Decision Trees

Michela Papandrea

University of Applied Sciences and Arts of Southern Switzerland

michela.papandrea@supsi.ch

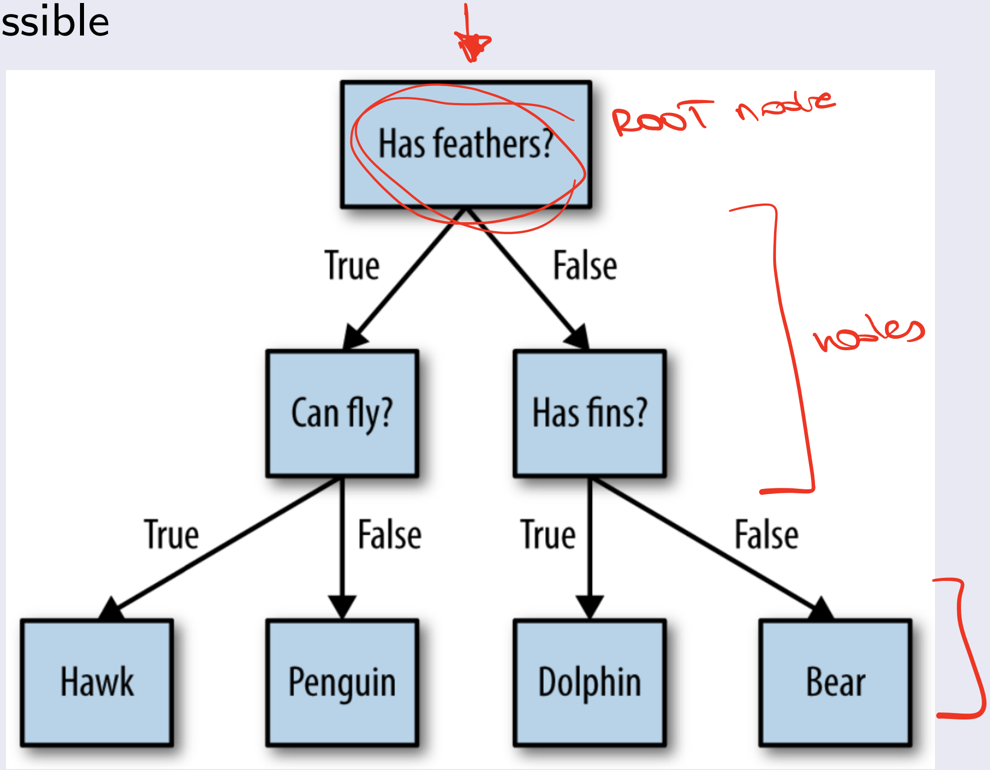
Overview

1. Visualize and make predictions with Decision Trees
2. CART training algorithm
3. Decision Trees for regression tasks
4. Decision Trees limitations

Decision Trees

Decision Trees

- widely used models for classification and regression tasks
- they learn a *hierarchy of if/else questions*, leading to a decision
- **Example**: Imagine you want to distinguish between the following four animals: bears, hawks, penguins, and dolphins. Your goal is to get to the right answer by asking as few if/else questions as possible



ROOT node

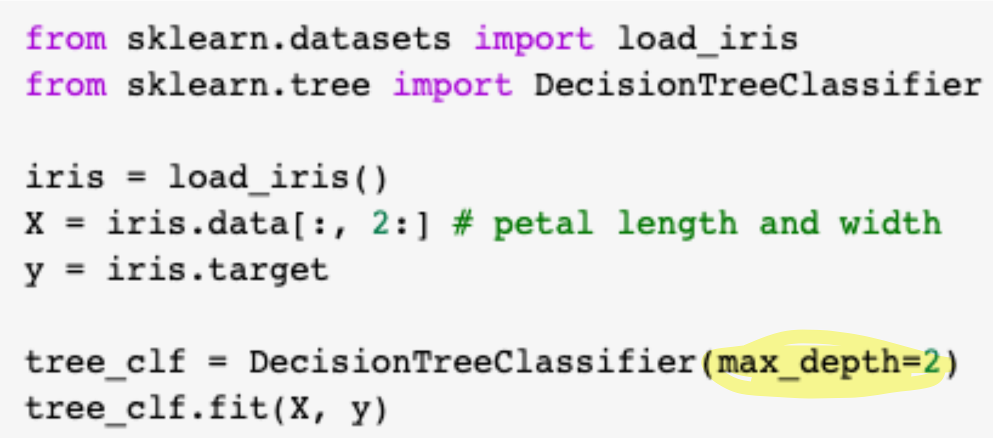
nodes

LEAF nodes

Decision Trees

- Each node in the tree either represents a question or a terminal node (also called a **leaf**) that contains the answer.
- The edges connect the answers to a question with the next question you would ask.
- We built a model to distinguish between four classes of animals (hawks, penguins, dolphins, and bears) using the three features "has feathers," "can fly," and "has fins."
- Instead of building these models by hand, we can learn them from data using supervised learning.

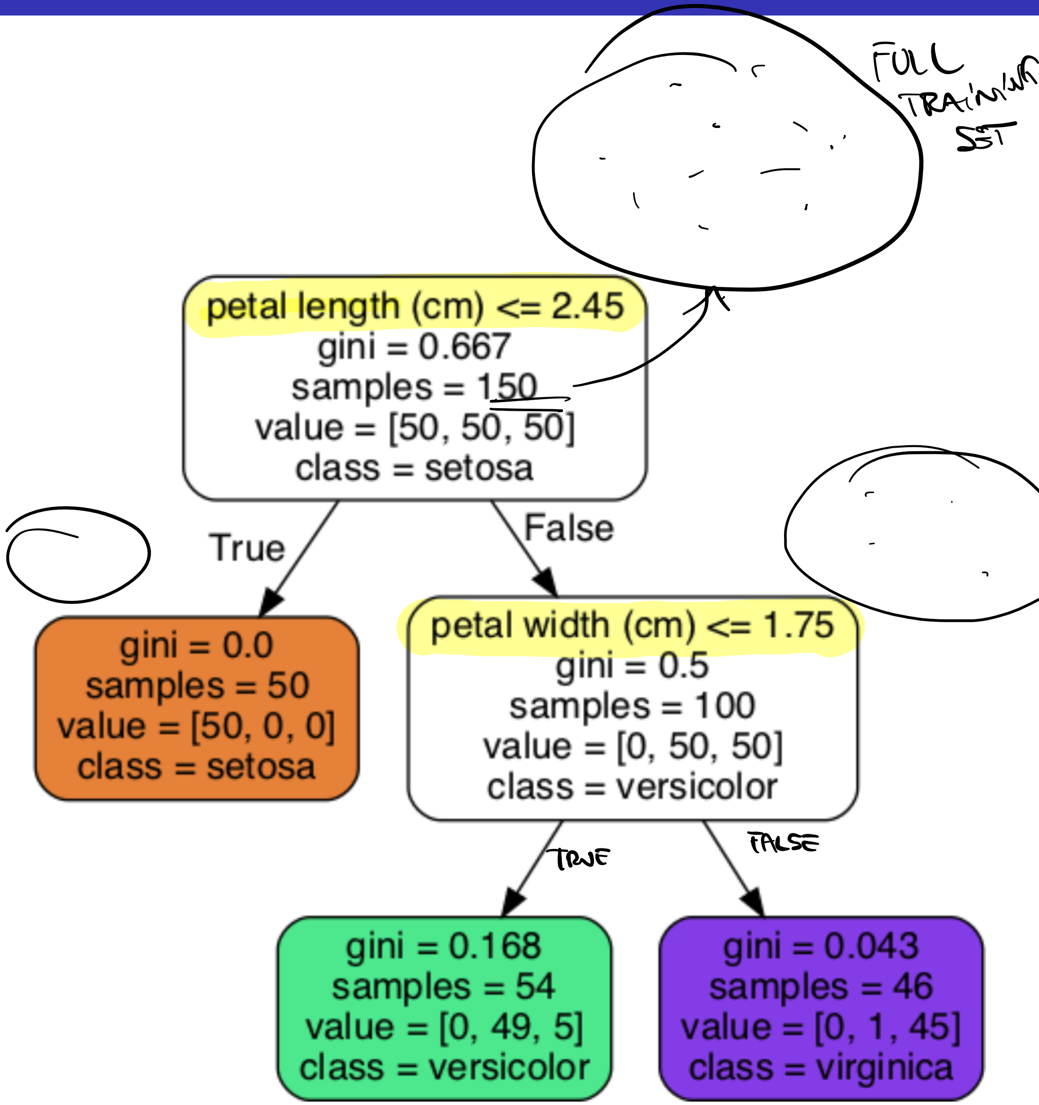
Decision Tree Visualization



```
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)
```



petal length (cm) <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
class = setosa

True

gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa

False

petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor

gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor

gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginica

23/11

Making Predictions

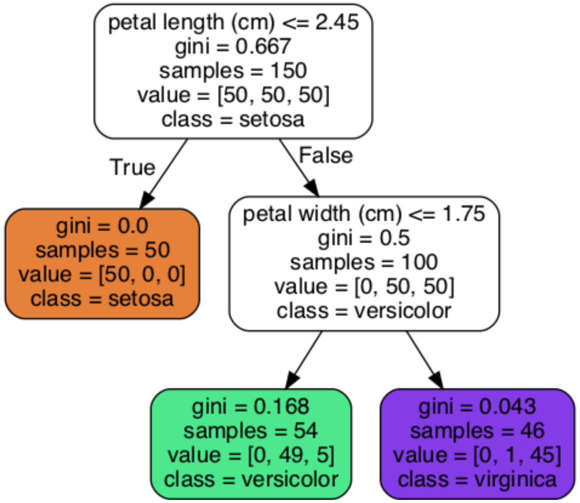
START 28/11

Suppose you find an iris flower and you want to classify it.

- You start at the **root node** (depth 0, at the top): this node asks whether the flower's petal length is smaller than 2.45 cm.
- If it is, then you move down to the root's left child node (depth 1, left). In this case, it is a **leaf node** (i.e., it does not have any children nodes), so it does not ask any questions: you can simply look at the predicted class for that node and the Decision Tree predicts that your flower is an Iris-Setosa (`class=setosa`).

Now suppose you find another flower, but this time the petal length is greater than 2.45 cm.

- You must move down to the root's right child node (depth 1, right), which is not a leaf node
- It asks another question: is the petal width smaller than 1.75 cm?
- If it is, then your flower is most likely an Iris-Versicolor (depth 2, left).
- If not, it is likely an Iris-Virginica (depth 2, right).



petal length (cm) <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
class = setosa

True

False

gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa

petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor

gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor

gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginica

Making Predictions

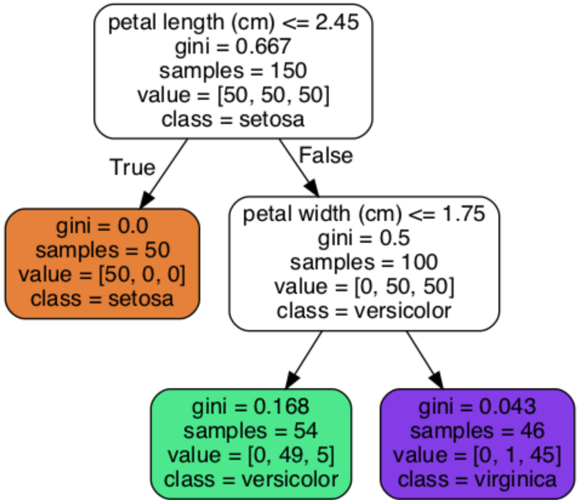
- **Node's Samples**: counts how many training instances it applies to

  Example: 100 training instances have a petal length greater than 2.45 cm (depth 1, right), among which 54 have a petal width smaller than 1.75 cm (depth 2, left).

- **Node's Value**: tells how many training instances of each class this node applies to

  Example: the bottom-right node applies to 0 Iris-Setosa, 1 Iris- Versicolor, and 45 Iris-Virginica.

- **Node's Gini**: measures its impurity
  - a node is *pure* (gini=0) if all training instances it applies to belong to the same class
    Example: since the depth-1 left node applies only to Iris-Setosa training instances, it is pure and its gini score is 0.



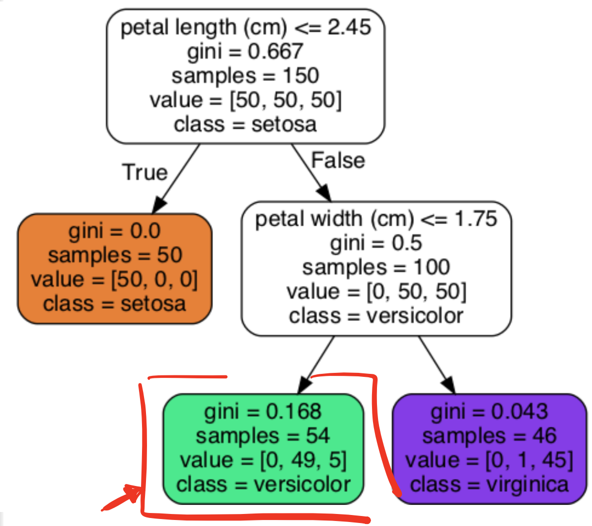
Gini impurity

The training algorithm computes the *gini score* G_i of the i^{th} node with the following formula

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

where

- $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node
- **Example**: the depth-2 left node has a gini score equal to $1 – (\frac{0}{54})^2 - (\frac{49}{54})^2 - (\frac{5}{54})^2 \approx 0.168$.



petal length (cm) <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
class = setosa

True

False

gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa

petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor

gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor

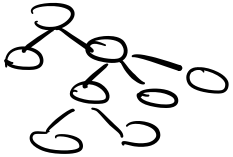
gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginica

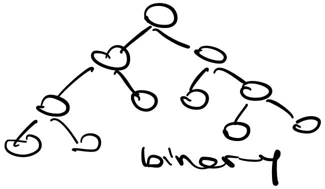
$$G = 1 - \sum_{k=0}^{2} p_k^2 =$$

$$= 1 - p_0^2 - p_1^2 - p_2^2 =$$

$$= 1 - \left(\frac{0}{54}\right)^2 - \left(\frac{49}{54}\right)^2 - \left(\frac{5}{54}\right)^2 = 0{,}168$$

Decision Trees





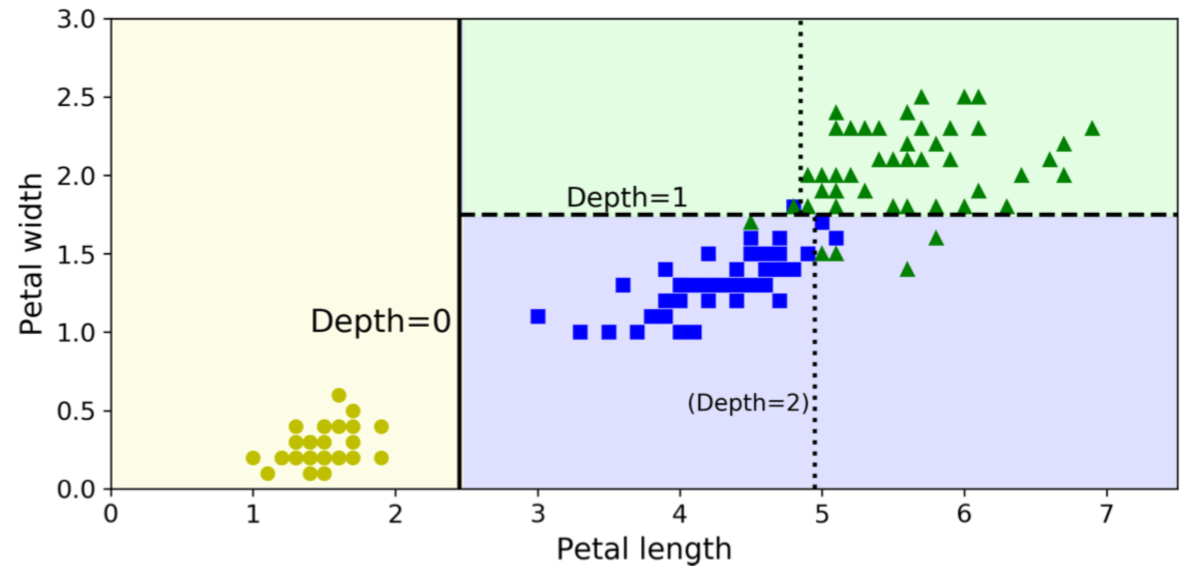
Remark

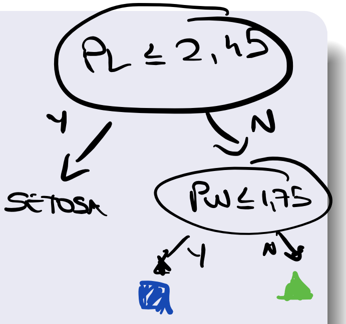
Decision Trees require very little data preparation
→ they do not require feature scaling or centering at all.

Remark

- The Decision Tree CART algorithm (Training Algorithm), produces binary trees
  → non-leaf nodes always have two children (i.e., questions only have yes/no answers).
- Other algorithms (i.e., ID3) can produce Decision Trees with nodes that have more than two children.

Decision Boundaries





- The thick vertical line represents the decision boundary of the root node (depth 0): petal length = 2.45 cm.
- The left area is pure (only Iris-Setosa) ⇒ it cannot be split any further.
- The right area is impure ⇒ the depth-1 right node splits it at petal width = 1.75 cm (represented by the dashed line).
- *max_depth* was set to 2 ⇒ the Decision Tree stops right there.
- if we set *max_depth* to 3 ⇒ the two depth-2 nodes would each add another decision boundary (represented by the dotted lines).

Model Interpretation

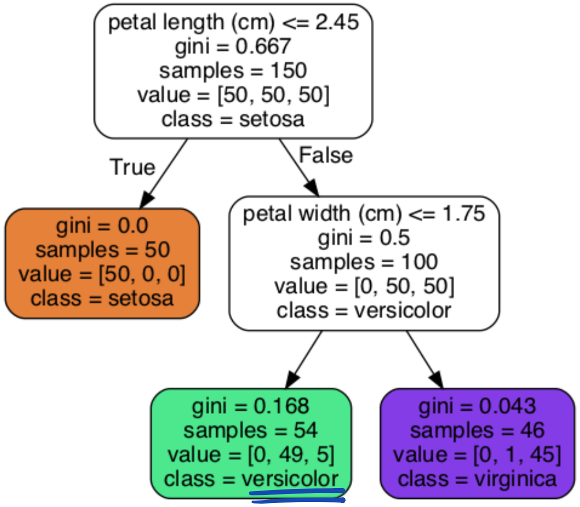
- Decision Trees are fairly intuitive and their decisions are easy to interpret.
- Such models are often called **white box models**.
- in contrast, Random Forests or neural networks are generally considered **black box** models.
  - They make great predictions, and you can easily check the calculations that they performed to make these predictions; nevertheless, it is usually hard to explain in simple terms why the predictions were made.
  - **Example**: if a neural network says that a particular person appears on a picture, it is hard to know what actually contributed to this prediction: *"did the model recognize that person's eyes? Her mouth? Her nose? Her shoes? Or even the couch that she was sitting on?"*
  - Conversely, Decision Trees provide nice and simple classification rules that can even be applied manually if need be (e.g., for flower classification).

Estimating Class Probabilities

A Decision Tree 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
- then it returns the ratio of training instances of class *k* in this node.

Example: PW suppose you have found a flower whose petals are PL 5 cm long and 1.5 cm wide.
→ The corresponding leaf node is the depth-2 left node, so the Decision Tree should output the following probabilities: 0% for Iris-Setosa (0/54), 90.7% for Iris-Versicolor (49/54), and 9.3% for Iris-Virginica (5/54).
→ The predicted class it will be Iris-Versicolor (class 1) since it has the highest probability.



petal length (cm) <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
class = setosa

True

False

gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa

petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor

gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor

gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginica

$$\mathbb{P} = [p_0, p_1, p_2] = \left[\frac{0}{54}; \frac{49}{54}; \frac{5}{54}\right]$$

CART training algorithm

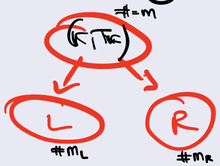
Classification and Regression algorithm - CART

- This is one of the algorithms used to train Decision Trees, aka *growing* trees

ROOT NODE

- 1 The algorithm first splits the training set in two subsets using a single feature k and a threshold t_k

  Example: `petal length` ≤ 2.45 `cm`.

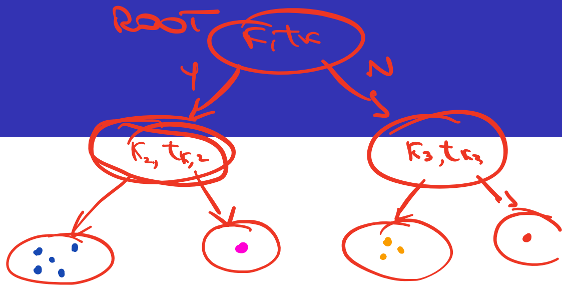


- the pair (k, t_k) is chosen in order to produce the purest subsets (weighted by their size)
- The cost function that the algorithm tries to minimize is

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

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

CART training algorithm



Classification and Regression algorithm - CART (cont.)

- Once it has successfully split the training set in two, it splits the subsets using the same logic
- It recursively splits the sub-subsets until it reach a **stop condition**
- Different *Stop Conditions* might be defined
    - ① maximum depth reached
    - ② the algorithm cannot find a split that will reduce impurity
    - ③ a node reached the minimum number of samples a node must have before it can be split, or the ④ minimum number of samples a leaf node must have,
    - ⑤ the model reached the maximum number of leaf nodes.

CART training algorithm

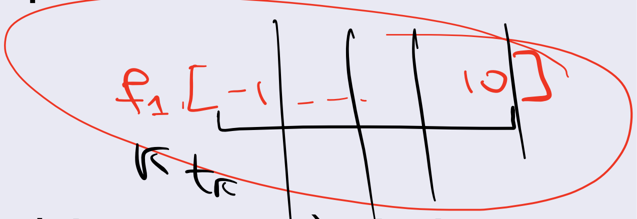
- CART algorithm is a *greedy algorithm*: it greedily searches for an optimum split at the top level, then repeats the process at each level.
- It does not check whether or not the split will lead to the lowest possible impurity several levels down.
- A greedy algorithm often produces a reasonably good solution, but it is not guaranteed to be the optimal solution.
- Finding the optimal tree is known to be an NP-Complete problem: it requires $O(exp(m))$ time

Computational Complexity

- Making predictions requires **traversing the Decision Tree from the root to a leaf**
- Traversing the Decision Tree requires going through roughly $O(log_2(m))$ nodes (if balanced tree).
- Since each node only requires checking the value of one feature, the overall prediction complexity is just $O(log_2(m))$, independent of the number of features.

  ⇒ predictions are very fast, even when dealing with large training sets.
- The training alg., instead, compares all features (or less if a `maximum number of features` has been set) on all samples at each node.
- The training complexity is $O(n \times mlog(m))$

  n is the number of features, m is the number of samples



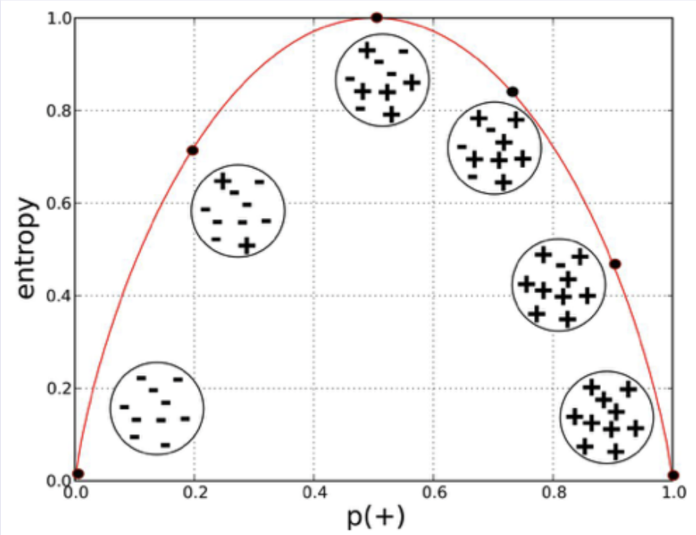
- For small training sets (less than a few thousand instances), it is possible to speed up the training by presorting the data
  - Scikit-Learn allows this operation by means of a parameter 'presort' but this slows down training considerably for larger training sets.

Gini Impurity VS Entropy

- Both **Gini** or **Entropy** might be used as measure of impurity
- definition of the entropy of the i^{th} node

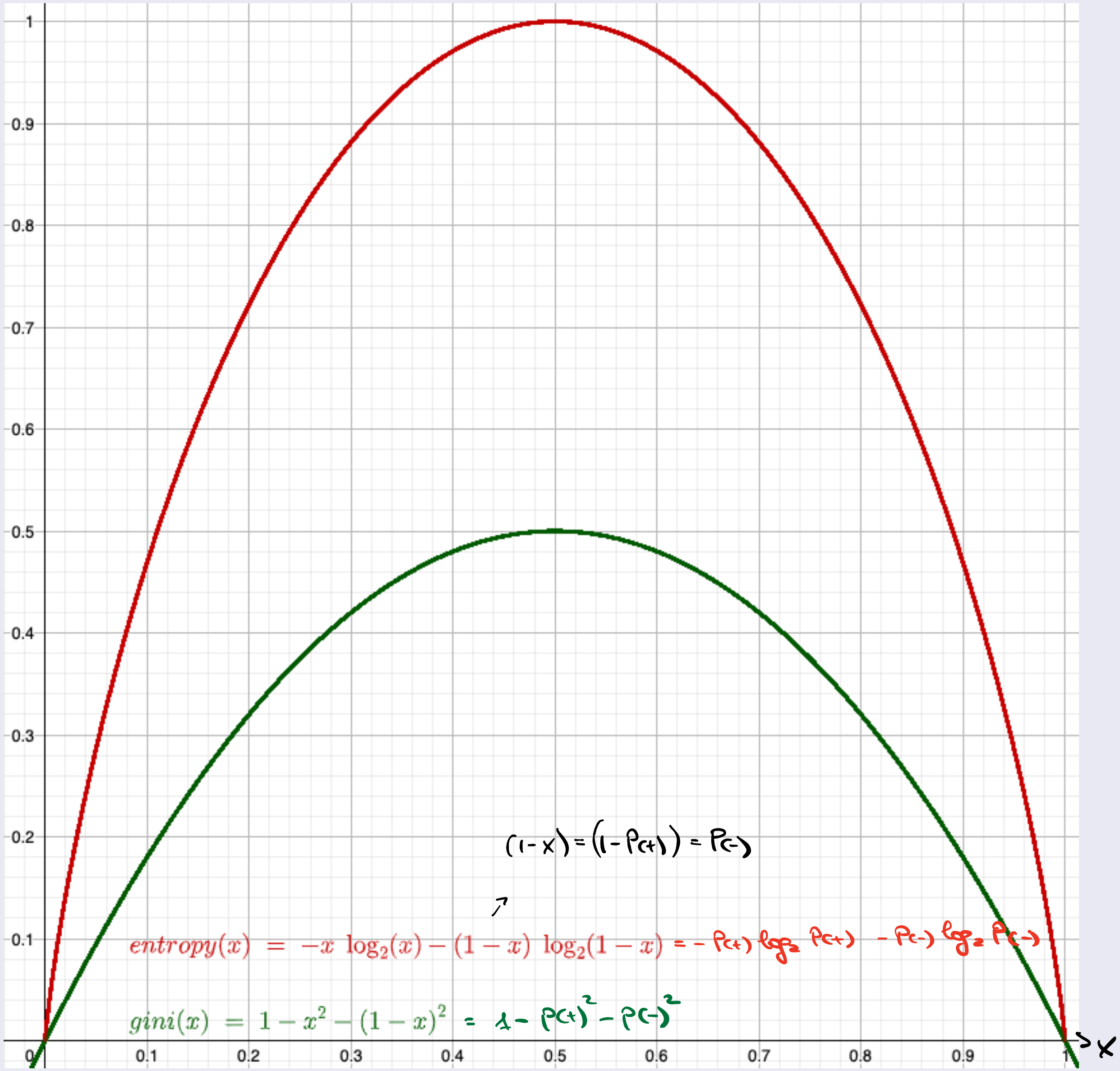
$$H_i = -\sum_{k=1}^{n} p_{i,k} \log_2(p_{i,k}), \text{ for all } p_{i,k} \neq 0$$

- $p_{i,k}$ is the relative frequency of class k at node i
- a set's entropy is zero when it contains instances of only one class



Gini VS Entropy

Binary Classification Task



$(1-x) = (1-P(+)) = P(-)$

$entropy(x) = -x\ \log_2(x) - (1-x)\ \log_2(1-x) = -P(+) \log_2 P(+) - P(-) \log_2 P(-)$

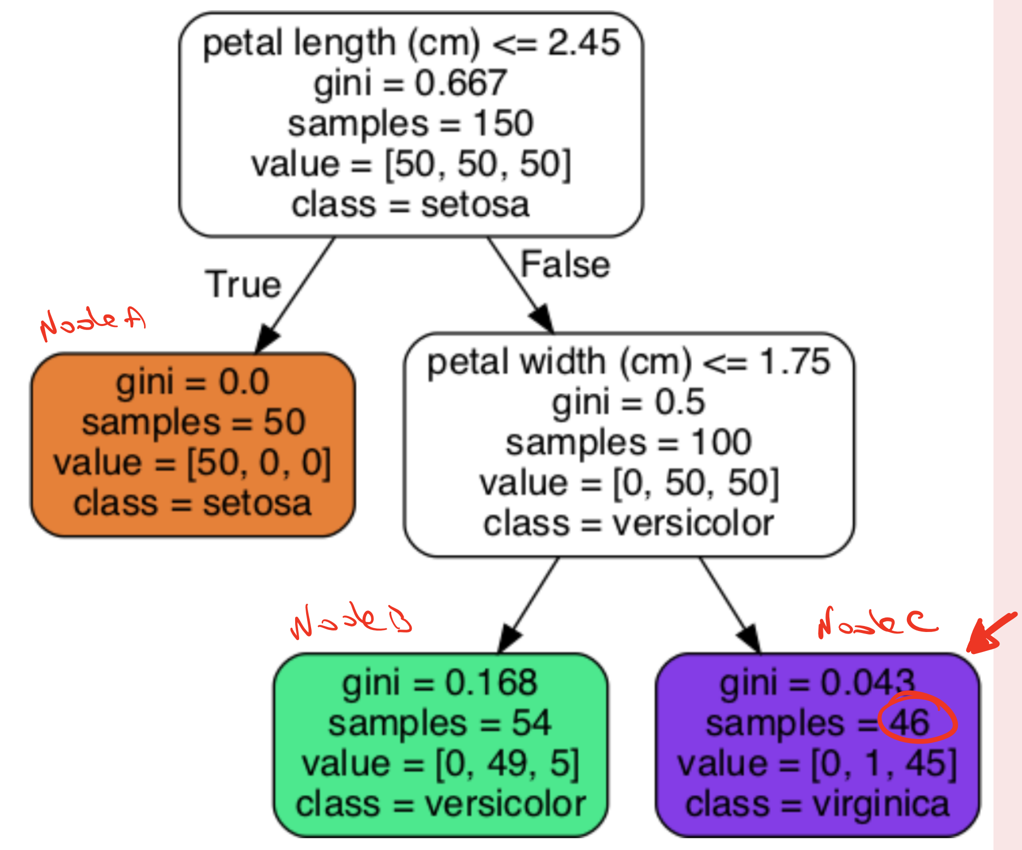
$gini(x) = 1 - x^2 - (1-x)^2 = 1 - P(+)^2 - P(-)^2$

$x = P(+)$

Entropy

PRACTICE 1

What is the entropy value of each leaf in the following tree?



petal length (cm) <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
class = setosa

True

False

gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa

petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor

gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor

gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginica

$H_c = ?$

$$H_c = -\sum_{k=0}^{2} p_{c,k} \log_2(p_{c,k}) \qquad p_{c,k} \neq 0$$

$p_{c,0} = \frac{0}{46} = \varnothing$ -> not considered

$p_{c,1} = \frac{1}{46}$

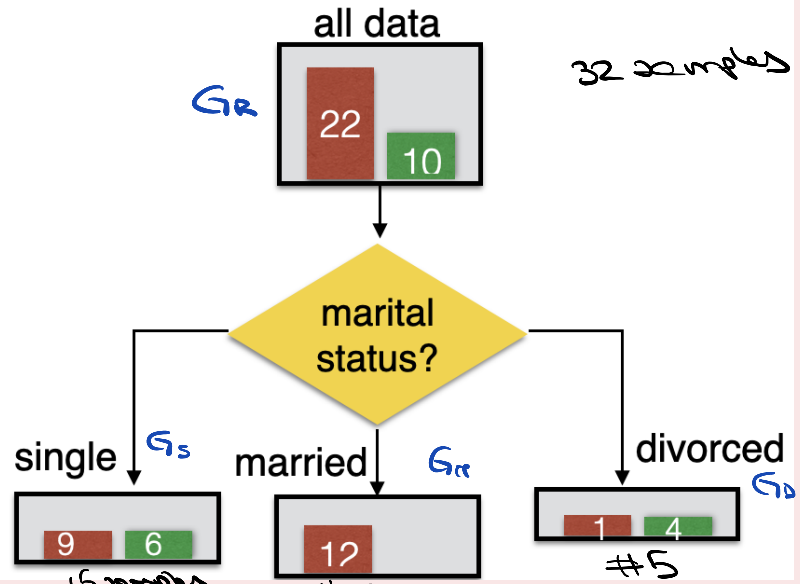
$p_{c,2} = \frac{45}{46}$

$$H_c = -\frac{1}{46}\log_2\left(\frac{1}{46}\right) - \frac{45}{46}\log_2\left(\frac{45}{46}\right) = ?$$

Gini

PRACTICE 2

- 1 What is the Gini impurity score of the root node? G_R
- 2 Calculate the Gini impurity score for each leaf G_S, G_M, G_D
- 3 Calculare the total Gini impurity score of the tree after the split (weighted average of Gini impurity score per leaf, weighed on the number of items in the leaf)



32 samples in total (TRAINING set)

15 samples

12

#5

The algorithm decides to split if this reduces the total Gini impurity.

$$G_{\substack{INI \\ node}} = 1 - \sum_{k=1}^{n} p_k^2$$

END
28/11

START
30/11

① $G_{ROOT} = 1 - \underbrace{\left(\frac{22}{32}\right)}_{P}^2 - \left(\frac{10}{32}\right)^2 = \boxed{0,42}$

$$P_{RED,ROOT} = \frac{\#\text{red samples in ROOT}}{\#\text{samples in ROOT}}$$

$$P_{GREEN,ROOT} = \frac{\#\text{green samples in ROOT}}{\#\text{samples in ROOT}}$$

② $G_S = 1 - \left(\frac{9}{15}\right)^2 - \left(\frac{6}{15}\right)^2 = \frac{12}{25} = 0,48$

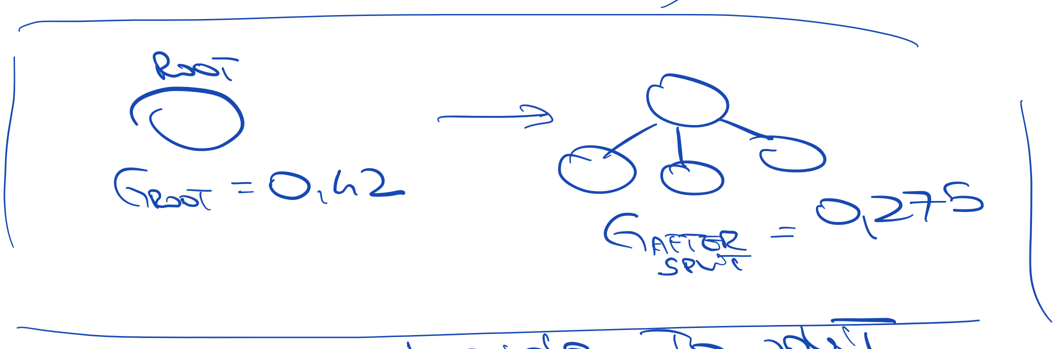
$G_{\pi} = 1 - \left(\frac{12}{12}\right)^2 - \left(\frac{0}{12}\right)^2 = \emptyset$

$G_D = 1 - \left(\frac{1}{5}\right)^2 - \left(\frac{4}{5}\right)^2 = 0,32$

③ $G_{\text{AFTER THE SPLIT}} = \left(\frac{15}{32}\right) G_S + \left(\frac{12}{32}\right) G_{\pi} + \left(\frac{5}{32}\right) G_D$

$$\frac{\#\text{samples in the node}}{\text{Tot }\#\text{samples in the ROOT}}$$

$= \left(\frac{15}{32}\right) \cdot 0,48 + \cancel{\left(\frac{12}{32}\right) \cdot 0} + \left(\frac{5}{32}\right) \cdot 0,32 = \boxed{0,275}$



we decide to split
because Gimpurity is reduced

Gini VS Entropy

- Choosing Entropy or Gini does not make a big difference most of the time: they lead to similar trees
- Gini impurity is slightly faster to compute
- Gini impurity tends to isolate the most frequent class in its own branch of the tree
- Entropy, instead, tends to produce slightly more balanced trees

Regularization Hyperparameters

- Decision Trees make very few assumptions about the training data (as opposed to linear models)
- If left unconstrained ⇒ the tree structure will adapt itself to the training data, fitting it very closely, and most likely overfitting it.
- Such a model is often called a *nonparametric model*: the number of parameters is not determined prior to training, so the model structure is free to stick closely to the data.
- It is opposed to the *parametric model*: which has a predetermined number of parameters, so its degree of freedom is limited, reducing the risk of overfitting (but increasing the risk of underfitting).
    - **Example**: linear models

Regularization Hyperparameters

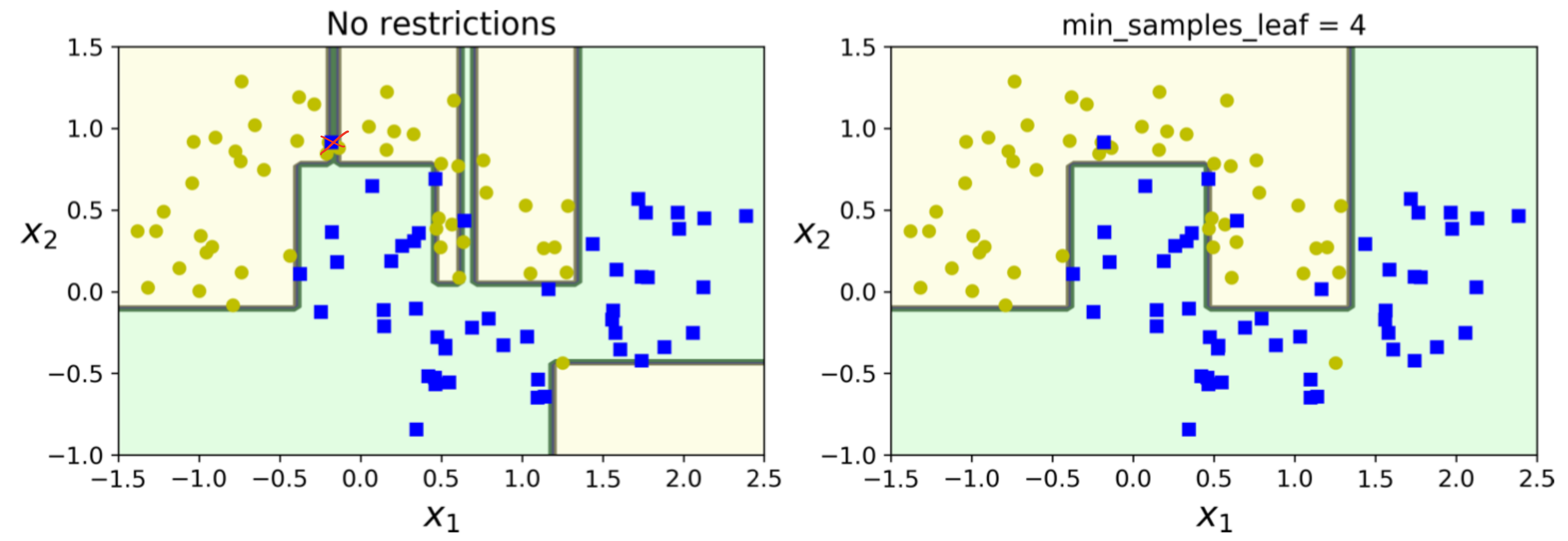
- To avoid overfitting the training data, we need to restrict the Decision Tree's freedom during training: **regularization**.
- There are different regularization hyperparameters for decision trees:
  - we can restrict the maximum depth of the Decision Tree: reducing the depth will regularize the model and thus reduce the risk of overfitting.
  - we can increase the minimum number of samples a node must have before it can be split
  - we can increase the minimum number of samples a leaf node must have
  - we can restrict the maximum number of leaf nodes a tree might have

Regularization procedures

Pruning

- An other methodology performed to regularize a tree foresees first the training of the Decision Tree without restrictions, then **pruning** (deleting) unnecessary nodes.
- A node whose children are all leaf nodes is considered unnecessary if the purity improvement it provides is not *statistically significant*
- Standard statistical tests, such as the χ^2 *test*, are used to estimate the probability that the improvement is purely the result of chance (which is called the *null hypothesis*).
- If this probability, called the *p- value*, is higher than a given threshold (typically 5%, controlled by a hyperparameter), then the node is considered unnecessary and its children are deleted.
- The pruning continues until all unnecessary nodes have been pruned.

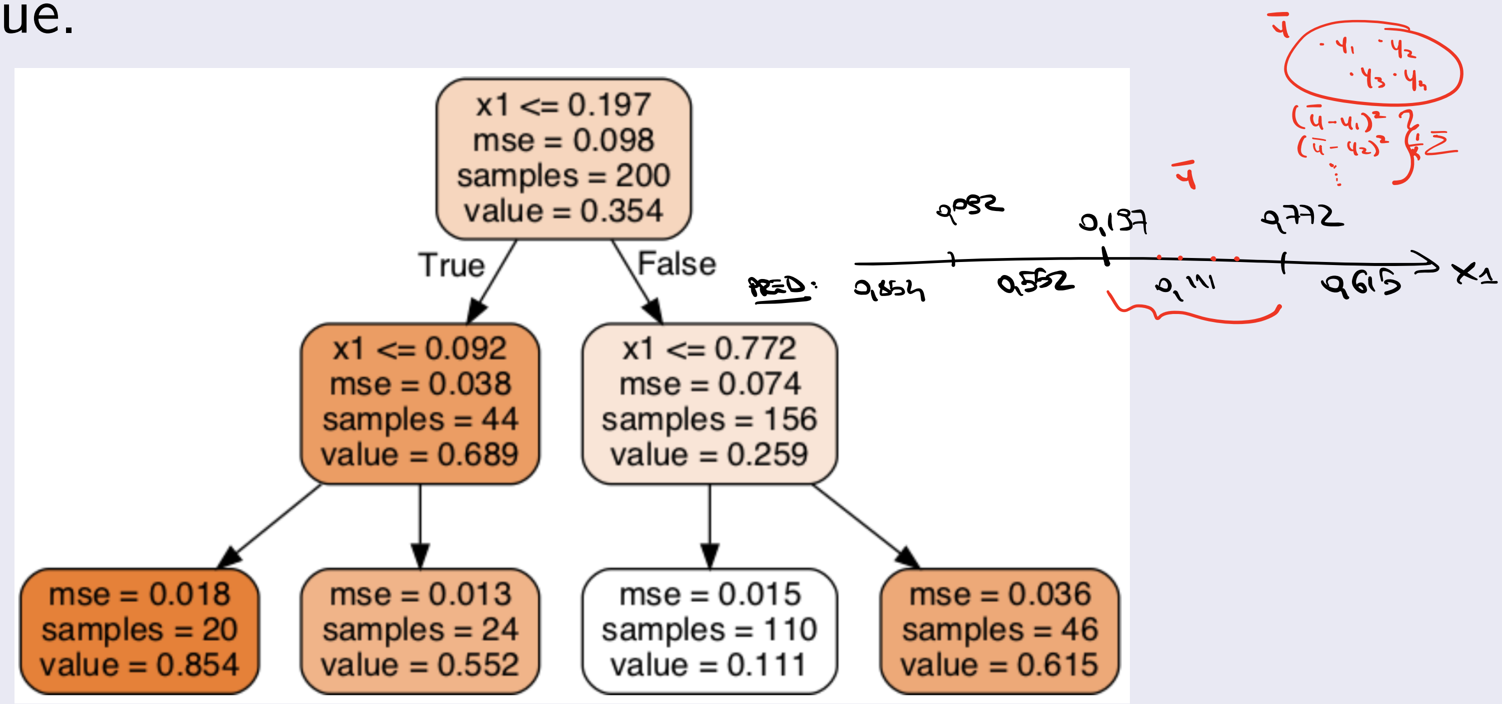
Regularization example



- **LEFT**: Decision Tree is trained unconstrained (no restrictions)
- **RIGHT**: Decision Tree is trained with a constraint for the minimum number of samples per leaf $= 4$
- The model on the left is overfitting, and the model on the right will probably generalize better.

Decision Trees for regression tasks

- Decision Trees are also capable of performing regression tasks.
- Instead of predicting a class in each node, it predicts a value.
- The prediction is simply the average target value of the training instances associated to the reached leaf node.
- Mean Squared Error (MSE) per leaf is also calculated, for the related prediction value.



x1 <= 0.197
mse = 0.098
samples = 200
value = 0.354

True

False

x1 <= 0.092
mse = 0.038
samples = 44
value = 0.689

x1 <= 0.772
mse = 0.074
samples = 156
value = 0.259

mse = 0.018
samples = 20
value = 0.854

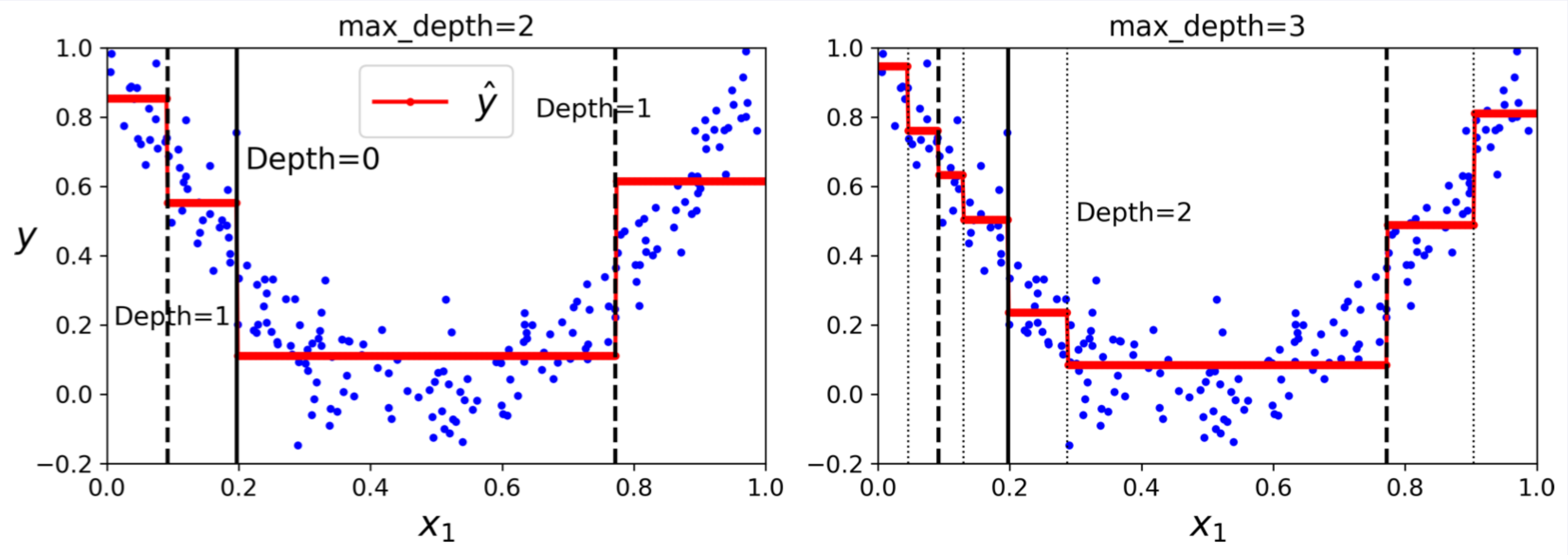
mse = 0.013
samples = 24
value = 0.552

mse = 0.015
samples = 110
value = 0.111

mse = 0.036
samples = 46
value = 0.615

Decision Trees for regression tasks

Prediction

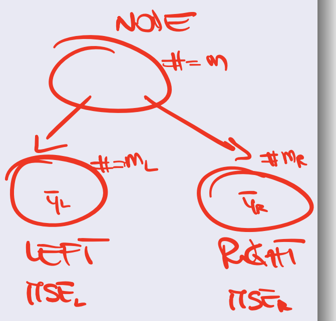


- The predicted value for each region is always the average target value of the instances in that region.
- The algorithm splits each region in a way that makes most training instances as close as possible to that predicted value.

CART cost function for Regression

CART for Regression

- The CART algorithm works mostly the same way as described above
- It tries to split the training set in a way that minimizes the MSE.
- The cost function that the algorithm tries to minimize is:



$$J(\underset{\overset{\uparrow}{\text{feature}}}{k}, \underset{\overset{\nwarrow}{\text{Th}}}{t_k}) = \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right}$$

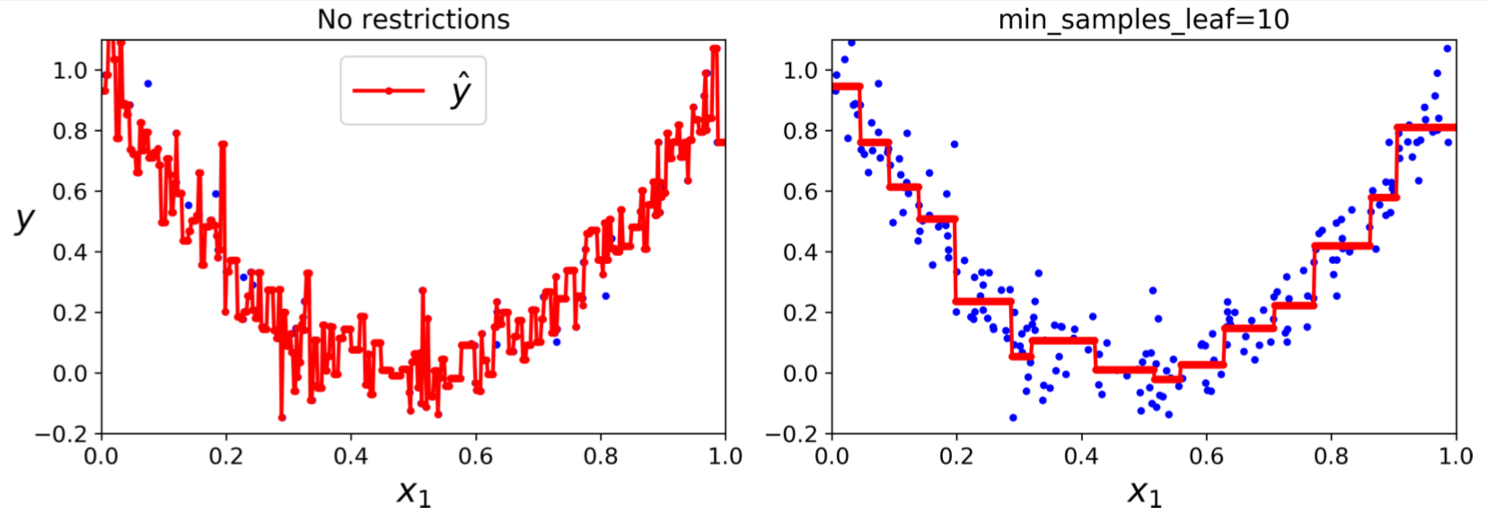
where

$$\begin{cases} MSE_{node} = \frac{1}{m_{node}} \sum_{i \in node} (\overset{\nearrow \text{average}}{\boxed{\hat{y}_{node}}} - y^{(i)})^2 \\ \hat{y}_{node} = \frac{1}{m_{node}} \sum_{i \in node} y^{(i)} \end{cases}$$

Regularization

Regularization

- Decision Trees are prone to overfitting also when dealing with regression tasks.



- **LEFT**: prediction ~~form~~ FROM model without regularization → it is obviously overfitting the training set very badly.
- **RIGHT**: just setting the minimum number of samples per leaf $= 10$ results in a much more reasonable model

Instability

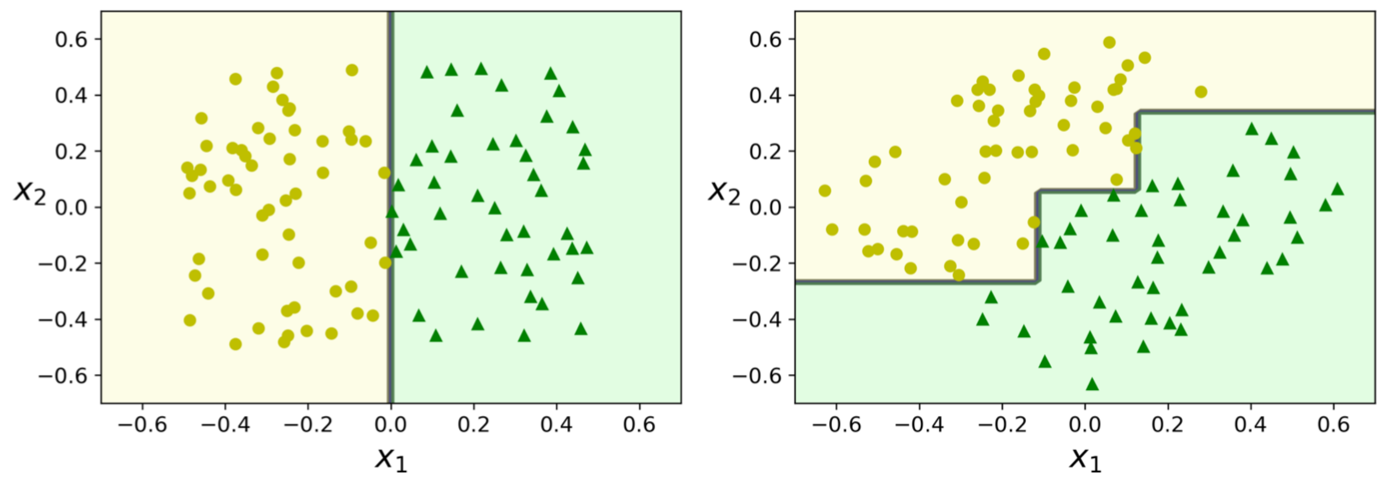
Pros and Cons

(+) DTs are simple to understand and interpret, easy to use, versatile, and powerful.

(−) DTs have orthogonal decision boundaries (all splits are perpendicular to an axis), which makes them sensitive to training set rotation.

(−) DTs are very sensitive to small variations in the training data.

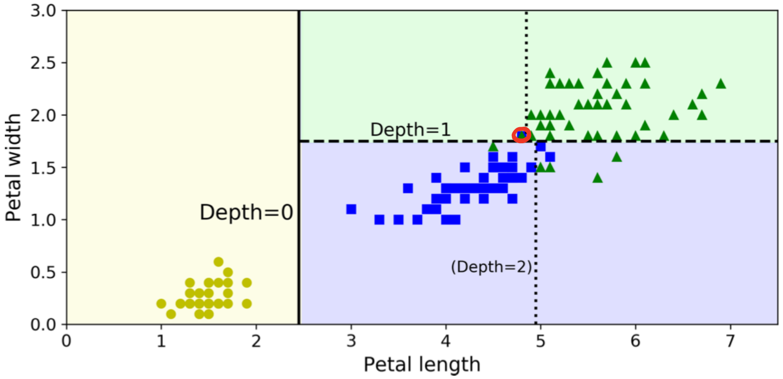
DT sensitive to training set rotation: example



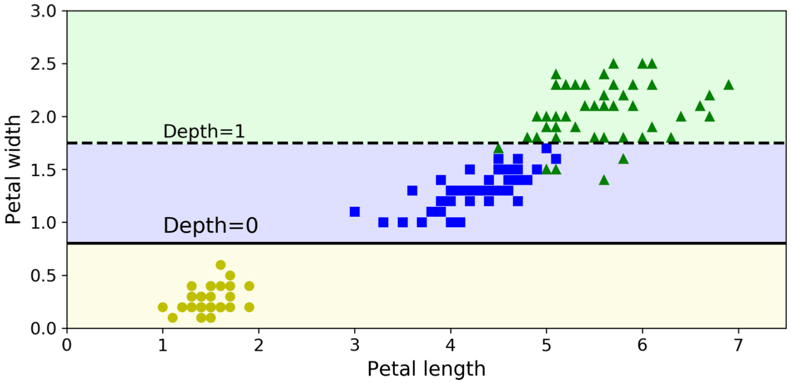
Example in figure:

- **LEFT**: simple linearly separable dataset ⇒ a Decision Tree can split it easily
- **RIGHT**: the dataset is rotated by 45° ⇒ the decision boundary looks unnecessarily convoluted.
- Although both Decision Trees fit the training set perfectly, it is very likely that the model on the right will not generalize well.
- One way to limit this problem is to use PCA, which often results in a better orientation of the training data.

DT sensitivity to small variations in training set



- The main issue with Decision Trees is that they are **very sensitive to small variations in the training data**.
- For example, if we remove the widest Iris-Versicolor from the iris training set (petals 4.8 cm long and 1.8 cm wide) and train a new Decision Tree ⇒ we might get a different model.
- Training algorithm is stochastic ⇒ we might get very different models even on the same training data



References

Aurélien Géron (2019)
Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow
Chapter 6: Decision Trees (pp. 177 – 190)
Published by O'Reilly Media, Inc..