INFN Machine Learning course

Prof. Amir Farbin, Prof. Daniele Bonacorsi

20-22 May 2019

Camogli, Italy

Decision Trees

Decision Trees (**DT**)

Decision Trees are quite versatile ML algos

- they can perform both *classification* and *regression* tasks (even multi-output tasks)
- they are powerful enough to be capable of fitting complex datasets
- DTs are also the fundamental components of **Random Forests**, which are among the most powerful ML algos available today

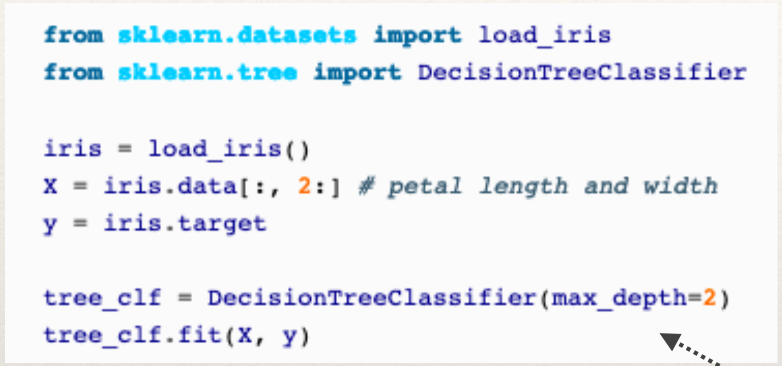
Overview

Briefly, on:

- how to train, visualize, and make predictions with Decision Trees
- CART training algorithm used by Scikit-Learn
- how to regularize trees and use them for regression tasks
- some of the limitations of Decision Trees

Example of a DT for **classification**

Sklearn on the iris flowers dataset
→ `DecisionTreeClassifier`

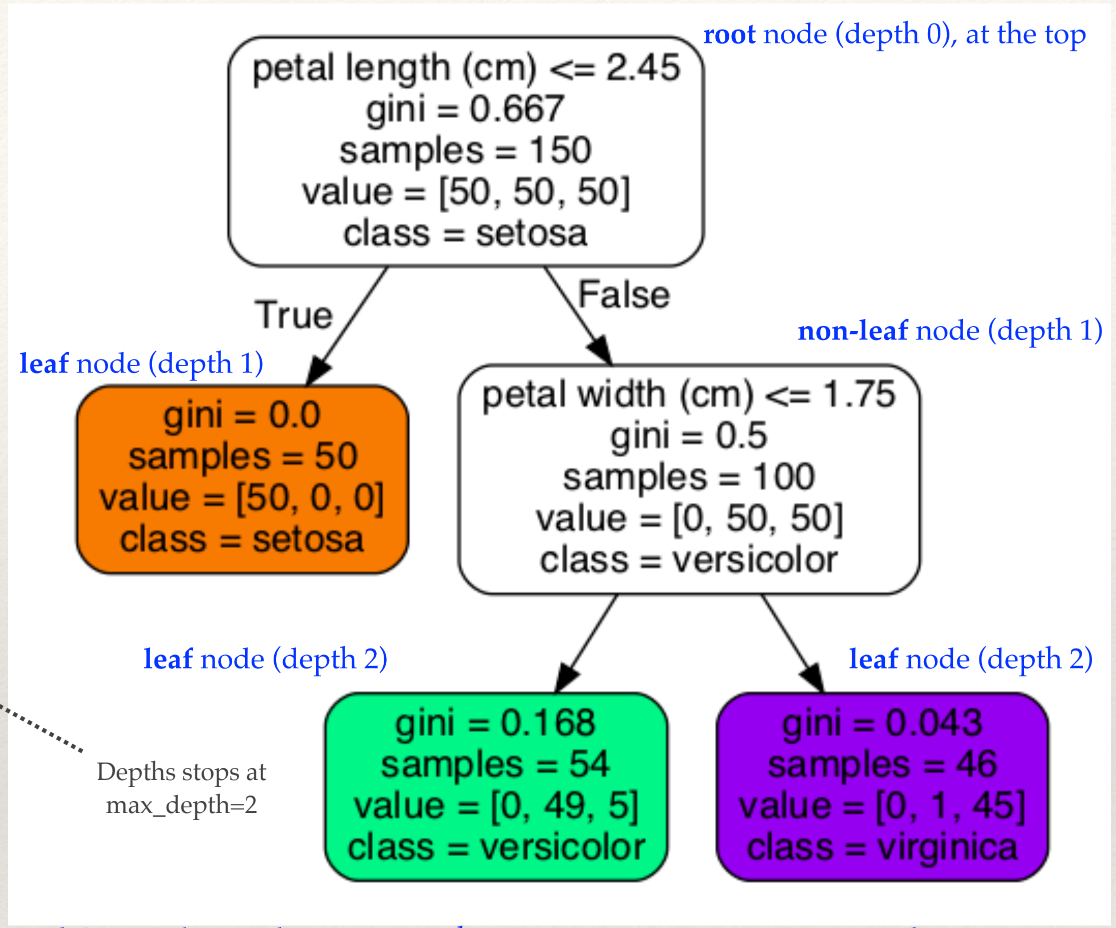


```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

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)
```

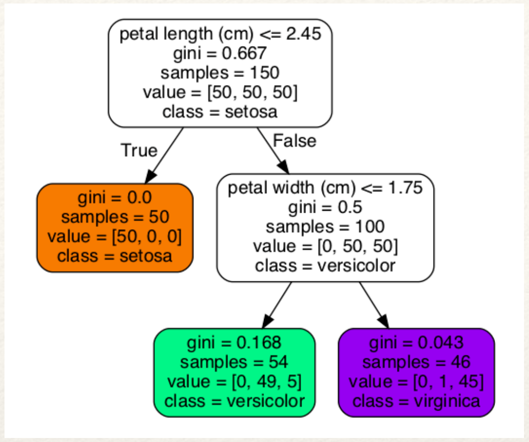
creates this:



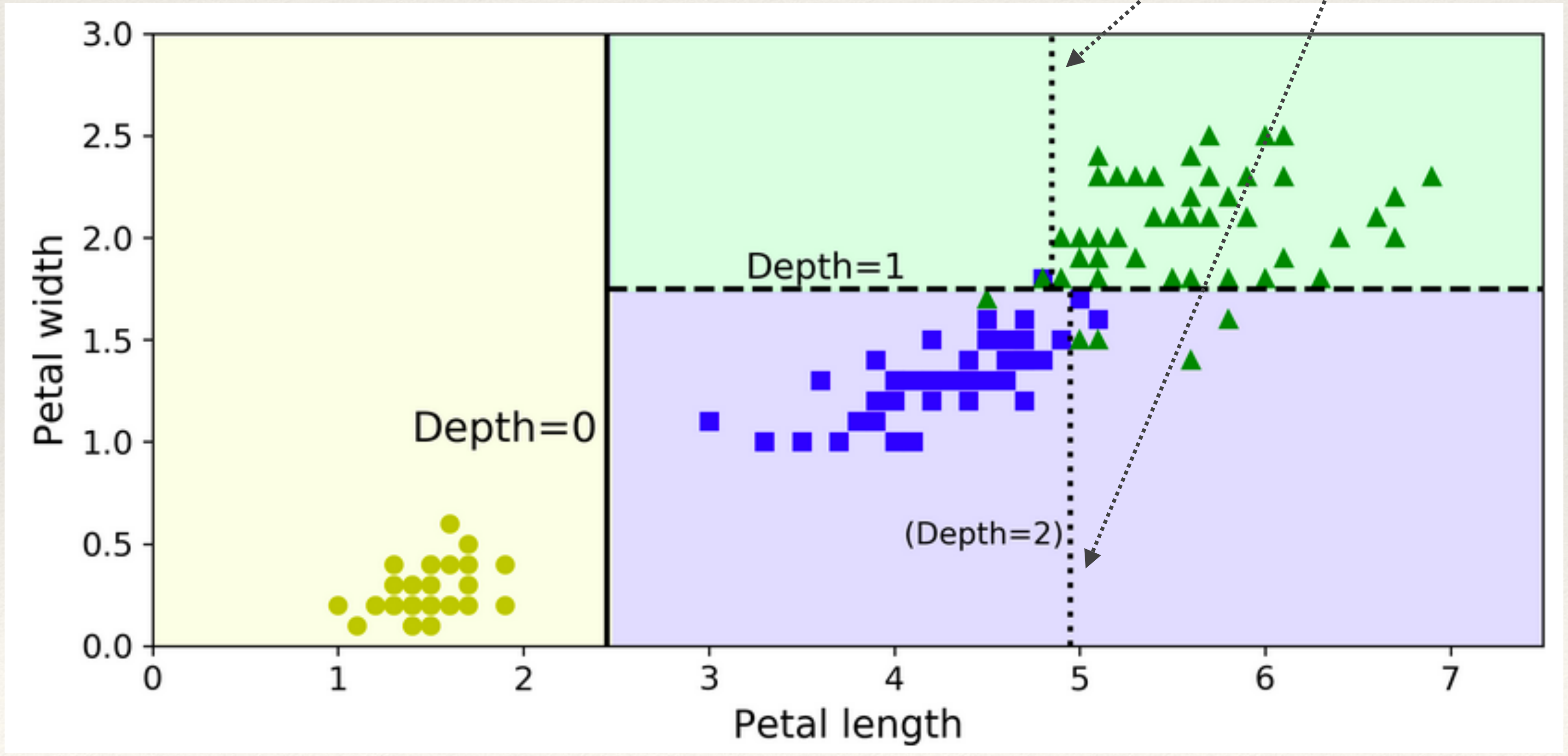
node's **samples** attribute = counts how many training instances it applies to
node's **value** attribute = how many training instances of each class this node applies to
node's **gini** attribute = measures its impurity (pure node - gini=0 - if all training instances it applies to belong to the same class.

Among the qualities of DTs is that they require very little data preparation

- in particular, they don't require feature scaling or centering at all



max_depth=3 would have added "depth 3" decision boundaries



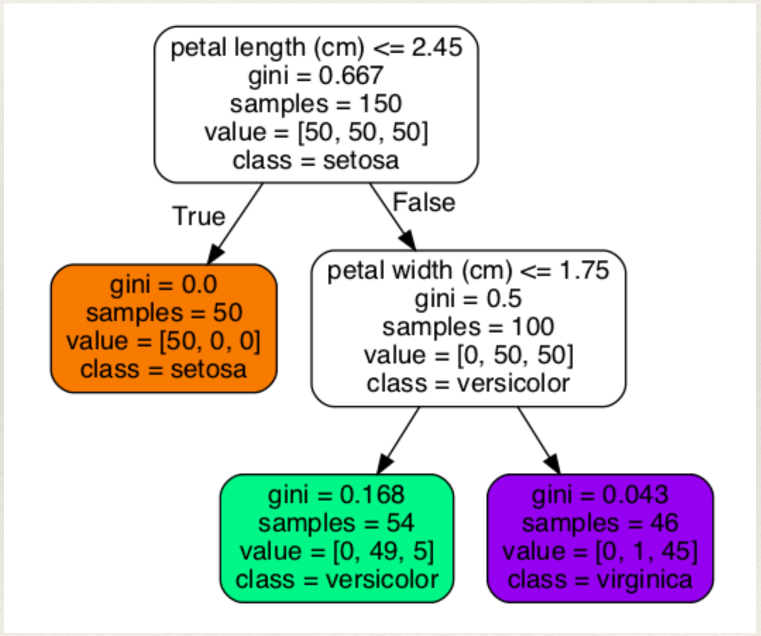
DT: estimating class probabilities

A DT can estimate the probability that an instance belongs to a particular class **k**

- first, it traverses the Tree to find the leaf node for this instance
- second, it returns the ratio of training instances of class k in this node

Example:

- a iris flower with 5cm long and 1.5cm wide petals corresponds to the green, bottom left, depth 2 node
- DT outputs the following probabilities:
  - 0/54 → 0% for Iris-Setosa
  - 49/54 → 90.7% for Iris-Versicolor
  - 5/54 → 9.3% for Iris-Virginica
- prediction outputs Iris-Versicolor (class 1)



```
>>> tree_clf.predict_proba([[5, 1.5]])
array([[0.        , 0.90740741, 0.09259259]])
```

White box vs black box

How is your intuition of the characteristics of this algo?

Decision Trees fairly intuitive and decisions <u>easy</u> to interpret

- → **"white box" models**
- you may even apply manually the same (and known) classification rules the DT itself applied

Random Forests or Neural Networks give great predictions, you can check calculations but <u>harder</u> to explain why a prediction was made

- → **"black box" models**
- e.g. a cat recognised in a picture from.. the ears? the tail? at which %?

Classification And Regression Tree (**CART**)

Sklearn uses the **CART** algo to train DTs

- it produces only binary trees (non-leaf nodes always have 2 children, i.e. questions only have yes/no answers)
  - Other algos (such as **ID3**) can produce DTs with nodes that have >2 children

The idea behind the CART algo is quite simple

- it split the training set in 2 subsets using a single feature **k** and a threshold **t_k**
- it chooses **k** and **t_k** by searching for the pair that produces the purest subsets (weighted by their size)
  - The cost function that the CART algo tries to minimize for classification is:

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

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

- then, it splits the 2 sub-sets in 2 again using the same logic, and so on, recursively, until indicated 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).

Note: CART is a greedy algo (searches for an optimum split at the top level already, and repeats, no check whether the split will eventually lead to the lowest possible impurity several levels down). OK for good solution, no guarantees for the optimal one - which is a problem that scales exponentially with m, so the problem becomes intractable for fairly small training sets. With DT and CART, do trade for a "reasonably good" solution..

Example of a DT for **regression**

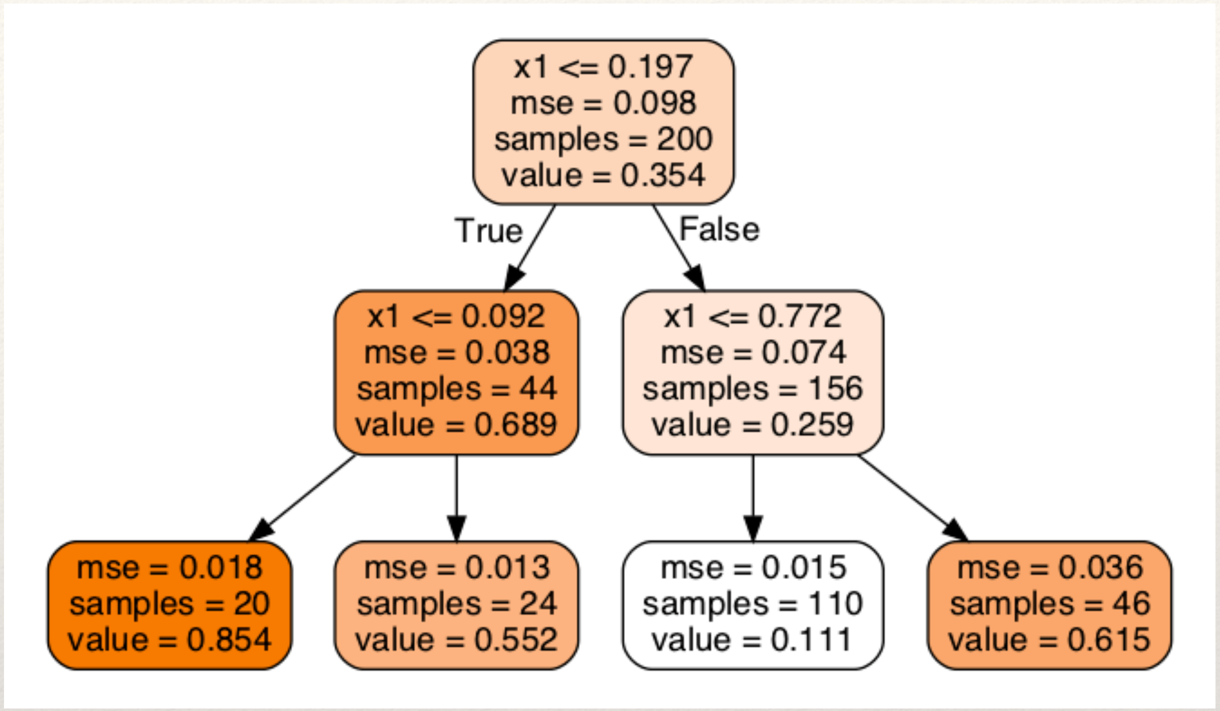
Sklearn on a noise quadratic dataset

→ `DecisionTreeRegressor`

```python
from sklearn.tree import DecisionTreeRegressor

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

creates this:



It does not predict a **class**, but a **value**

- example: I want a value prediction for a new instance with x_1=0.6, I traverse tree starting and reach the leaf node that predicts value=0.111 - which is the average target value of the 110/200 training instances associated to this leaf node, and this prediction results in a Mean Squared Error (MSE) equal to 0.015 over these 110 instances.

Example of a DT for **regression**

Sklearn on a noise quadratic dataset

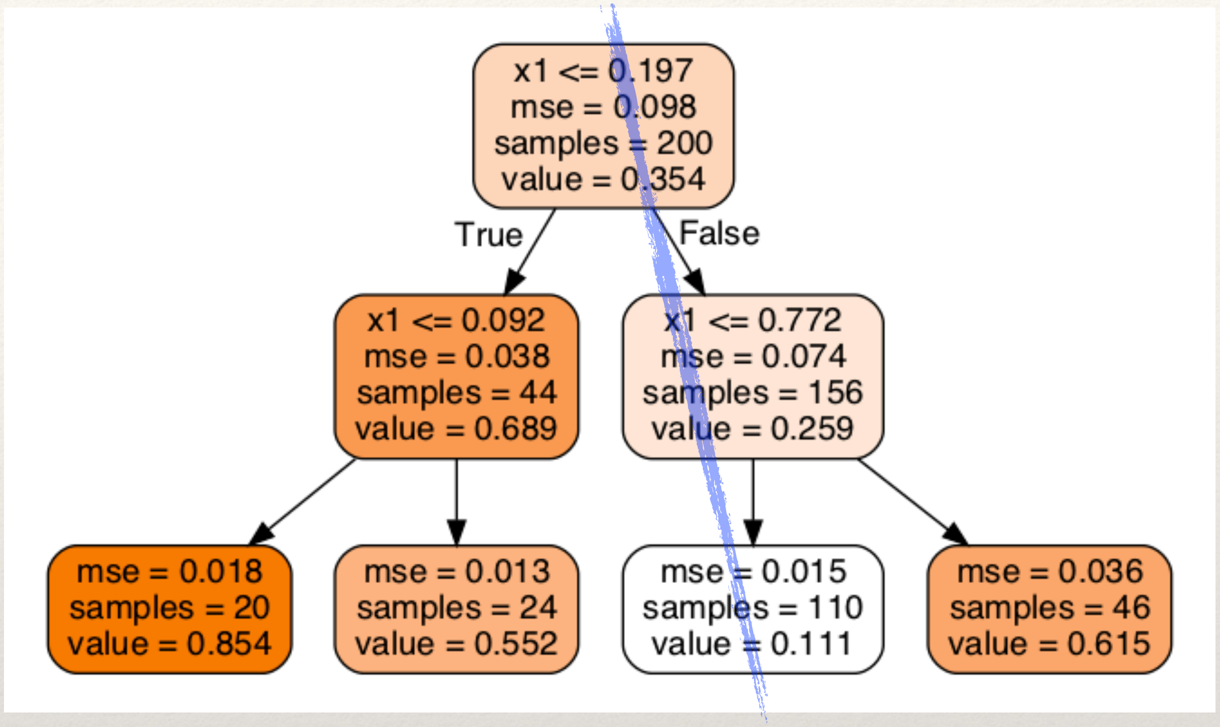
→ `DecisionTreeRegressor`

```
from sklearn.tree import DecisionTreeRegressor

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

creates this:

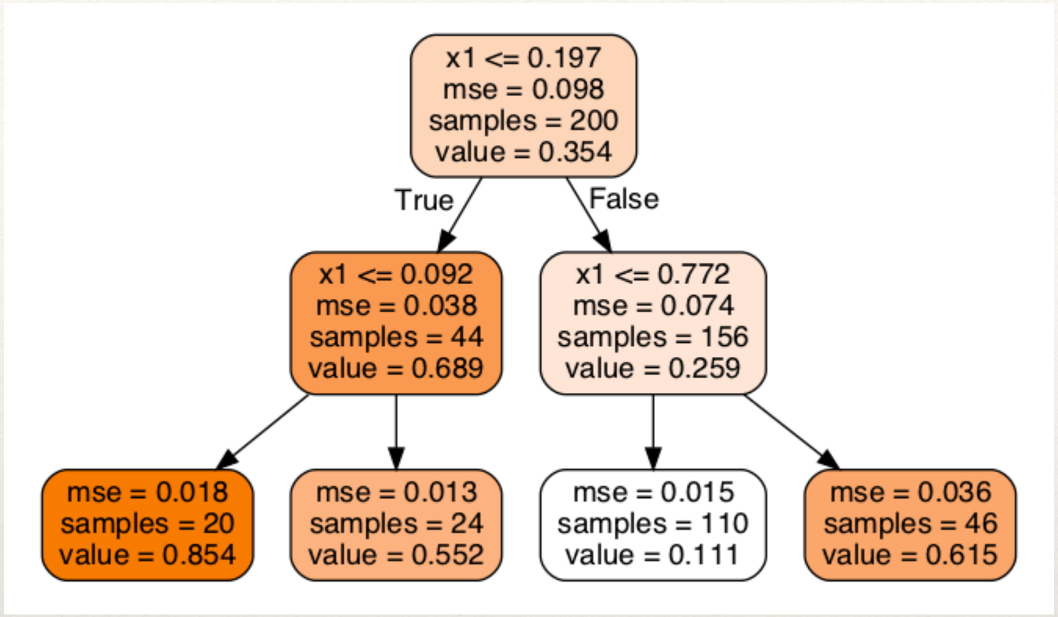
e.g. x_1=0.6

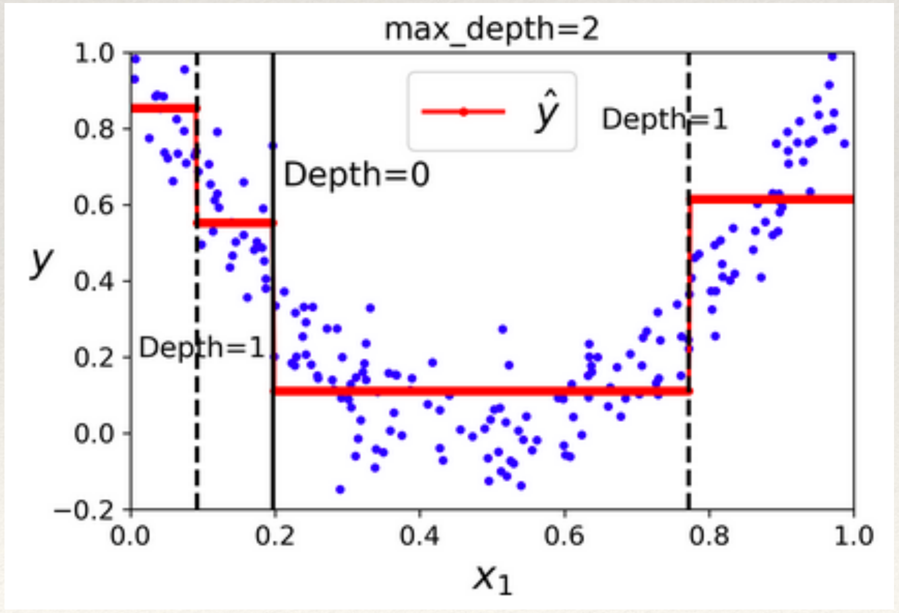


It does not predict a **class**, but a **value**

- example: I want a value prediction for a new instance with x_1=0.6, I traverse tree starting and reach the leaf node that predicts value=0.111 - which is the average target value of the 110/200 training instances associated to this leaf node, and this prediction results in a Mean Squared Error (MSE) equal to 0.015 over these 110 instances.

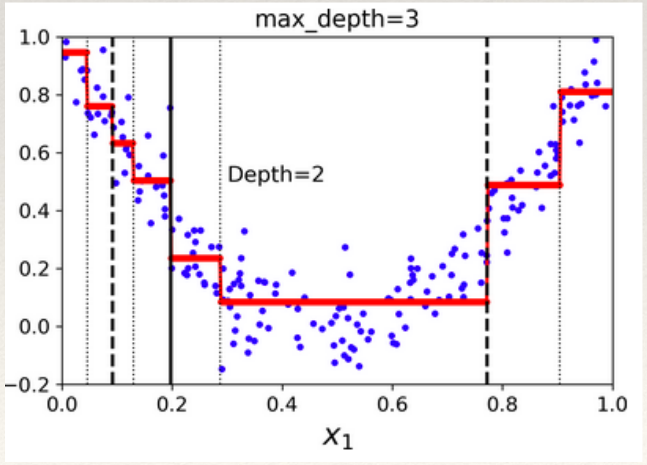
Example of a DT for **regression**





Note: the **predicted value** for each region is always the average target value of the instances in that region.

If you change to max_depth=3:



Example of a DT for **regression**

The CART algo for regression (classification) tries to split the training set in a way that minimises impurity (MSE)

- The cost function that the CART algo tries to minimize for classification was:

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

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

- … for regression is:

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

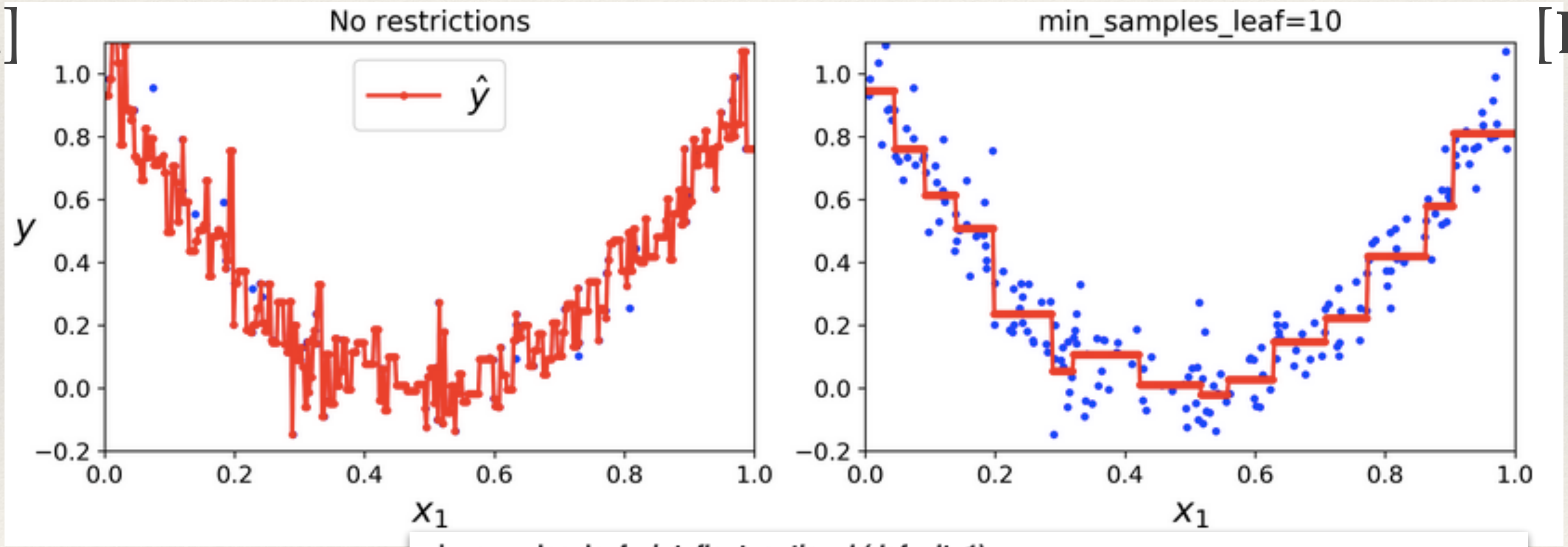
DT prone to overfitting

Note that - just like for classification tasks - DTs are prone to **overfitting** also when dealing with regression tasks.

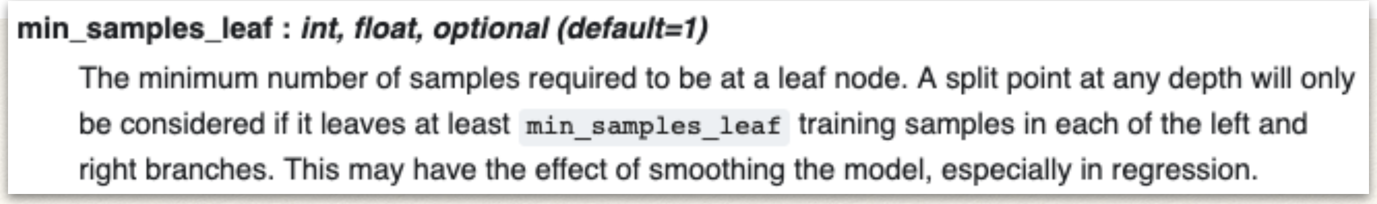
- Using the default hyper-parameters, i.e. without any regularization, you get the predictions of type [A] below. Just setting min_samples_leaf=10 in sklearn results in a much more reasonable model as in [B] below

[A]

[B]



from sklearn documentation:



min_samples_leaf : *int, float, optional (default=1)*

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

DT instabilities and limitations

In summary, DTs have a lot of pros:

- simple to understand and **interpret**
- **easy** to use, **versatile**, **powerful**

They also have cons:

- In general, main issue with DTs is that **they are very sensitive to small variations in the training data**
- <u>Example</u>: DTs are inclined towards orthogonal decision boundaries (all splits are perpendicular to an axis): this makes them sensitive to training set rotation
    - e.g. rotate a data points distribution by 45° and an easy split [A] becomes an unnecessarily convoluted decision boundary [B]. Both DTs fit the training set perfectly, but one should not be surprised that the model as in [B] may not generalize well.
    - One way to limit this problem is to use PCA, which often results in a better orientation of the training data..

Actually, since the training algo used by sklearn is stochastic, one may get different models even on the same training data (unless one sets the random_state hyper-parameter)

Random Forests can limit this instability by averaging predictions over many trees.