SAIF

Shanghai Advanced Institute of Finance
上海高级金融学院

量化俱乐部-机器学习-DecisionTrees

2019-08-31

Menu

1. Training and Visualizing a Decision Tree
2. Making Predictions
3. Estimating Class Probabilities
4. The CART Training Algorithm
5. Computational Complexity
6. Regularization Hyperparameters
7. Regression

自我介绍

张骁喆，2010年大连理工大学软件学院本科毕业，高金FMBA 2017PTE。9年计算机工作经验，熟悉开发，测试，运维，项目管理，产品设计各环节。
曾就职eBay，唯品会，2016加入SAP，现担任SAP Cloud部门开发团队主管。
2017开始接触量化投资和python，目前毕业论文研究方向使用机器学习在A股进行量化投资。
量化投资/机器学习咨询培训/项目管理产品管理。

Intro

Like SVMs, Decision Trees are versatile Machine Learning algorithms that can perform both classification and regression tasks, and even multioutput tasks. They are very powerful algorithms, capable of fitting complex datasets.

In this chapter we will start by discussing how to train, visualize, and make predictions with Decision Trees. Then we will go through the CART training algorithm used by Scikit-Learn, and we will discuss how to regularize trees and use them for regression tasks. Finally, we will discuss some of the limitations of Decision Trees.

Training and Visualizing a Decision Tree

To understand Decision Trees, let's just build one and take a look at how it makes predictions. The following code trains a DecisionTreeClassifier on the iris dataset

In [7]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width  X.shape: (150,2)
y = iris.target

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

Out[7]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=42, splitter='best')
```

Training and Visualizing a Decision Tree

You can visualize the trained Decision Tree by first using the export_graphviz() method to output a graph definition file called iris_tree.dot

```
In [9]: 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
            )
```

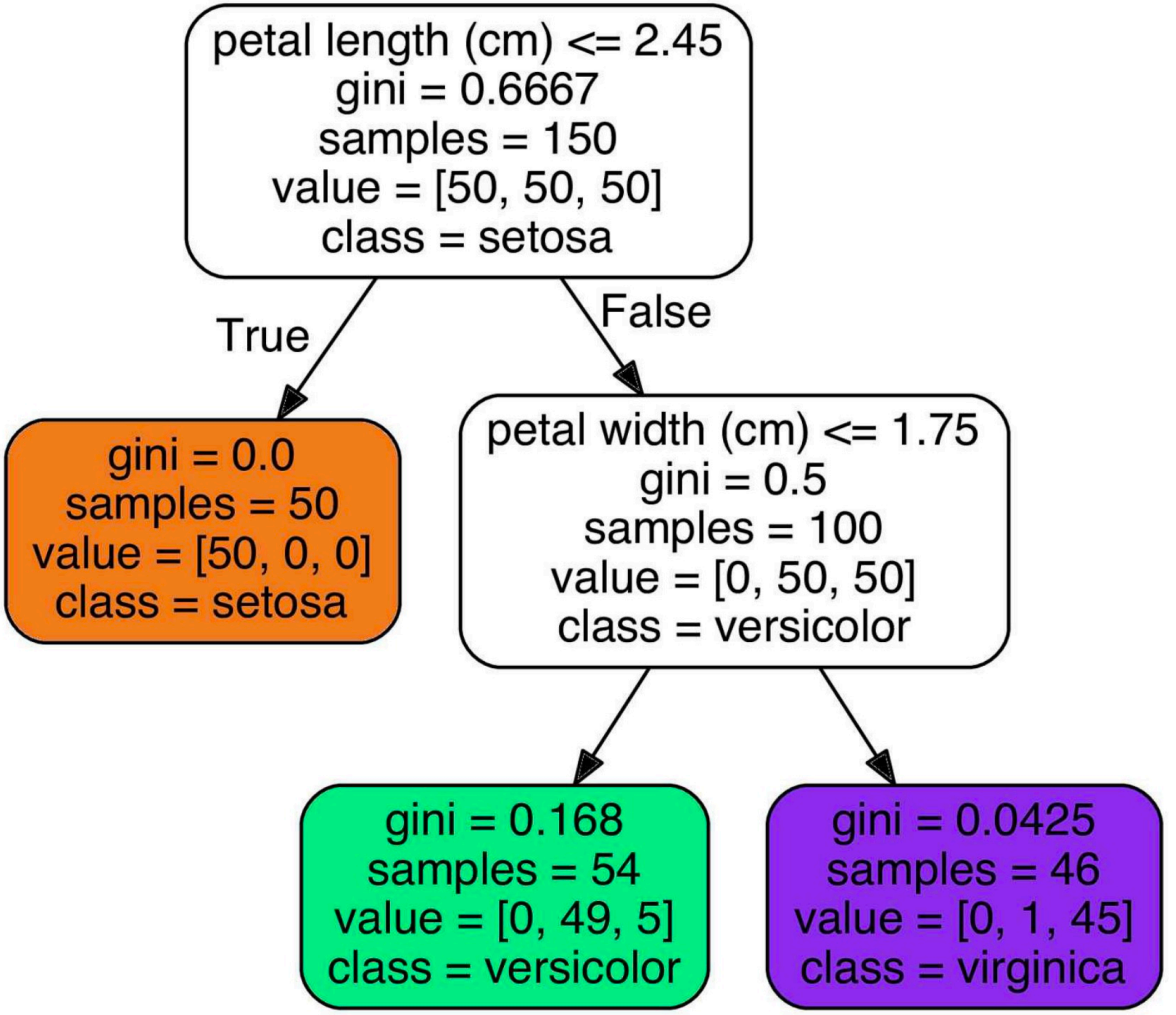
Training and Visualizing a Decision Tree

Then you can convert this .dot file to a variety of formats such as PDF or PNG using the dot command-line tool from the graphviz package.
This command line converts the .dot file to a .png imag

```
$ dot -Tpng iris_tree.dot -o iris_tree.png
```

Training and Visualizing a Decision Tree

Your first decision tree looks like:



Making Predictions

A node’ s gini attribute measures its impurity: a node is “pure” (gini=0) if all training instances it applies to belong to the same class

For example, since the depth-1 left node applies only to Iris-Setosa training instances, it is pure and its gini score is 0. Equation 6-1 shows how the training algorithm computes the gini score Gi of the ith node. For example, the depth-2 left node has a gini score equal to
1 – (0/54)^2 – (49/54)^2 – (5/54)^2 ≈ 0.168

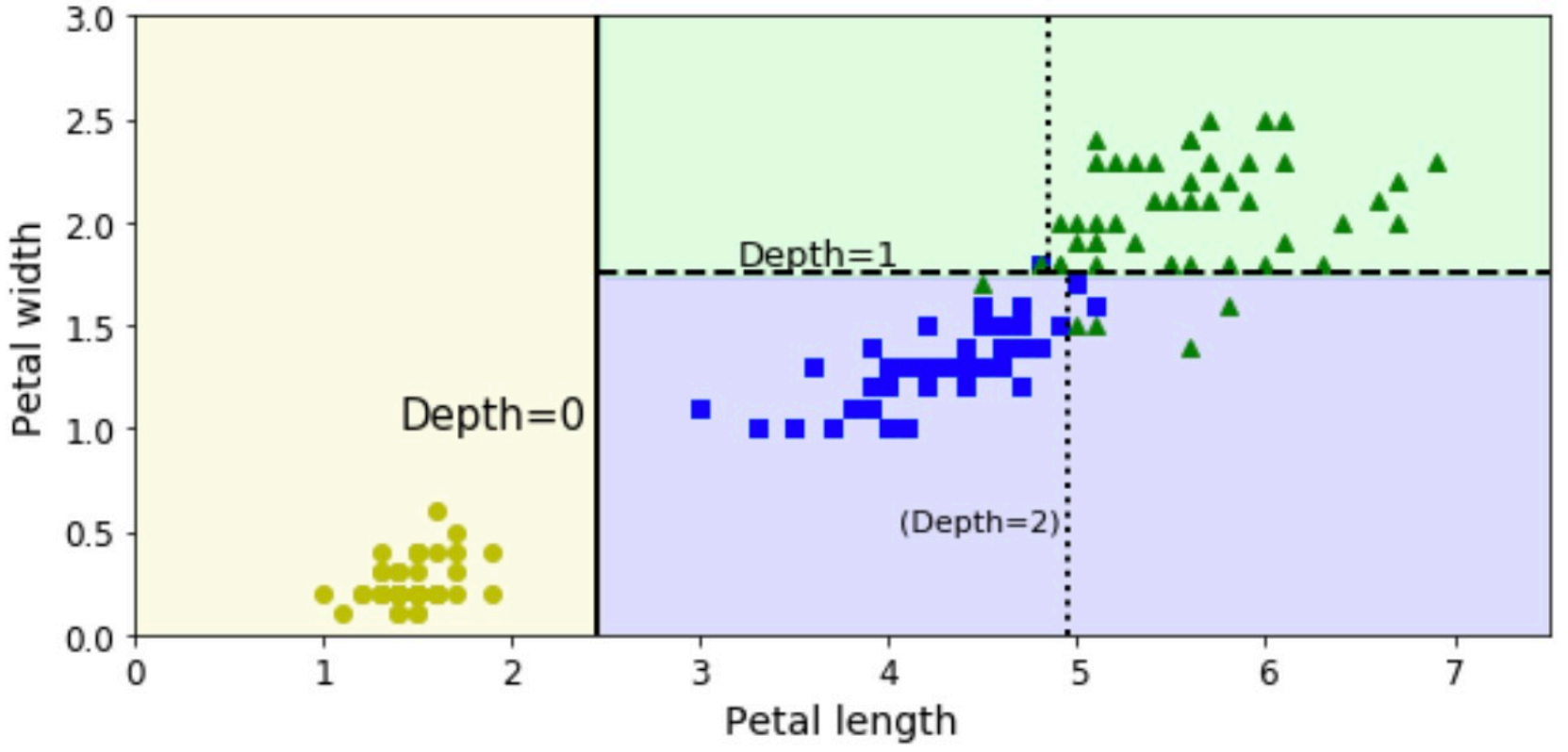
Equation 6-1. Gini impurity

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

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

Making Predictions

shows this Decision Tree's decision boundaries



Estimating Class Probabilities

A Decision Tree can also estimate the probability that an instance belongs to a particular class k

```
# Predicting classes and class probabilities

In [11]: tree_clf.predict_proba([[5, 1.5]])
Out[11]: array([[0.        , 0.90740741, 0.09259259]])

In [12]: tree_clf.predict([[5, 1.5]])
Out[12]: array([1])

In [13]: tree_clf.predict_proba([[7, 1.5]])
Out[13]: array([[0.        , 0.90740741, 0.09259259]])
```

The CART Training Algorithm

Scikit-Learn uses the 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).
The cost function that the algorithm tries to minimize is given :

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

The CART Training Algorithm

Once it has successfully split the training set in two, it splits the subsets using the same logic, then the sub-subsets and so on, recursively. 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.
A few other hyperparameters (described in a moment) control additional stopping conditions (***min_samples_split***, ***min_samples_leaf***, ***min_weight_fraction_leaf***, and ***max_leaf_nodes***).

Unfortunately, finding the optimal tree is known to be an NP-Complete problem: It requires O(exp(m)) time, making the problem intractable even for fairly small training sets. This is why we must settle for a "reasonably good" solution.

Computational Complexity

Making predictions requires traversing the Decision Tree from the root to a leaf. Decision Trees are generally approximately balanced, so traversing the Decision Tree requires going through roughly O(log2(m)) nodes. Since each node only requires checking the value of one feature, the overall prediction complexity is just O(log2(m)), independent of the number of features. So predictions are very fast, even when dealing with large training sets.

However, the training algorithm compares all features (or less if max_features is set) on all samples at each node. This results in a training complexity of O(n × m log(m)). For small training sets (less than a few thousand instances), Scikit-Learn can speed up training by presorting the data (set presort=True), but this slows down training considerably for larger training sets.

Regularization Hyperparameters

DT is often called a nonparametric model, not because it does not have any parameters (it often has a lot) but because the number of parameters is not determined prior to training

In contrast, a parametric model such as a linear model has a predetermined number of parameters, so its degree of freedom is limited, reducing the risk of overfitting (but increasing the risk of underfitting)

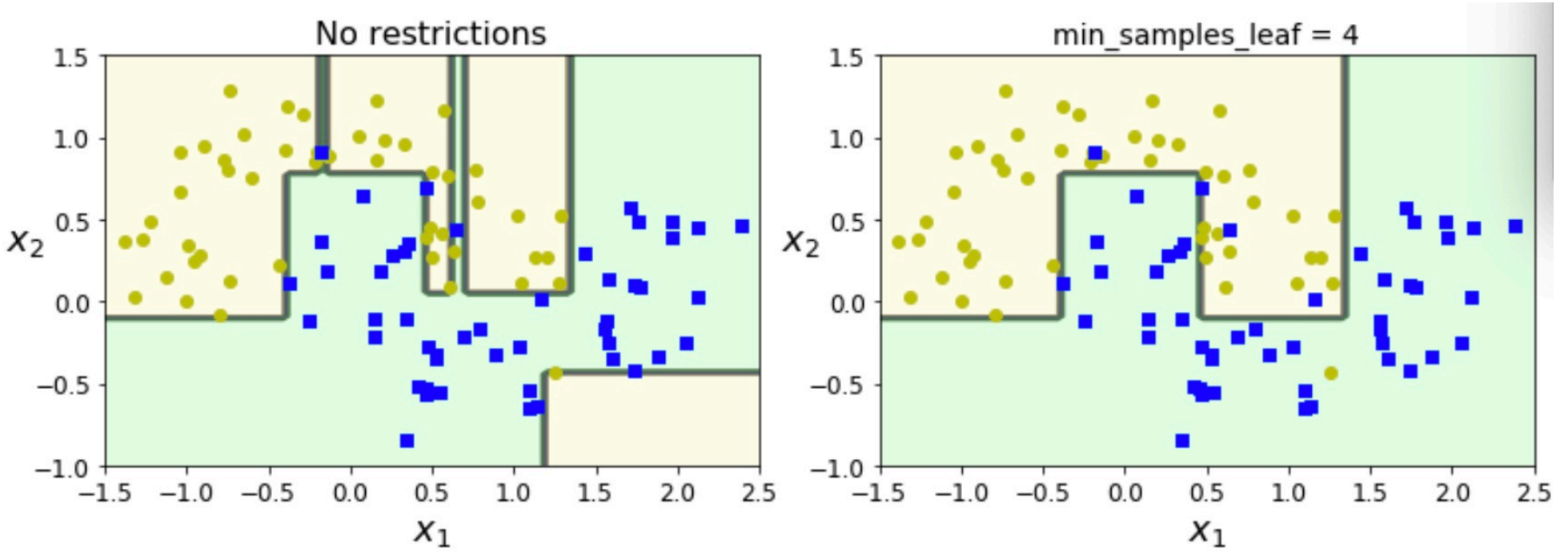
To avoid overfitting the training data, you need to restrict the Decision Tree's freedom during training. As you know by now, this is called regularization. The regularization hyperparameters depend on the algorithm used, but generally you can at least restrict the maximum depth of the Decision Tree. In Scikit-Learn, this is controlled by the ***max_depth*** hyperparameter

The DecisionTreeClassifier class has a few other parameters that similarly restrict the shape of the Decision Tree: ***min_samples_split***, ***min_samples_leaf***, , ***max_leaf_nodes***, and ***max_features***.

Increasing min_* hyperparameters or reducing max_* hyperparameters will regularize the model

Regularization Hyperparameters

Below shows two Decision Trees trained on the moons dataset. On the left, the Decision Tree is trained with the default hyperparameters (i.e., no restrictions), and on the right the Decision Tree is trained with min_samples_leaf=4.
It is quite obvious that the model on the left is overfitting, and the model on the right will probably generalize better.



Regression

Decision Trees are also capable of performing regression tasks. Let's build a regression tree using Scikit-Learn's DecisionTreeRegressor class, training it on a noisy quadratic dataset with max_depth=2

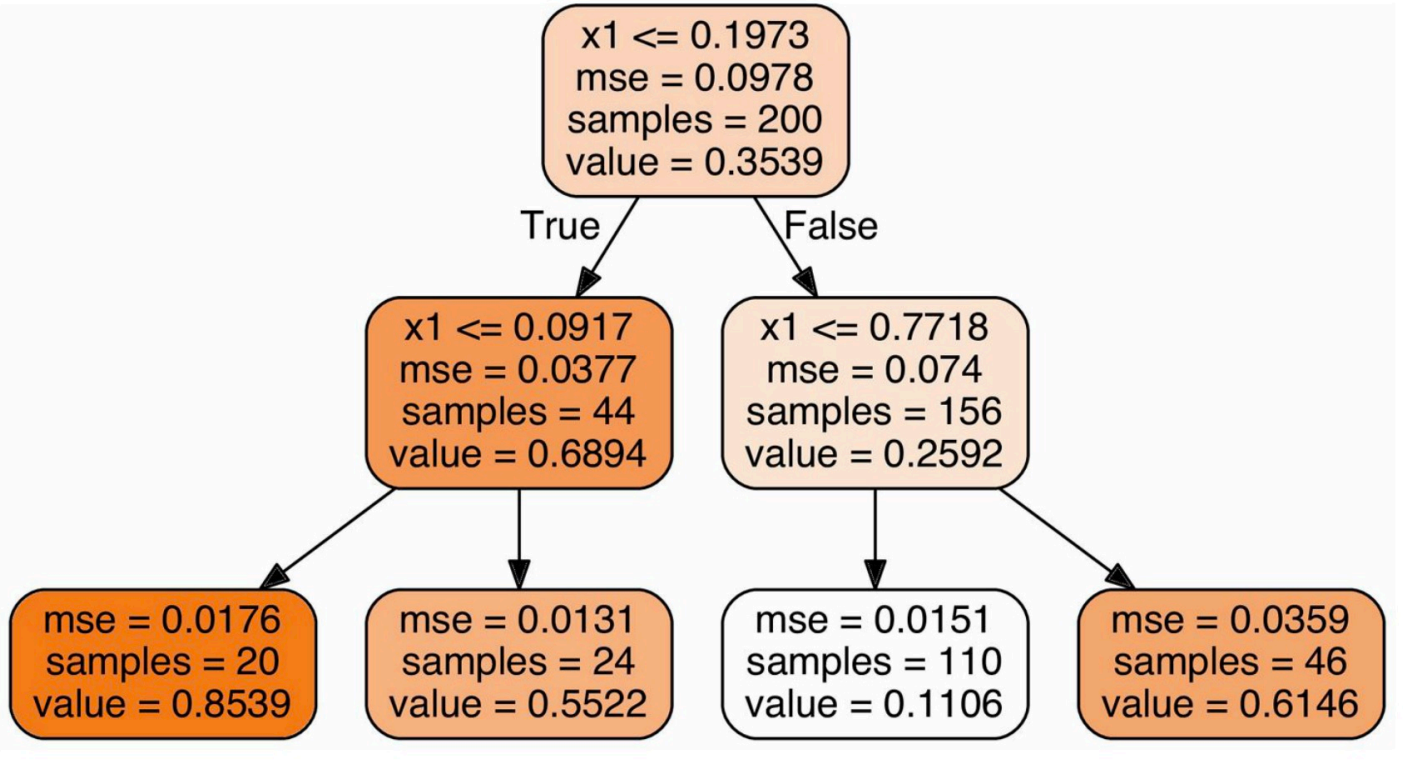
In [9]:
```
# Quadratic training set + noise
np.random.seed(42)
m = 200
X = np.random.rand(m, 1)
y = 4 * (X - 0.5) ** 2
y = y + np.random.randn(m, 1) / 10
```

In [10]:
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X, y)
```

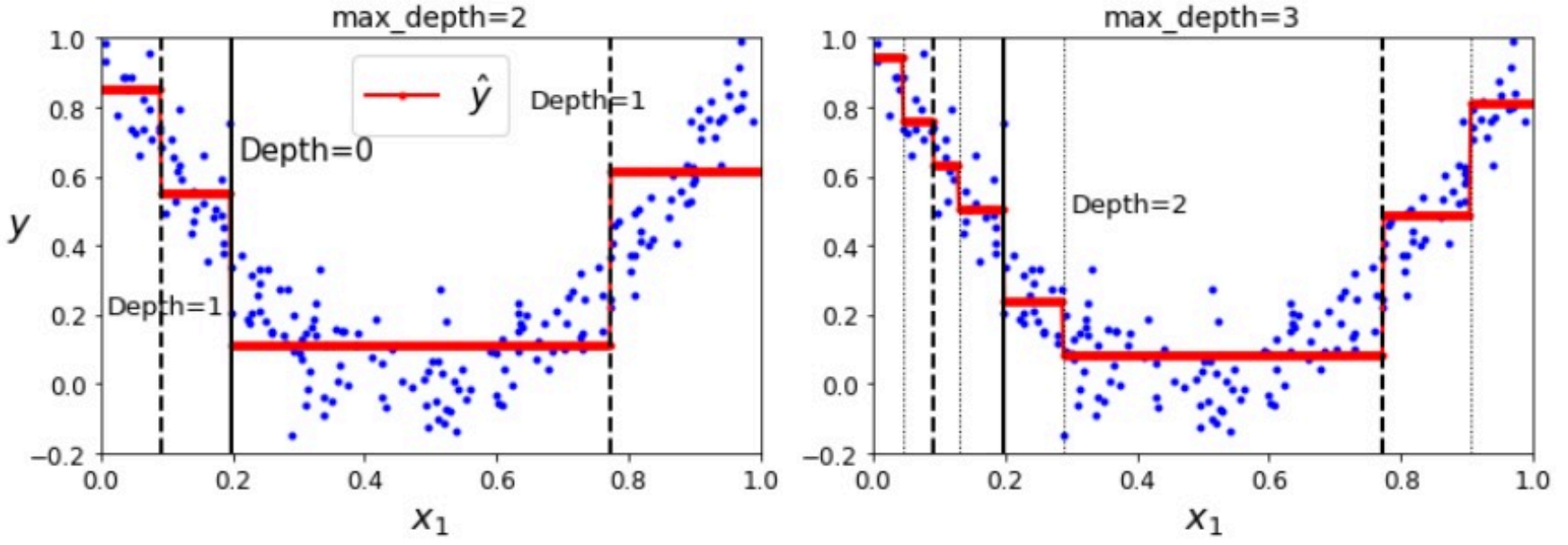
Regression

suppose you want to make a prediction for a new instance with x1 = 0.6. You traverse the tree starting at the root, and youtually reach the leaf node that predicts value=0.1106



Regression

This model’ s predictions are represented on the left. If you set max_depth=3, you get the predictions represented on the right. Notice how the predicted value for each region is always the average target value of the instances in that region. The algorithm splits each region in a way that makes most training instances as close as possible to that predicted value



Regression

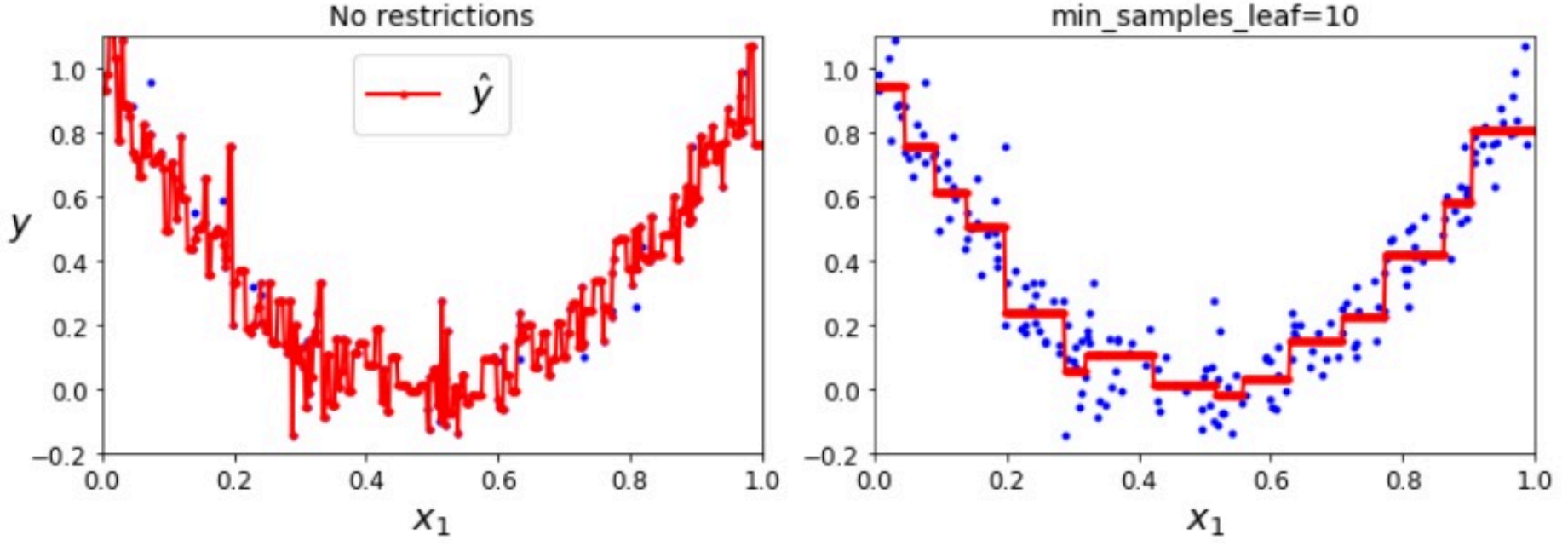
The CART algorithm works mostly the same way as earlier, except that instead of trying to split the training set in a way that minimizes impurity, it now tries to split the training set in a way that minimizes the MSE. Equation shows the cost function that the algorithm tries to minimize

Equation 6-4. CART cost function for regression

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

Regression

Just like for classification tasks, Decision Trees are prone to overfitting when dealing with regression tasks. Without any regularization (i.e., using the default hyperparameters), you get the predictions on the left of Figure. It is obviously overfitting the training set very badly. Just setting min_samples_leaf=10 results in a much more reasonable model



SAIF

Shanghai Advanced
Institute of Finance
上海高级金融学院