[:, 1][y==1].max()) & (y==1)] # widest Iris-Versicolor flower
print("X:",X)
not_widest_versicolor = (X[:, 1]!=1.8) | (y==2)

X_tweaked = X[not_widest_versicolor]
y_tweaked = y[not_widest_versicolor]

tree_clf_tweaked = DecisionTreeClassifier(max_depth=2, random_state=40)
tree_clf_tweaked.fit(X_tweaked, y_tweaked)

plt.figure(figsize=(8, 4))
plot_decision_boundary(tree_clf_tweaked, X_tweaked, y_tweaked, legend=False)
plt.plot([0, 7.5], [0.8, 0.8], "k-", linewidth=2)
plt.plot([0, 7.5], [1.75, 1.75], "k--", linewidth=2)
plt.text(1.0, 0.9, "Depth=0", fontsize=15)
plt.text(1.0, 1.80, "Depth=1", fontsize=13)
```