BTS | Barcelona Technology School

Classical Data Analysis

Master in Big Data Solutions 2020-2021

Filipa Peleja

Víctor Pajuelo

filipa.peleja@bts.tech

victor.pajuelo@bts.tech

Today's class

Contents

Today's Objective

1. Get to know Nonparametric models through Decision Trees
2. Get acquainted with Decision Trees potential and pitfalls
3. Develop an intuition for the need of Ensemble Models and Random Forests

Let's git things done!

Let’s see it again

Pull Session 6 notebooks

$ git clone https://github.com/vfp1/bts-cda-2020.git

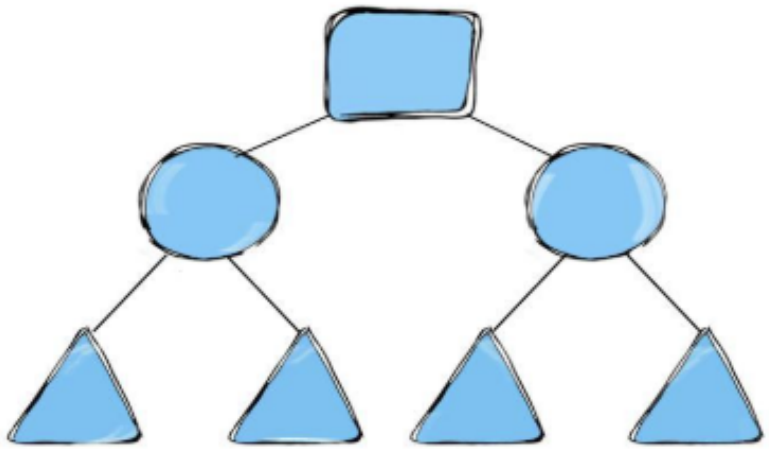
Decision Trees

Introduction

Decision Trees – Introduction

Definitions

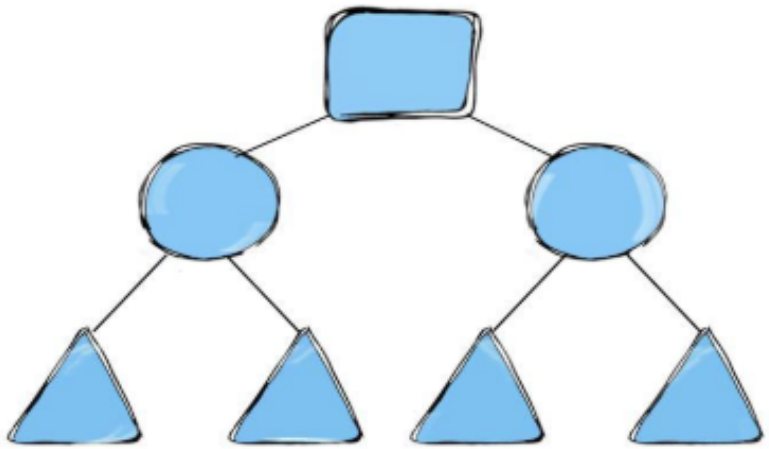
- Decision Trees (DTs from now on) are a powerful and versatile tool that can be used to solve a wide-range of problems
- DTs can be scratched out by hand to make fast decisions
- DTs can also be coded into an algorithm to approach complex scenarios



Decision Trees – Introduction

Definitions

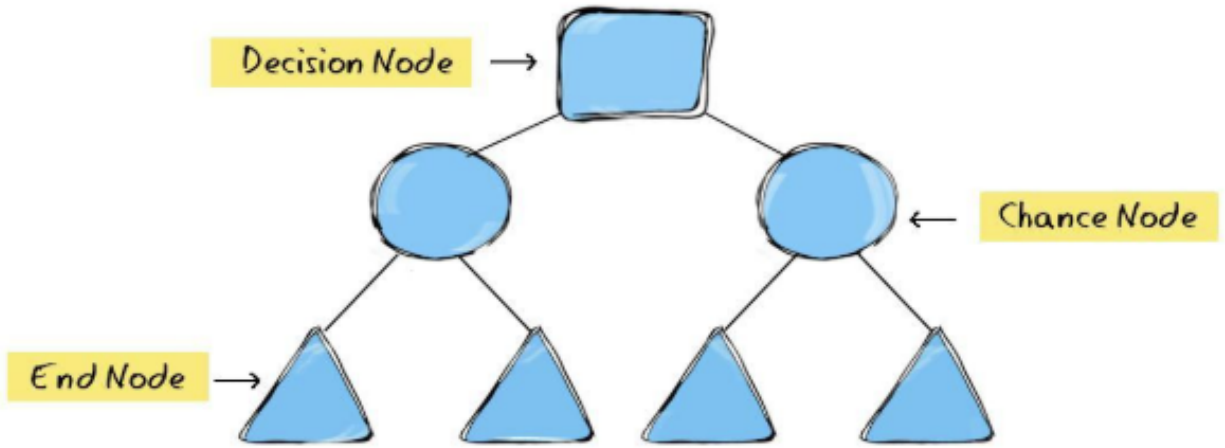
- DTs can be described in different ways:
  - A **tool** that helps **making decisions** by exploring outcomes and scenarios
  - A **graph** that uses **branching methods** to depict a course of action and its various outcomes
  - A **flowchart** that helps making decisions exploring outcomes
  - It helps to portray scenarios and consequences that you might not normally think of



Decision Trees – Introduction

Elements of a decision tree

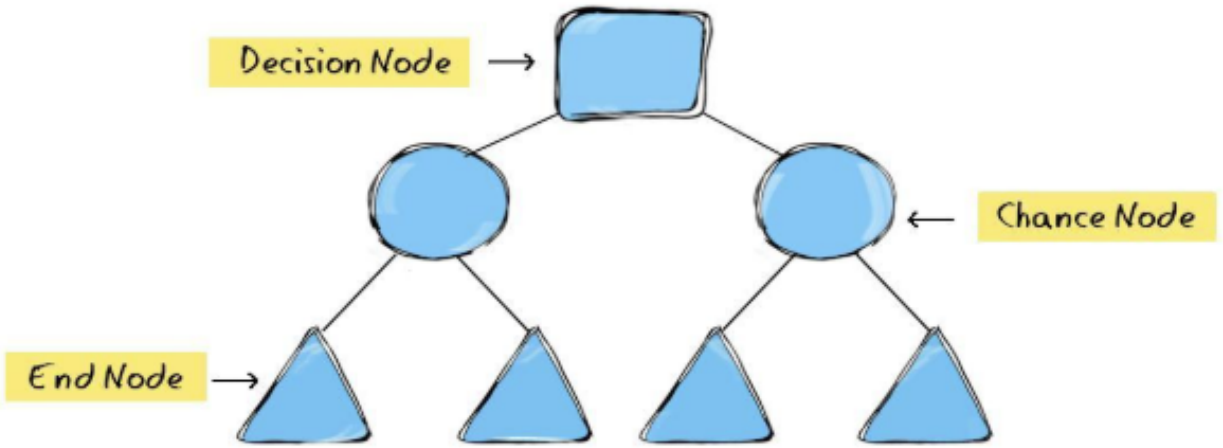
- **Nodes:** the “bubbles” or main elements in a tree are called *nodes* and there are three types:
  - **Decision node:** it involves to make a decision between two options. Sometimes called *conditions* or *internal nodes*.
  - **Chance node:** it represents an uncertain result.
  - **End node:** final answer, it does not split any further. They are also called *terminal nodes* or *leaf nodes.*



Decision Trees – Introduction

Elements of a decision tree

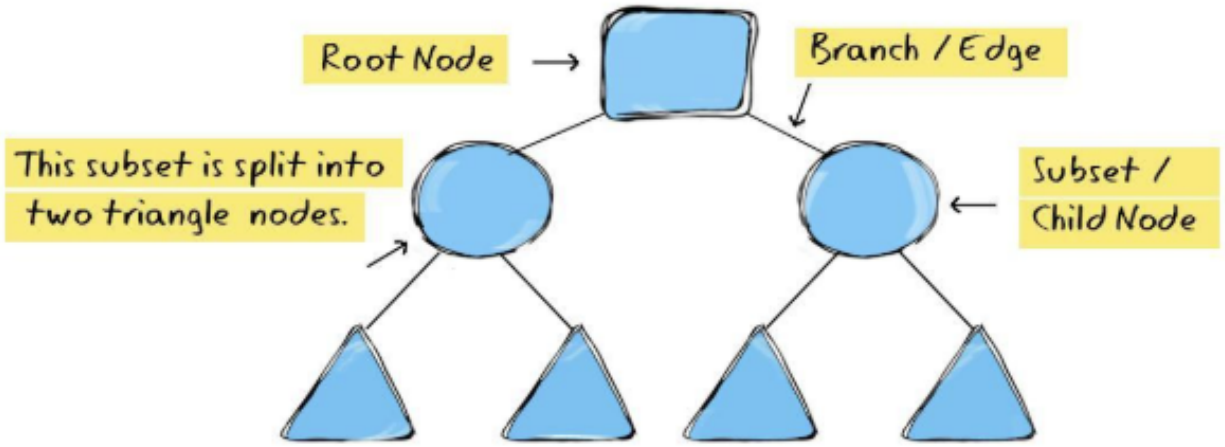
- **Nodes:** the “bubbles” or main elements in a tree are called *nodes* and there are three types:
  - **Decision node:** *SHOULD I BUY OR RENT A HOUSE?*
  - **Chance node:** *WILL INTEREST RATES GO UP OR DOWN?*
  - **End node:** *YES / NO*



Decision Trees – Introduction

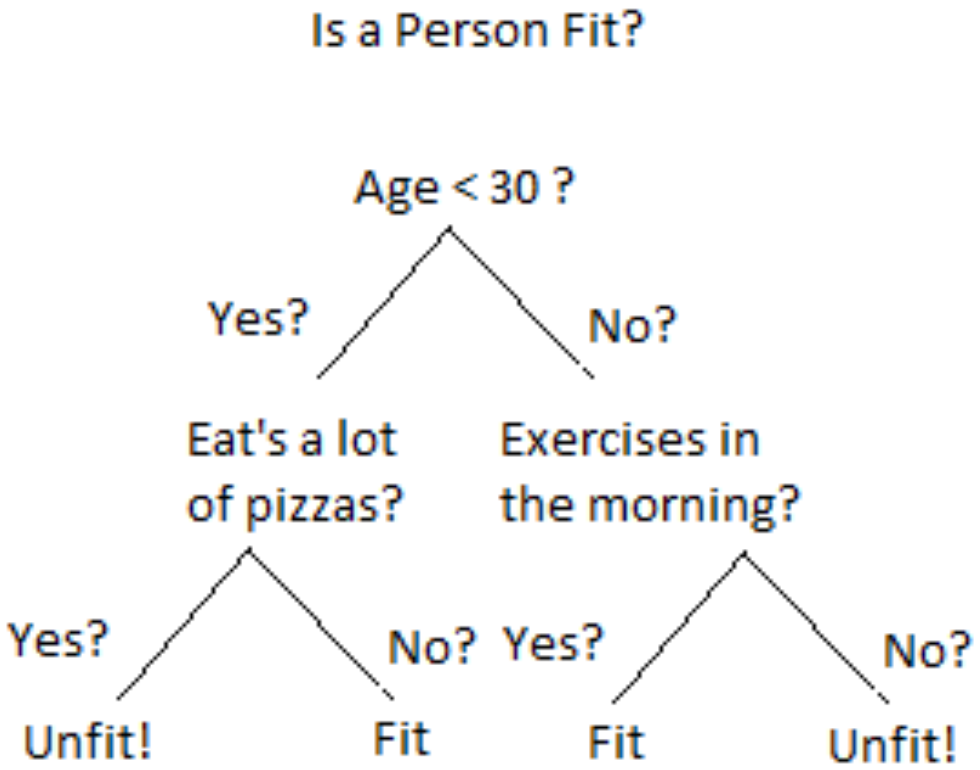
Elements of a decision tree

- **Root node:** the original question/decision that *starts the three*
- **Child node/subset:** the *result of splitting a node* into new nodes
- **Splitting:** *dividing a node* into two or more subsets
- **Pruning:** opposite of splitting, *removing a subset node from a decision node*. It involves removing branches of low importance.
- **Branch:** also called *edge*, is a subsection of an entire tree that connects nodes



Decision Trees – Introduction

Basic example



Decision Trees – Introduction

Practical uses

- Some practical examples of using DTs for our daily life:
  - Decide whether to buy or rent a home
  - Decide whether to purchase a product or build it yourself
  - Decide to rent a car or buy it
- DTs in the industry:
  - Assessing species mortality
  - Classifying medical text
  - Diagnosing medical disorders such as thyroid disease

- DTs are very useful, but the decisions are as good as our questions or our capability to design the splitting nodes

Decision Trees

Building it by hand

Decision Trees – Build by hand

Guide to tackle the Decision Tree building

- DTs have been historically created by hand. The origin of the usage of the technique in discussed nicely in this post
- Usually, one of the simplest approaches to build a DT for your daily decisions is as follows:
  1. Determine the **initial question/problem**
  2. Determine your **decision** and **unknown**
  3. Determine the **values**
  4. Determine the **probabilities**
  5. Calculate the **weighted value**
  6. Calculate the **net benefit** of **each decision**
- Let's portray this with a simple scenario

Decision Trees – Build by hand

Decision scenario

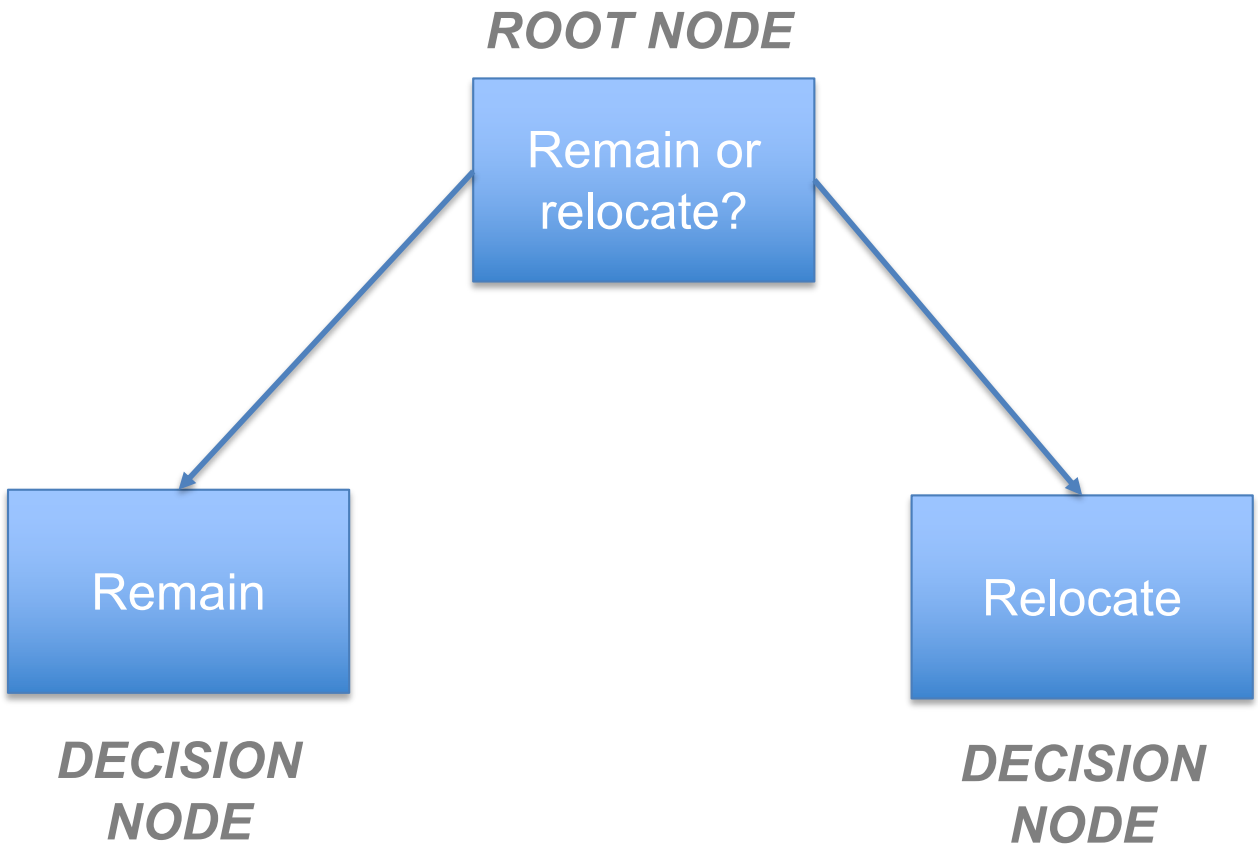
- Extremely simplified case for building up a DT:

We own a gourmet coffee shop and sales have been amazing. However, we are in a residential area, and we believe that by going to Poblenou, which has more offices, we will increase our sales. However, we don't know whether the sales are going to be strong or weak next season. If demand is strong, our profits will be substantial and help pay relocating debt. If demand is slow, we will be in trouble. What should we do?

Decision Trees – Build by hand

Determine the initial question/problem

- Should I buy or rent ?
  - This is the question of the **root node** from which **two decision** nodes flow



Decision Trees – Build by hand

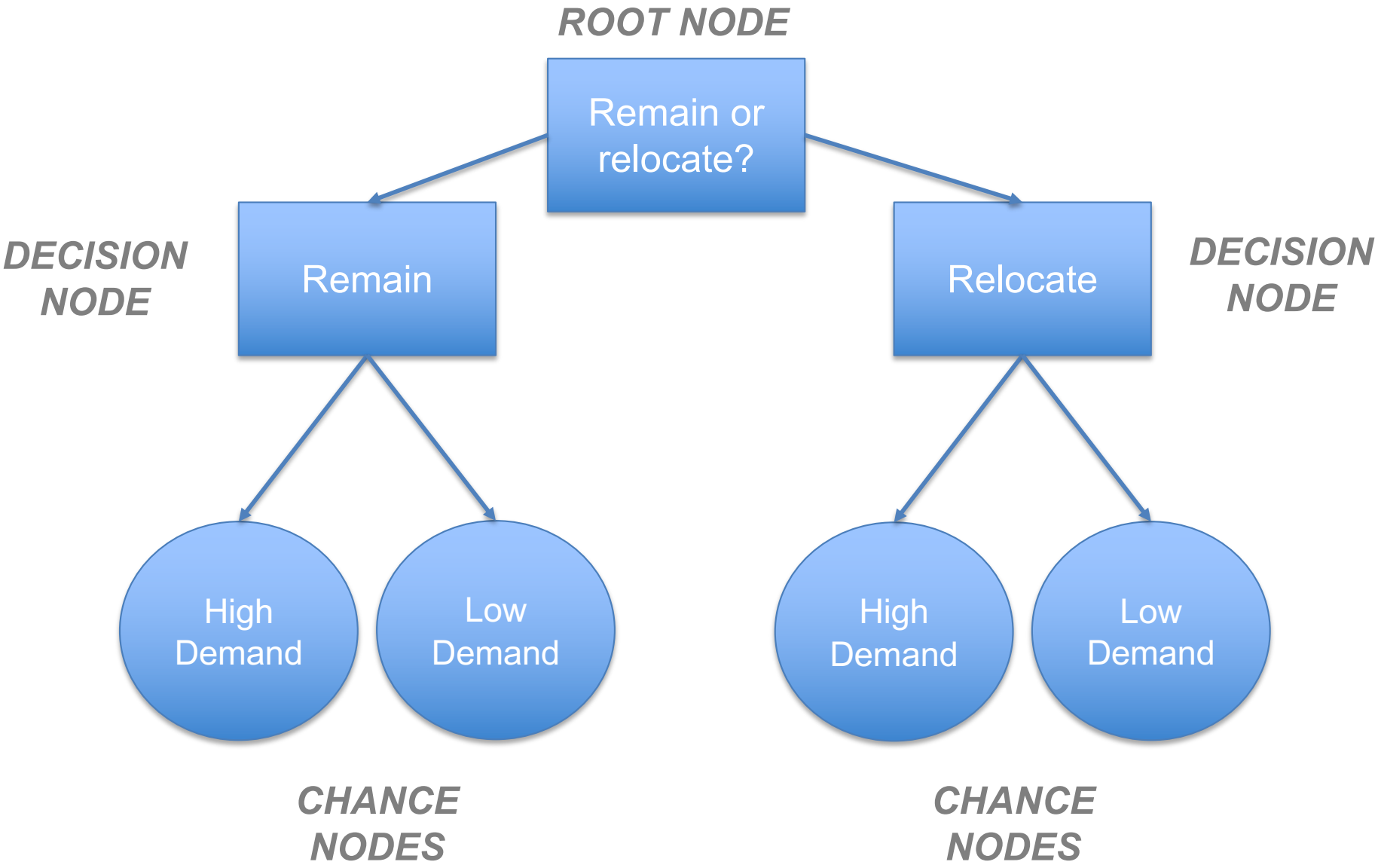
Determine the decision and unknown

- Our problem has **one type of decision** (remain/relocate) and **one unknown** (High/Low demand) so this leads to **4 possible outcomes**:

  - We have 4 potential *chance nodes:*
    - Relocate and High Demand
    - Relocate and Low Demand
    - Remain and High Demand
    - Remain and Low Demand

Decision Trees – Build by hand

Determine the decision and unknown



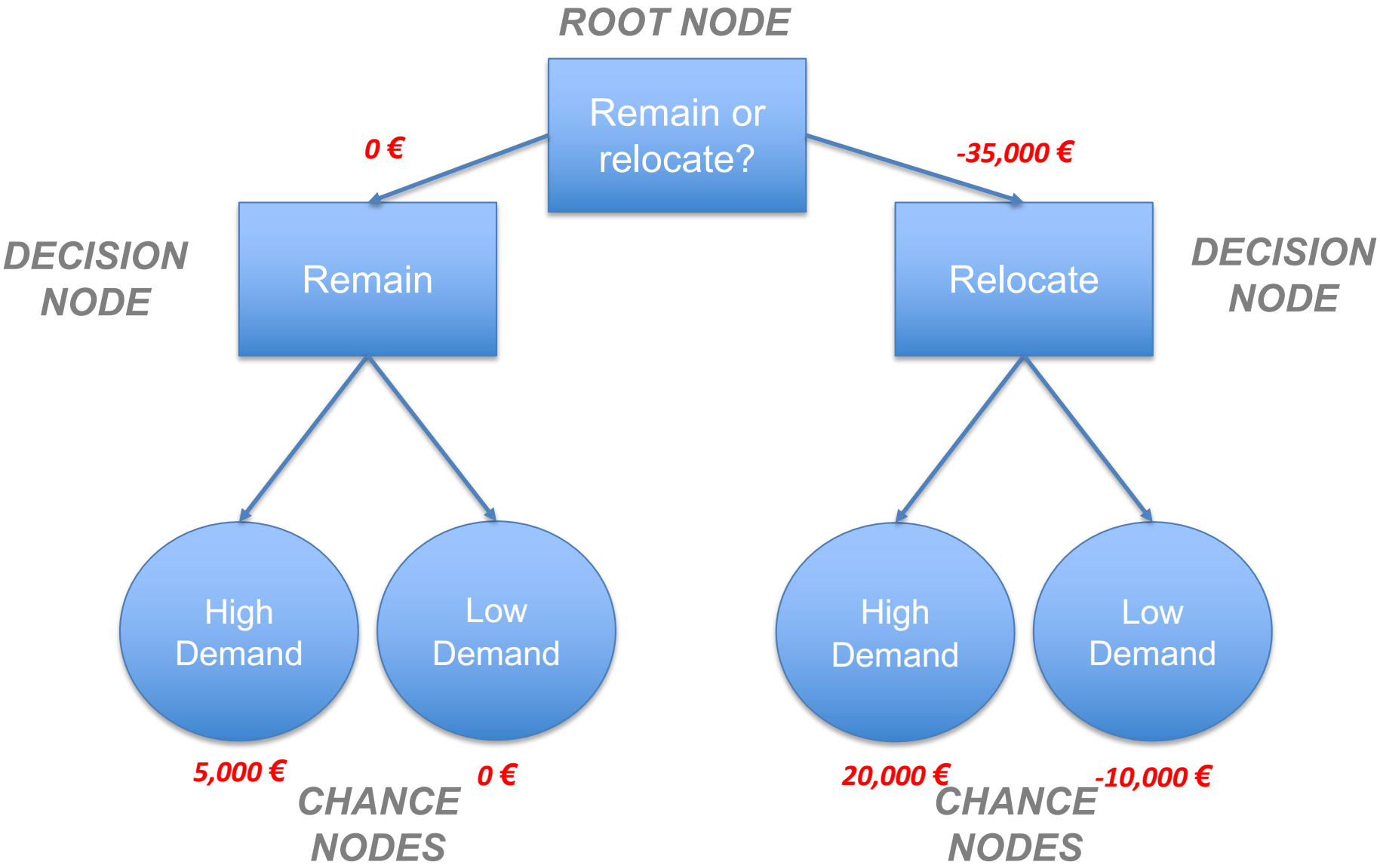
Decision Trees – Build by hand

Determine the Values of Each Branch

Node Concept	Node Value
Relocating coffee shop	35,000 €
Remaining at the current location	0 €
Expected profits if we **relocate** and **demand is high**	20,000 €
Expected costs if we **relocate** and **demand is low**	10,000 €
Expected profits if we **remain** and **demand is high**	5,000 €
Expected costs if we **remain** and **demand is low**	0 €

Decision Trees – Build by hand

Determine the decision and unknown



Decision Trees – Build by hand

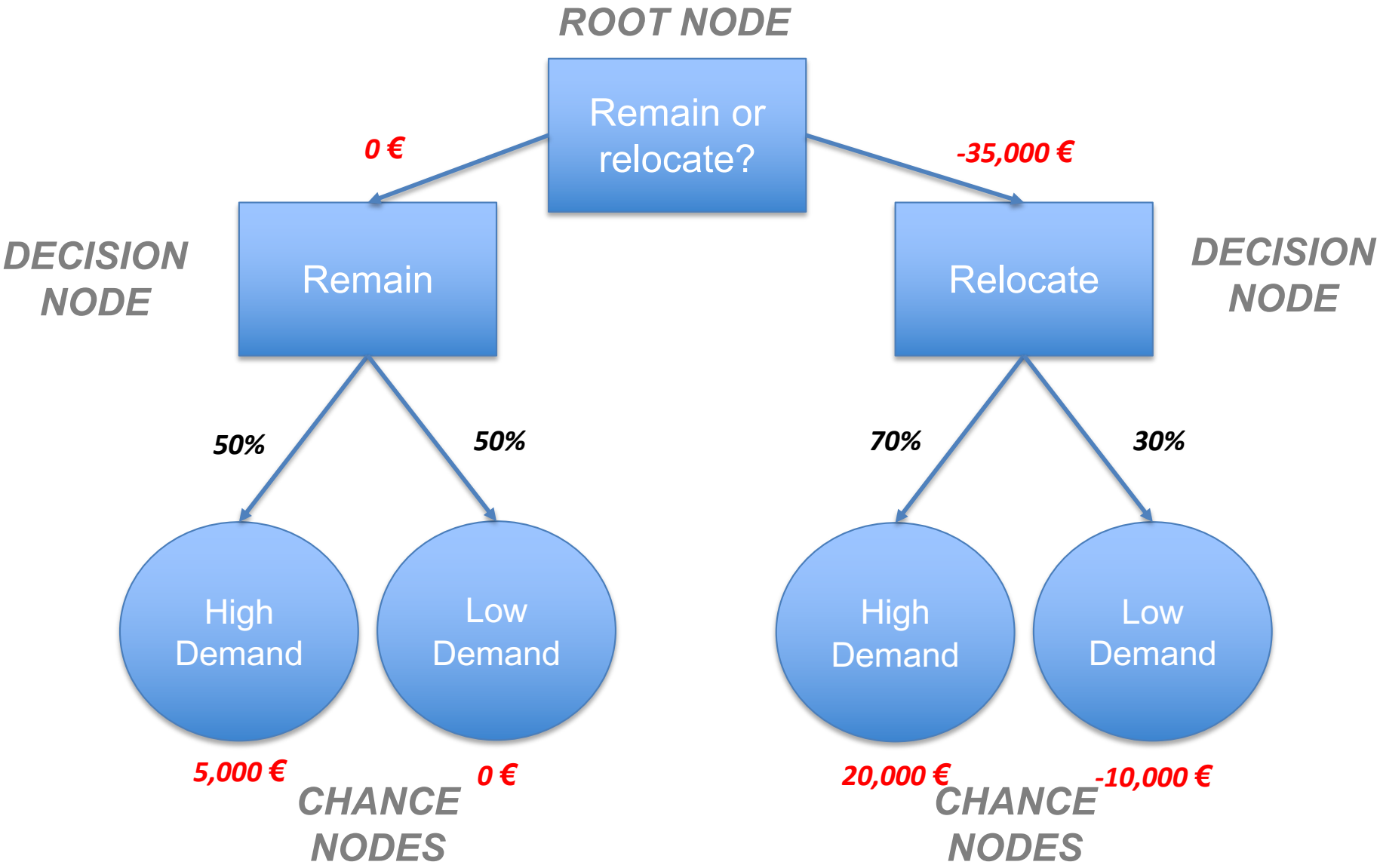
Calculate the weighted value of each decision

- Not all the branches have the same probabilities for occurring, so we need to label our probabilities.
- Obviously, the probabilities of our initial decision (remain/relocate) does not need to be labelled, only the subsequent unknowns:

Chance nodes	Probabilities
Remain AND High Demand	50%
Remain AND Low Demand	50%
Relocate AND High Demand	70%
Relocate AND Low Demand	30%

Decision Trees – Build by hand

Calculate the weighted value of each decision



Decision Trees – Build by hand

Determine the probabilities for each branch

- We need to calculate the value of each unknown for each decision, and then add them together

REMAIN	Calculation
High Demand	5,000*0,5 = 2,500 €
Low Demand	0*0 = 0 €
Weighted Value	2,500 + 0 = **2,500 €**

RELOCATE	Calculation
High Demand	20,000*0,7 = 14,000 €
Low Demand	-10,000*0,3 = -3,000 €
Weighted Value	14,000 + (-3,000) = **11,000 €**

Decision Trees – Build by hand

Calculate the Net Benefit for Each Decision

- We need to subtract the initial cost of the decision from the weighted value of each decision. The end result is the **net benefit** of each decision.

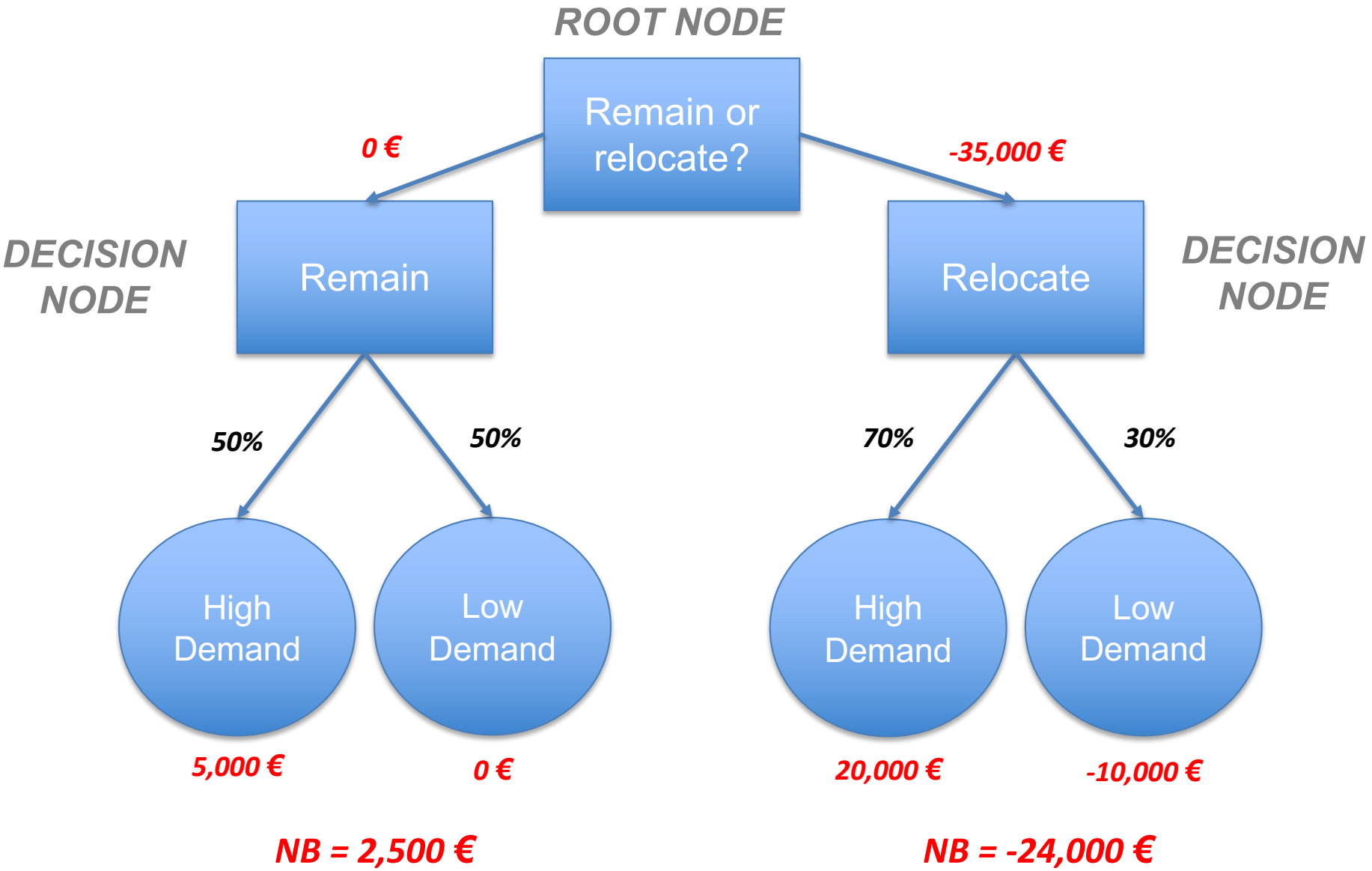
REMAIN	Calculation
Initial Cost	0 €
Weighted Value	**2,500 €**
NET BENEFIT	2,500 - 0 = **2,500 €**

RELOCATE	Calculation
Initial Cost	35,000 €
Weighted Value	11,000 €
NET BENEFIT	11,000 – 35,000 = **-24,000 €**

- According to the given probabilities the best is to remain. But if those probabilities are changed we might consider to relocate

Decision Trees – Build by hand

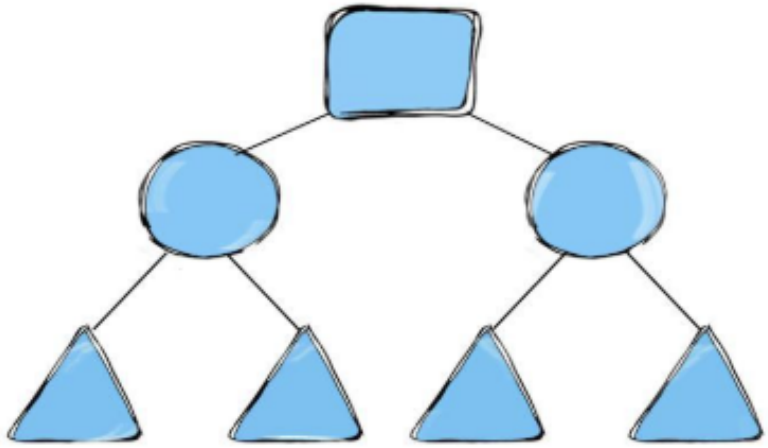
Calculate the Net Benefit for Each Decision



Decision Trees – Build by hand

Recap

- Now we have an intuition on how DTs are functioning and how they are used for taking complex decisions
- The following slides will dive into the concept of DTs as a machine learning tool



BTS | Barcelona Technology School

MBDS 2020-2021

Decision Trees

ML

Decision Trees – ML

Not so naïve

- DTs are a highly versatile ML algorithm that can:
  - Perform classification
  - Perform regression
  - Multioutput tasks

- Even if they seem naïve, they are **extremely powerful,** capable of fitting (and most of the times overfitting) complex datasets.

- DTs are the building blocks of **Random Forests** (RFs)**,** which are one of the most powerful ML algorithms used across the industry. Understanding DTs is essential to understand RFs.

Decision Trees – ML

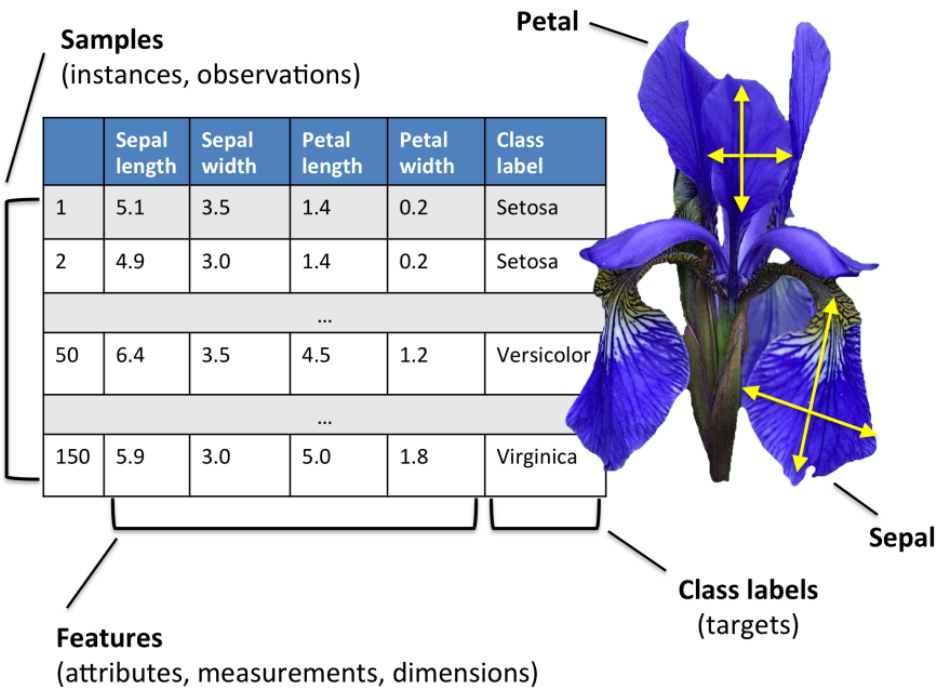
Our beloved scikit-learn

- The following slides will discuss how to **train**, **visualize** and **make predictions** with DTs
- We will see the **CART training algorithm** that sklearn uses, which is one of many implementations of DTs
- We will see how to **regularize** DTs
- We will learn how to use them for **regression**
- We will see some of its **limitations** and how RFs come in handy to overcome those

Decision Trees – ML

The Iris Dataset

- We will use the Iris dataset as well (like for SVM)
- Remember that the Iris Dataset, which comes packaged in sklearn:
  - The dataset consists of 3 different types of irises (Setosa, Versicolour and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray
  - The rows are the samples
  - The columns are:
    - Sepal Length
    - Sepal Width
    - Petal Length
    - Petal Width



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Decision Trees – ML

Training and visualization

- To train and visualize a DT in sklearn is deadly simple:

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

# Load the Iris dataset
iris = load_iris()

# Load only the petal lenght and width
X = iris.data[:, 2:] # petal length and width

# Load the Iris target
y = iris.target

# Call the DT using a maximum depth of 2 (the decision nodes)
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)

# Fit the DT
tree_clf.fit(X, y)
```

*Code to reproduce the graphs can be seen at **UUID - #S6AC1***

Decision Trees – ML

Training and visualization

- The console returns our call with several default options:

```
DecisionTreeClassifier(ccp_alpha=0.0,
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='deprecated',
random_state=42, splitter='best')
```

*Code to reproduce the graphs can be seen at **UUID - #S6AC1***

Decision Trees – ML

Training and visualization

- To visualize DTs in sklearn we can use Graphviz. You can install it from here.
- To convert the .dot file to a .png, use the Graphviz CLI:
  - `$ dot -Tpng iris_tree.dot -o iris_tree.png`

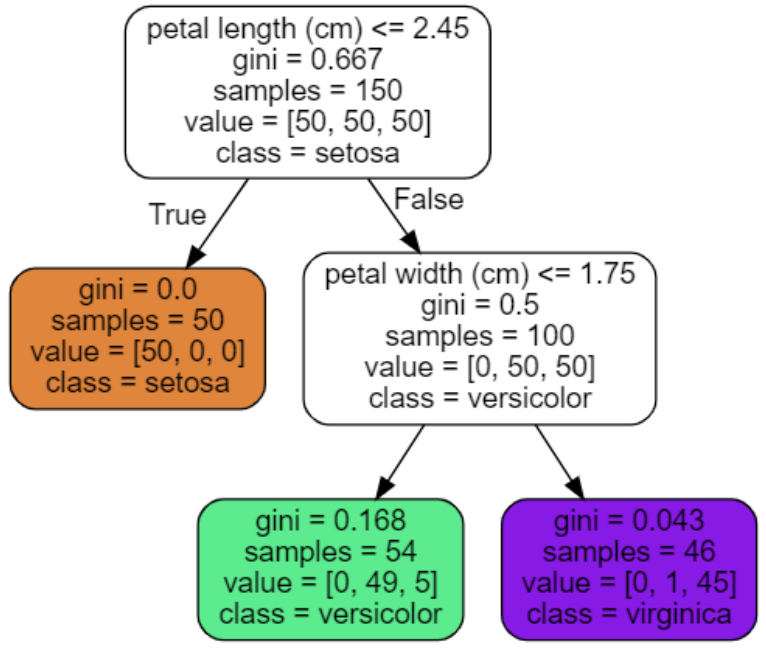
```
# Visualize the Decision Tree (you need to have graphviz installed)
from graphviz import Source
from sklearn.tree import export_graphviz

# Call the export with Graphviz
export_graphviz(
        tree_clf,
        out_file=os.path.join(IMAGES_PATH, "iris_tree.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

# Load the file in the notebook
Source.from_file(os.path.join(IMAGES_PATH, "iris_tree.dot"))
```

*Code to reproduce the graphs can be seen at **UUID - #S6AC1***

Decision Trees – ML

Training and visualization

- The resulting tree contains the branches automatically outlined
- There is a preset **root node** of petal length <= 2.45
- Then there is a **leaf** consisting on the setosa class, and one more **decision node** with another petal width
- But, *how are those decision nodes automatically outlined for us* and, *what is the Gini number?*

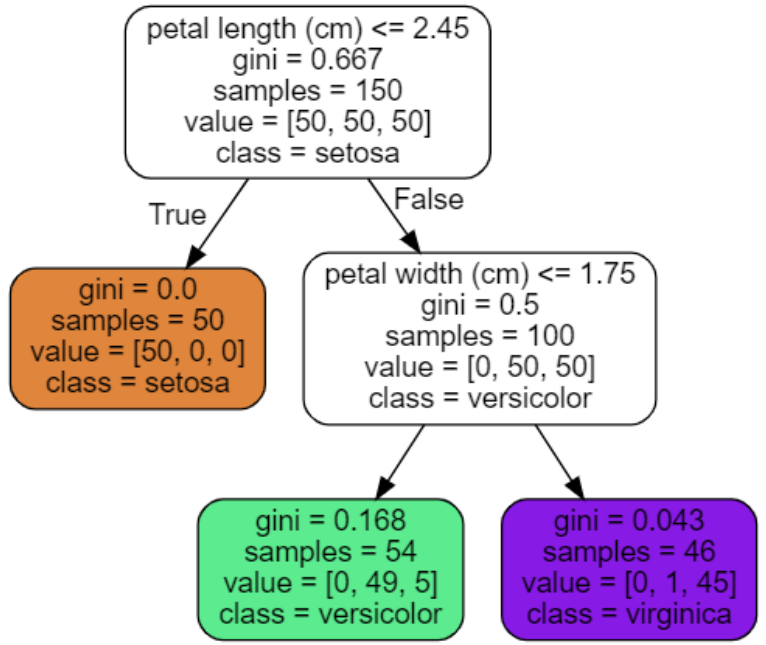


*Code to reproduce the graphs can be seen at **UUID - #S6AC1***

Decision Trees – ML

Prediction example

- Let's assume that we have found an Iris flower and we want to classify it:
  1. We start at the **root node** (depth 0). If the leave is less than 2.45, we move to **left child node** (depth 1, left). This is a **leaf** and has no further children nodes (no more questions asked). Your Iris is a **Setosa**.
  2. We start at the **root node** (depth 0). If the leave is more than 2.45, we move to **right child node** (depth 1, left). This is another **decision node**, which asks another question:
     1. If the petal width is smaller than 1.75, then we have an Iris Versicolor
     2. If the petal width is bigger than 1.75, then we have an Iris Virginica



*Code to reproduce the graphs can be seen at **UUID - #S6AC1***

Decision Trees – ML

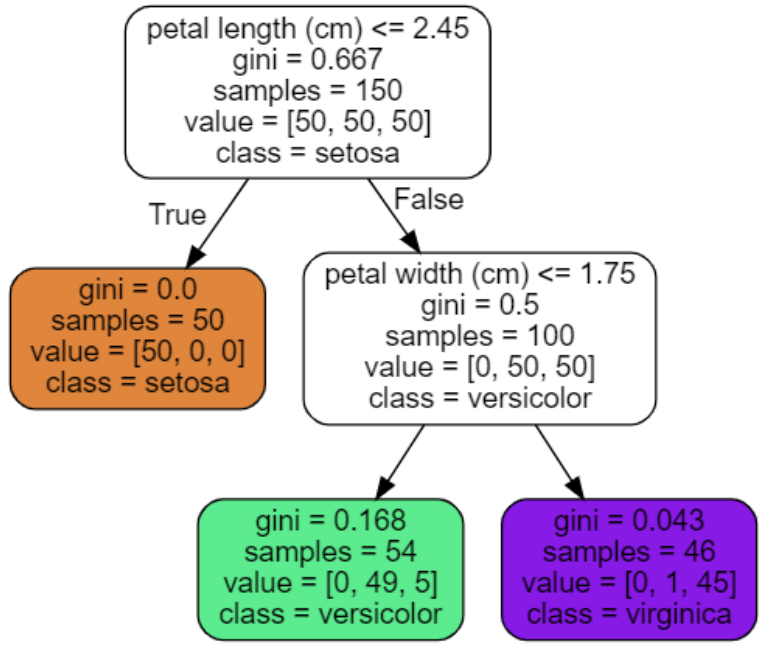
Prediction example

- You might have noticed that we did not use scaling prior to loading the data onto the DT. Actually, DTs require minimal data preparation, as they do not require feature scaling or centering at all.

Decision Trees – ML

Node attributes

- **Node samples**: how many training instances it applies to.
- **Node value**: how many training instances of each class this node applies to, i.e.
  - The bottom right leaf applies to 0 Iris Setosa, 1 Versicolor and 45 Virginica
- **Gini**: measures the *impurity* of the node

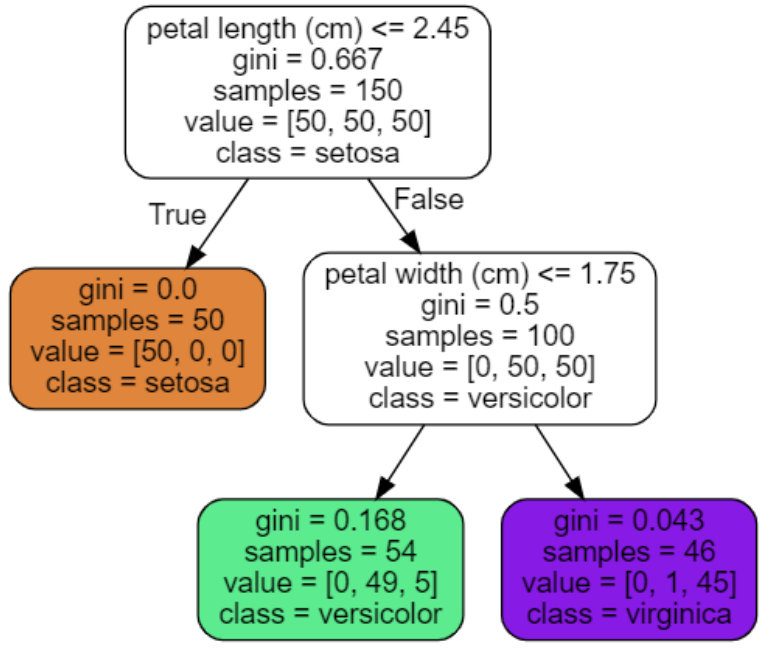


*Code to reproduce the graphs can be seen at **UUID - #S6AC1***

Decision Trees – ML

Gini Impurity

- A node is **pure** when all the training instances it applies to belong to the same class. The leaf for Setosa only applies to Setosa, thus the leaf is pure and gini=0



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

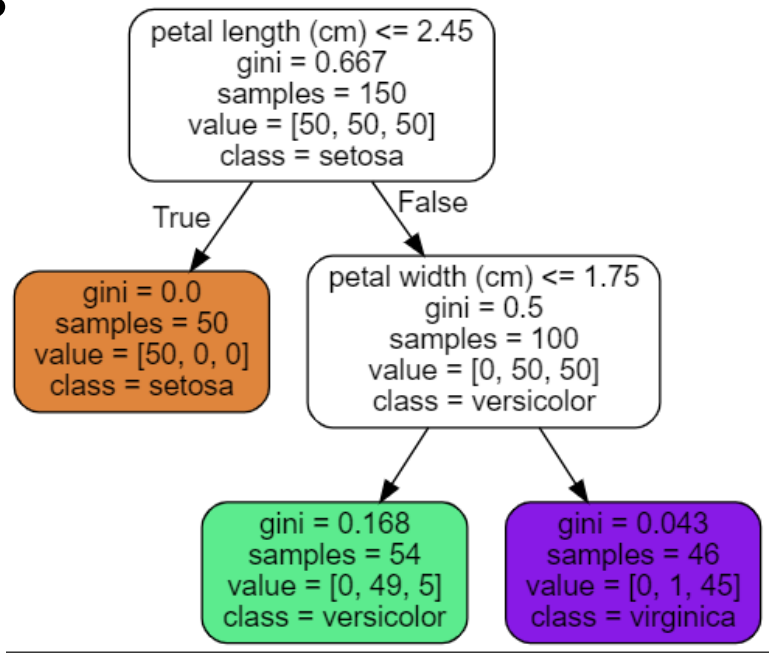
$p_{i,k}$ is the ratio (probability) of class k instances among the training instances in the i^{th} node

*Code to reproduce the graphs can be seen at **UUID - #S6AC1***

Decision Trees – ML

Gini Impurity

- The **Versicolor leaf is impure**, let's calculate the Gini:
- $G_{versicolor} = 1 - \left(\frac{0}{54}\right)^2 - \left(\frac{49}{54}\right)^2 - \left(\frac{5}{54}\right)^2 \approx 0.168$



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

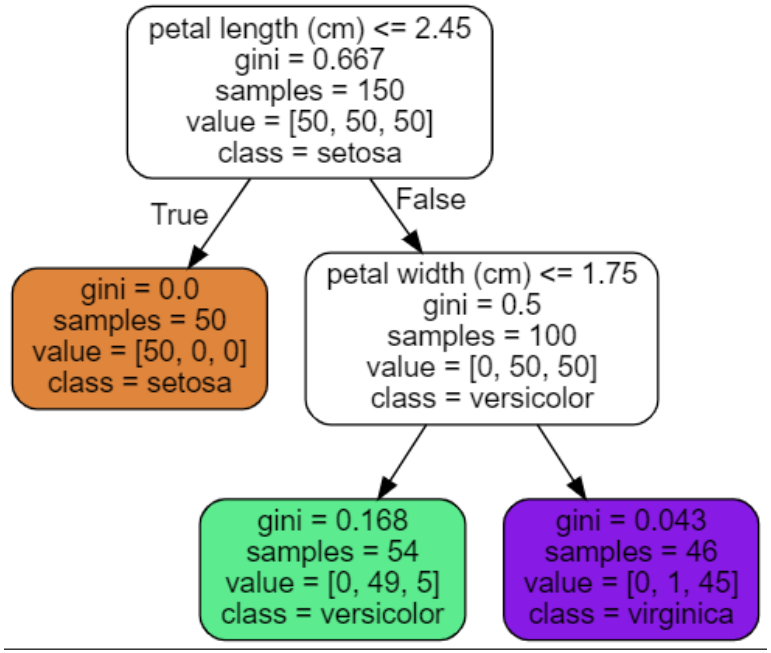
$p_{i,k}$ is the ratio (probability) of class k samples among the training samples in the i^{th} node

*Code to reproduce the graphs can be seen at **UUID - #S6AC1***

Decision Trees – ML

Gini Impurity

- Now, try to calculate the Gini for Virginica:
  - Which leaf is less impure?
  - Why?



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

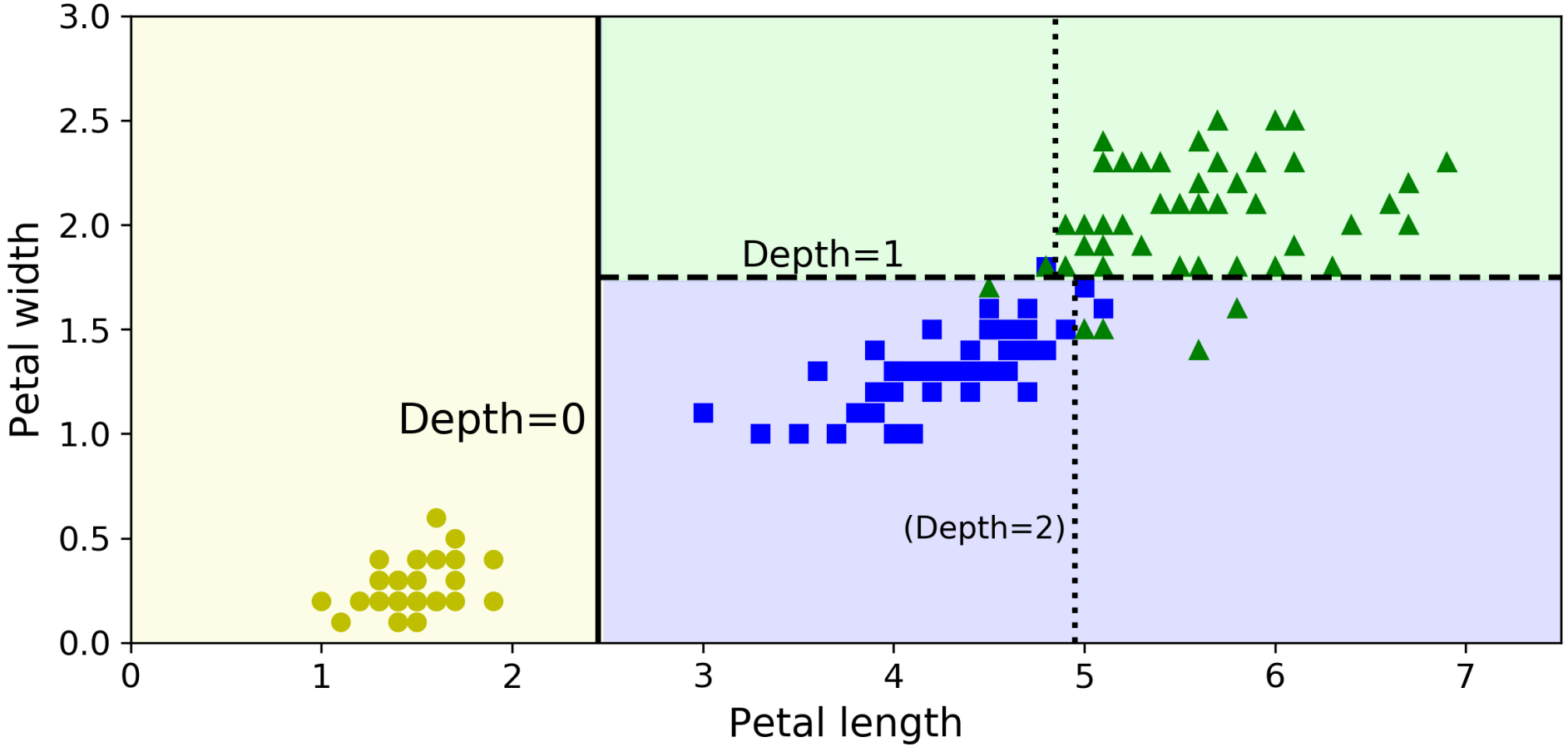
$p_{i,k}$ is the ratio (probability) of class k samples among the training samples in the i^{th} node

*Code to reproduce the graphs can be seen at **UUID - #S6AC1***

Decision Trees

Decision boundaries

- The plot below shows the decision boundaries of the DT that we have just build.

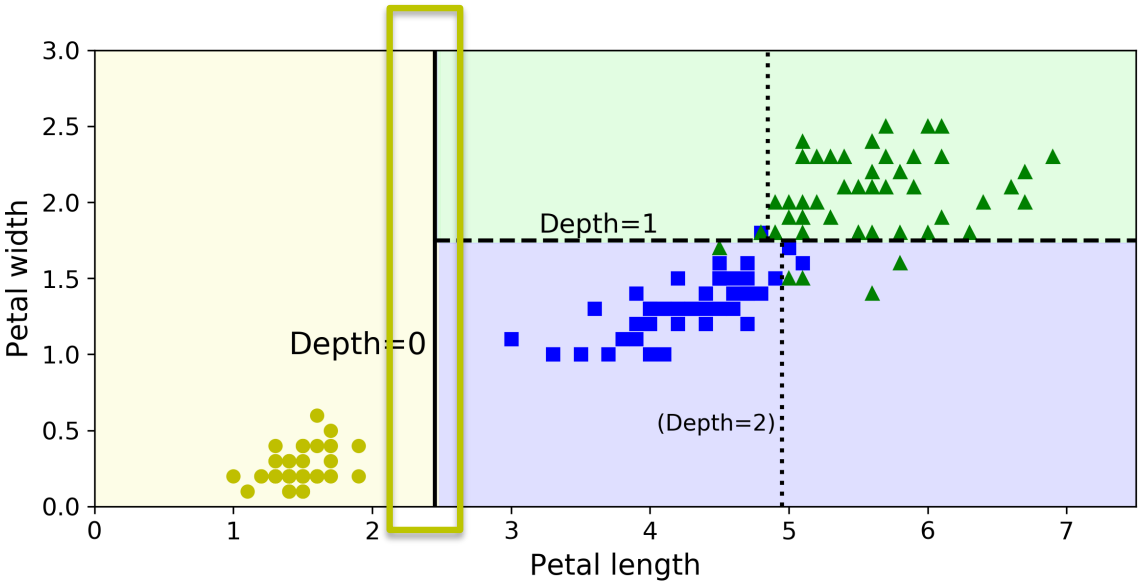


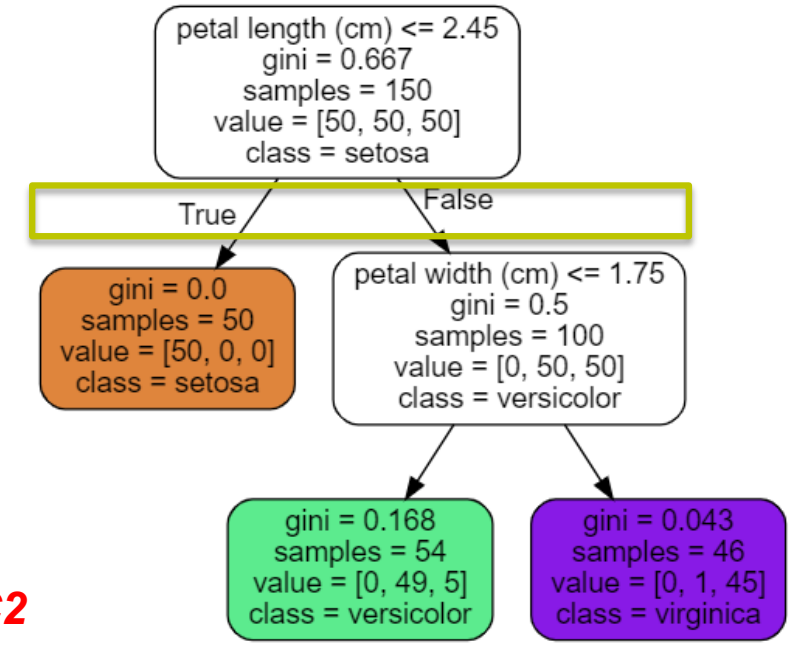
*Code to reproduce the graphs can be seen at **UUID - #S6AC2***

Decision Trees

Decision boundaries

- The plot below shows the decision boundaries of the DT that we have just build.
- The thick vertical line is the decision boundary for the root node at petal length 2.45cm
- The ***left area*** is ***pure*** and cannot be split any further, but the ***area at the right*** is ***impure*** and we can split it further



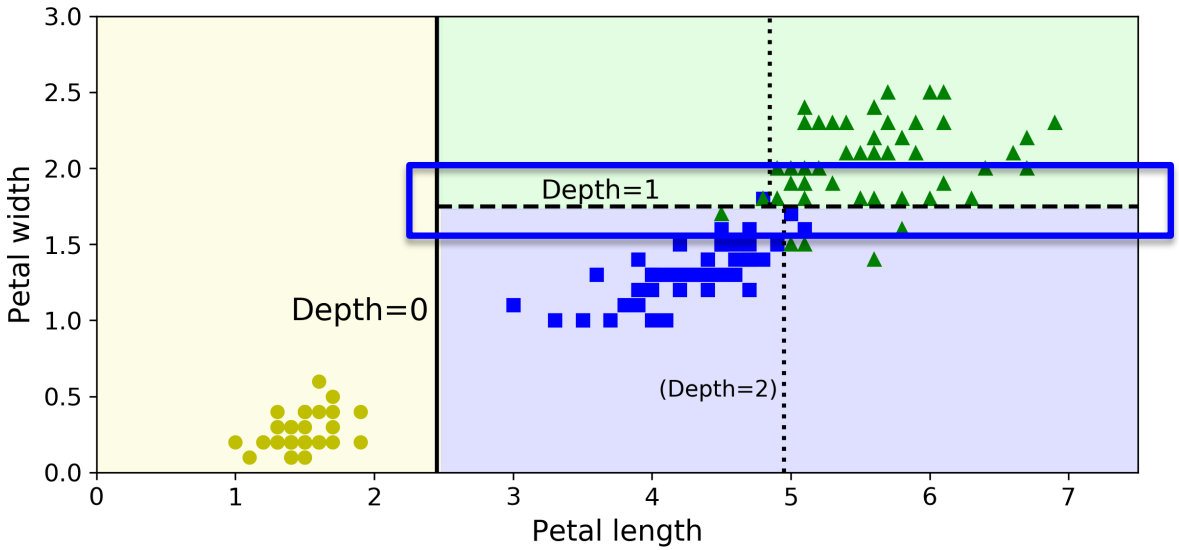


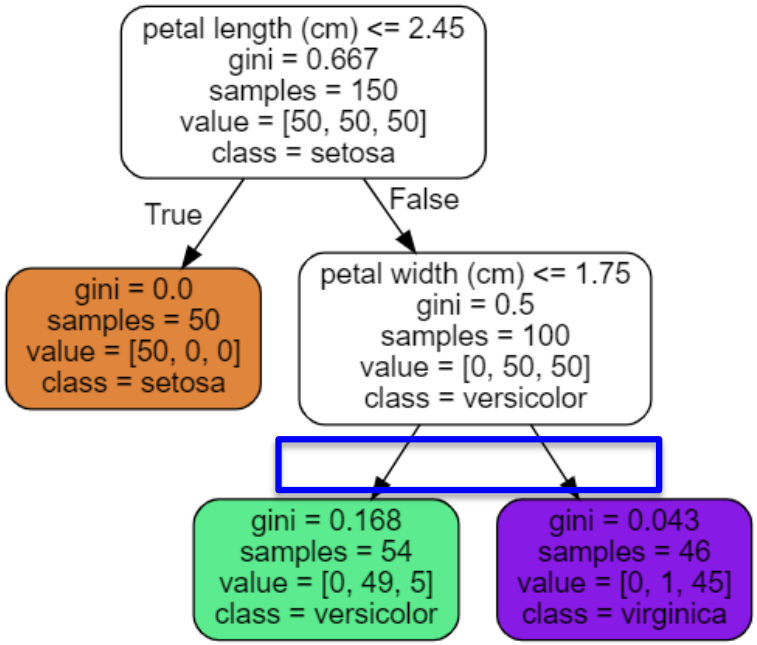
*Code to reproduce the graphs can be seen at **UUID - #S6AC2***

Decision Trees

Decision boundaries

- The dashed line is the decision boundary for the root node at petal width 1.75cm
- Because `max_depth=2`, the DT stops there.
  - If we go further adding more depth, we will add more decision boundaries (depth=2)





*Code to reproduce the graphs can be seen at **UUID - #S6AC2***

Decision Trees

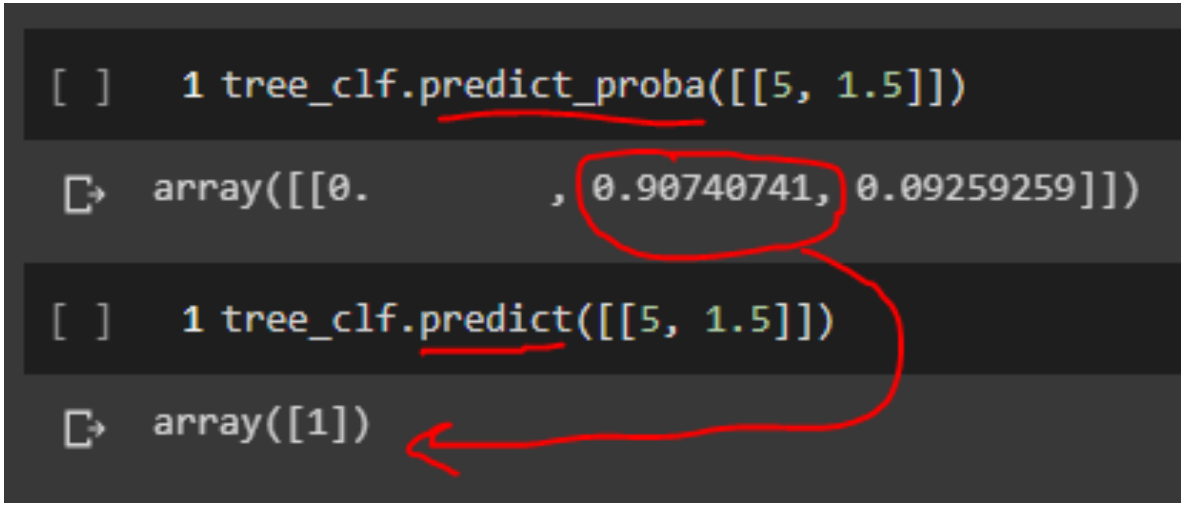
DT Interpretability

- DTs are very intuitive and results are easily interpretable
- This is extremely important for some clients, as DTs provide nice and simple classification rules that can even be applied manually
  - DTs are usually called **White Box models** due to their high interpretability
- Neural Networks and Random Forests are often criticized because they behave in a **Black box** manner, making it very difficult to interpret the algorithm
  - However, there is a huge movement now pushing for AI interpretability and explainability, with some very interesting approaches
  - There are certain ways of actually interpreting a neural network

Decision Trees

Estimating class probabilities

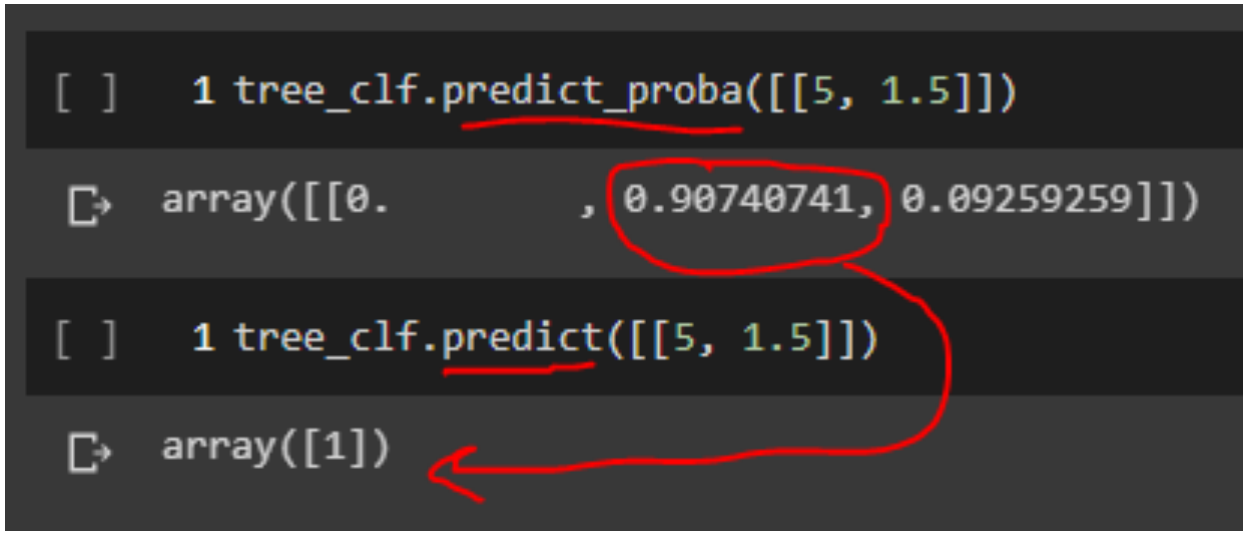
- DTs can calculate the probability that a sample belongs to a class k
- This is done by travelling through the leaf node for the instance, and returning the ratio of training instances of class k in the node:
  - Example: for the depth two left node:
    - Iris setosa – 0/54 – 0%
    - Iris versicolor – 49/54 – 90,7%
    - Iris virginica – 5/54 – 9,3%



*Code to reproduce the graphs can be seen at **UUID - #S6AC3***

Decision Trees

Estimating class probabilities



- Notice that the estimated probabilities are identical in the bottom right rectangle. All the probabilities are the same, even for those at the decision boundary.

*Code to reproduce the graphs can be seen at **UUID - #S6AC3***

Decision Trees

CART Algorithm

- Some of the algorithms available for building up DTs are:
  - **Iterative Dichotomiser 3 (ID3):** the grandfather of decision tree algorithms developed in 1986 by Ross Quinlan. The ID3 works well for classification but it is challenging to use for regression and highly susceptible to overfitting.
  - **C4.5:** invented in 1993 by Ross Quinlan, it is the successor to ID3. It improves a lot in terms of overfitting by using pruning and other techniques.
  - **CART:** stands for Classification And Regression Trees and is one of the most widely used type of DT algorithm. Developed in 1984, it does not have the capabilities of building non-binary trees such as ID3 and C4.5. **It is the algorithm used by scikit-learn**

Decision Trees

CART Algorithm

- The idea of CART algorithm is quite simple:
  - First it **splits the training set** in **two subsets** using a **single feature k** and a **threshold t_k**
    - Petal length ≤ 2.45cm
- But, how does it choose k and t_k ?
  - By searching the pair (k, t_k) that produces the purest subsets (weighted by size)
  - The cost function for training a DT for classification is as follows:

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

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 Trees

CART Algorithm

- Once the dataset is split in two, proceeds in the similar manner with the sub-subsets, and so on…
- When will it stop?
  - When it **reaches the defined maximum depth** defined by the `max_depth` hyperparameter
  - When it cannot find a split that will **reduce impurity**.

Decision Trees

Finding an optimal solution?

- The CART algorithm is a ***greedy*** *one* meaning that searches an optimum split at the top level, repeating the process at each level
  - It does not check what will happen several levels down, like a **dynamic algorithm** would do.

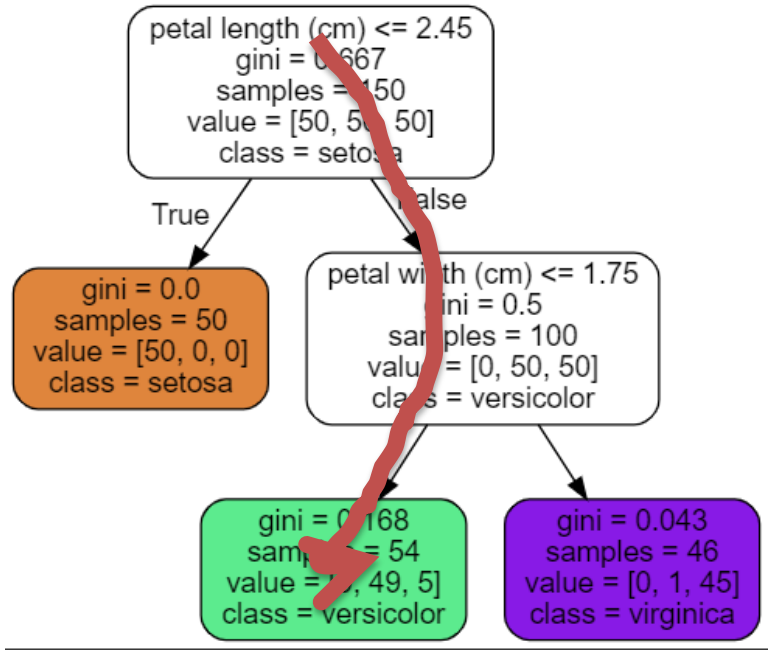
- As a greedy algorithm, CART will produce a good enough solution but not the optimal one

- Part of this is that finding an optimal tree is an **NP-Complete problem**. Having a O-notation of *O(exp(m)).* This makes the problem intractable even for small datasets, so “good enough” solutions are fine.

Decision Trees

Computational complexity - Prediction

- Even if finding an optimal tree is nearly impossible (unless we have a quantum computer), **predicting on DTs is extremely fast**
- When we make a prediction, **we go from root to leaf**. Because DTs are **generally balanced**, traversing through the tree is:
  - $O\left(log_2(m)\right) = \frac{\log(m)}{\log(2)}$
  - So, predictions are always the $log_2(m)$ **which is not affected by the number of features**
    - Thus, predictions are **fast**, even with **really big datasets**



m, number of samples

Decision Trees

Computational complexity - Training

- Unless `max_features` is set, the CART algorithm compares all the features, on all samples at each node
  - The computational complexity of the training is as follows:
  - ***O(n × m log(m))***

In order to **speed up training** on small datasets (less than few thousand instances), older version of Scikit-Learn could presort the data (using `presort=True`). However, this slows down training considerably for bigger datasets!!

Checkout this Kaggle example

m, number of samples
n, number of features

Decision Trees

Gini impurity or Entropy?

- One of the key components to split the decision nodes is ***Gini impurity*** measure, but in the `criterion` hyperparameter ***entropy*** can also be used
- Entropy is also a measure of impurity and disorder, i.e. entropy=0 when a node contains instances of only one class
- As an example, we can calculate Entropy for depth-2 left node
  - $H_{versicolor} = -\frac{49}{54}\log\left(\frac{49}{54}\right) - \frac{5}{54}\log\left(\frac{5}{54}\right) \approx 0.31$

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

Decision Trees

Gini impurity or Entropy?

- When to use one or the other? It really does not matter much… they lead to similar trees
- When the trees differ:
  - Gini – tends to isolate the most frequent class in its own branch
  - Entropy – tends to create more balanced trees

Gini tends to train faster

Decision Trees

Regularization Hyperparameters

- DTs are very sensitive to overfit!
  - They make very few assumptions about training data, e.g., linear models apply to linear data
  - If not constrained, the tree adapts to the data, with the possibilities to overfit
- Nonparametric vs parametric

Concept	Nonparametric model	Parametric model
Definition	The number of parameters is not determined prior to training	The number of parameters is determined prior to training
Example	Decision Trees	Linear Regression
Limited degrees of freedom	No	Yes
Tendency	Overfit	Underfit

Decision Trees

Regularization Hyperparameters

- DTs are very sensitive to overfit!
  - They make very few assumptions about training data, e.g. linear models apply to linear data
  - If not constrained, the tree adapts to the data, with the possibilities to overfit
- Nonparametric vs parametric

Concept	Nonparametric model	Parametric model
Definition	The number of parameters is determined prior to training	The number of parameters is determined prior to training
Example	Decision Trees	Linear Regression
Limited degrees of freedom	No	Yes
Tendency	Overfit	Underfit

We need to restrict DTs freedom during training!!

Decision Trees

Regularization Hyperparameters

Hyperparameter	Regularization action	Comments
`max_depth`	Controls the maximum allowed depth of the tree	The default is None, which means that the tree is not constrained and can have unlimited depth. Reduce it to regularize the model.
`min_samples_split`	A minimum number of samples a node must have before it can be split	Increase it to regularize the model.
`min_samples_leaf`	A minimum number of samples a leaf node must have	Increase it to regularize the model.
`min_weight_fraction_leaf`	Same as above but expressed as a fraction of the total number of weighted instances	Increase it to regularize the model.
`max_leaf_nodes`	Maximum number of leaf nodes	Reduce it to regularize the model.
`max_features`	Maximum number of features evaluated for splitting each node	Reduce it to regularize the model.

Decision Trees

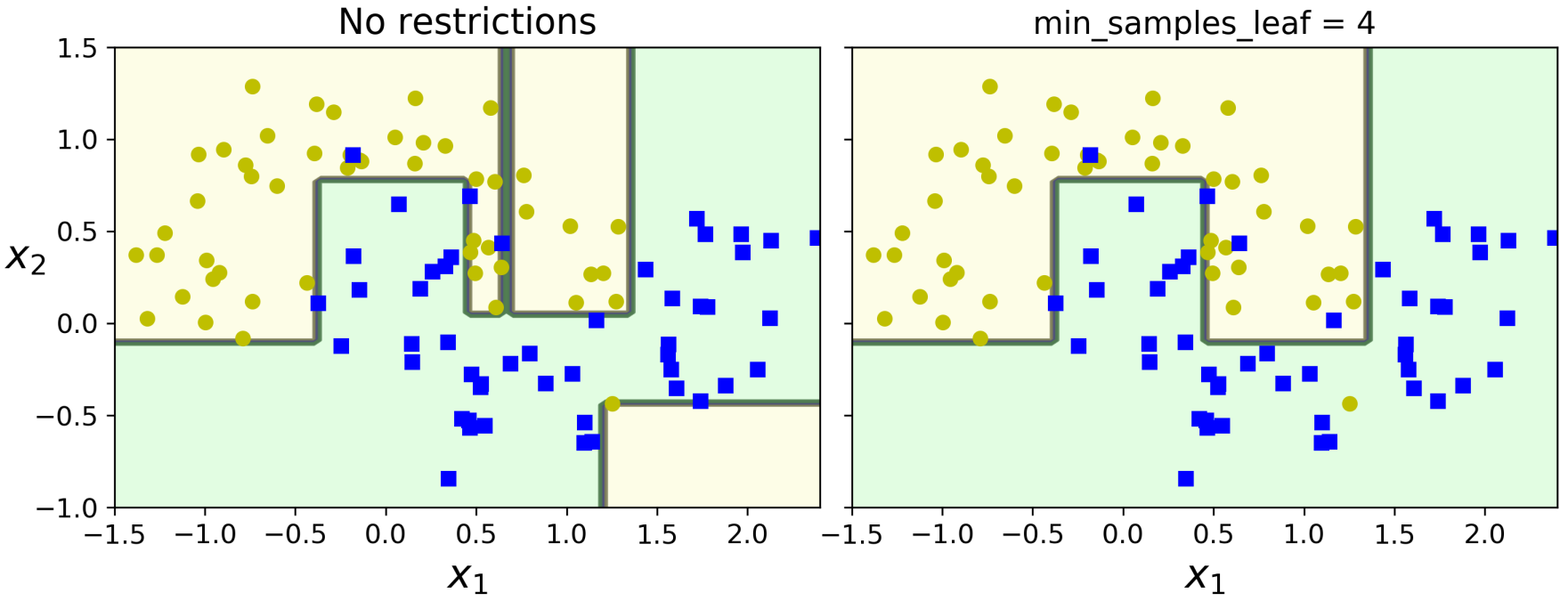
Another type of “regularization” - Pruning trees

- Another approach is to train the DT without restrictions and then ***prune*** (delete) all the unnecessary nodes
- The pruning is decided by checking if the purity improvement of deleting a node and its children is *statistically significant*
- If the p-value is 5%, the node is considered unnecessary, and the children deleted
- Pruning goes on and on until all the unnecessary nodes have been deleted (pruned)

Decision Trees

Regularization Hyperparameters

- The plot below depicts two DTs trained on the moons dataset
- The left one is overfitted and will poorly generalize
- The right one has the capped and will generalize better



*Code to reproduce the graphs can be seen at **UUID - #S6AC4***

Decision Trees

Regression

- DTs can also be used for regression!

```
1 # Quadratic training set + noise
2 np.random.seed(42)
3 m = 200
4 X = np.random.rand(m, 1)
5 y = 4 * (X - 0.5) ** 2
6 y = y + np.random.randn(m, 1) / 10
```

```
1 from sklearn.tree import DecisionTreeRegressor
2
3 tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
4 tree_reg.fit(X, y)
```

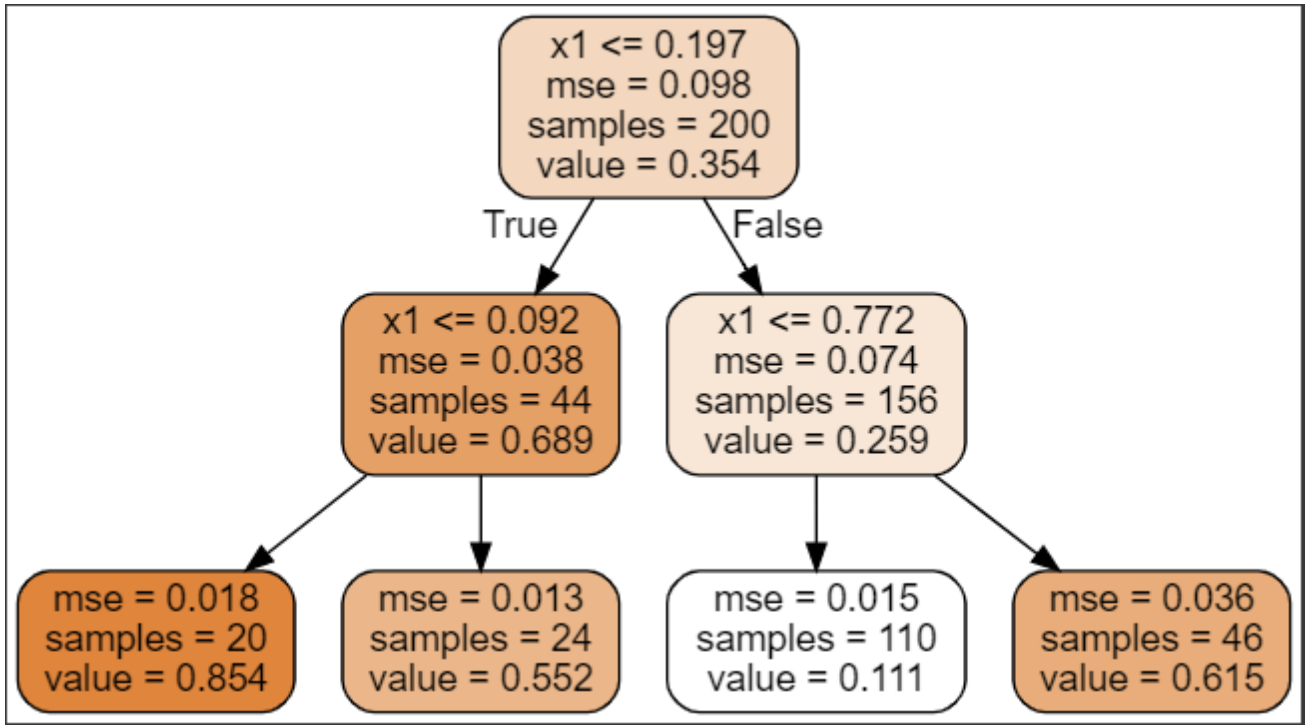
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=42, splitter='best')
```

*Code to reproduce the graphs can be seen at **UUID - #S6AC4***

Decision Trees

Regression

- The DT for regression is similar to classification, but here instead we predict a value instead of a class
- The predictions are always the average target value for the associated training instances to the leaf node

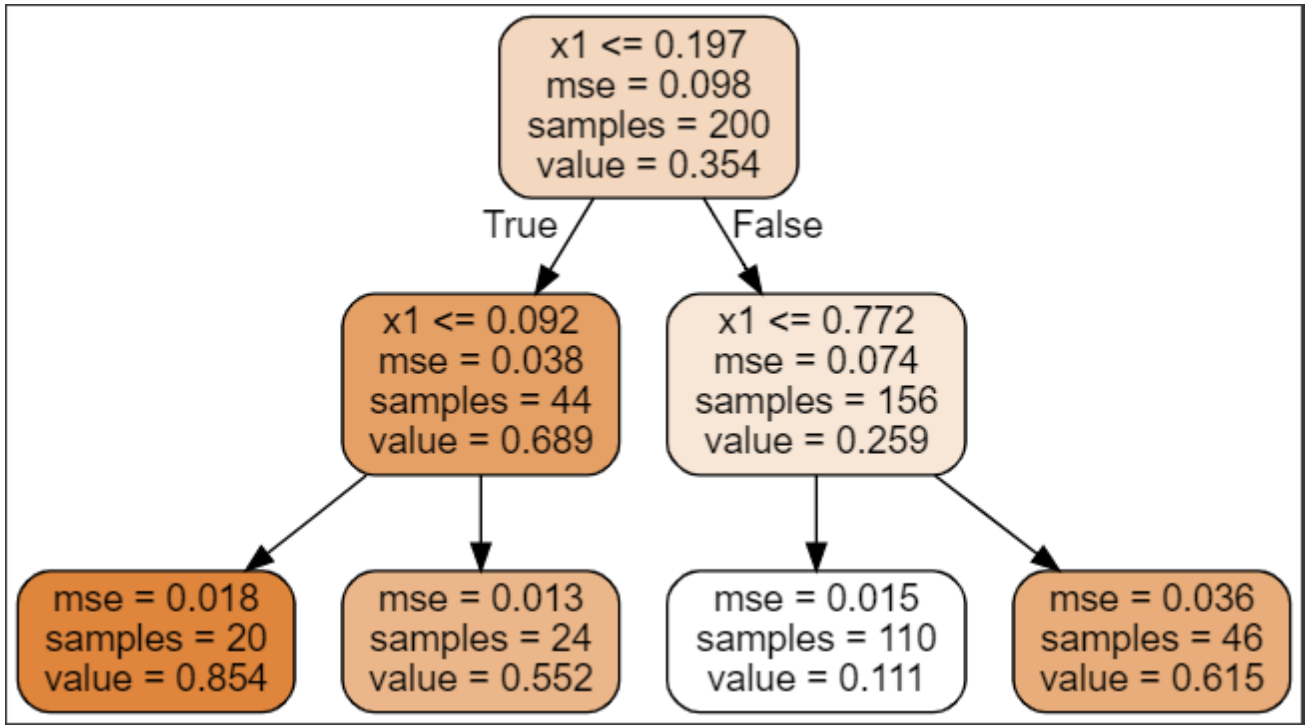


*Code to reproduce the graphs can be seen at **UUID - #S6AC5***

Decision Trees

Regression

- If we want to predict the value for x=0.6, we will reach the leaf node that predicts value=0.111
- Again, this prediction is the average value over 110 samples, having a Mean Squared Error (MSE) of 0.015

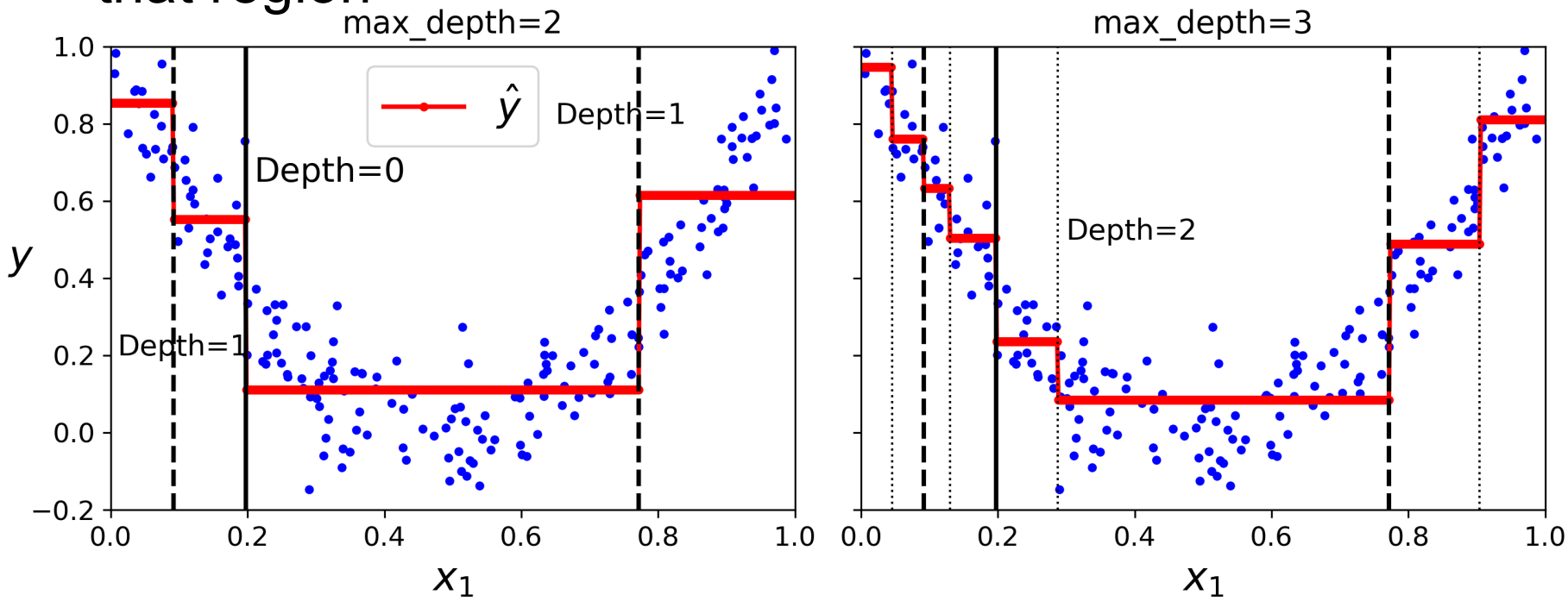


*Code to reproduce the graphs can be seen at **UUID - #S6AC4***

Decision Trees

Regression

- Below the noisy quadratic regression is represented with two types of regularization
- Notice how always the predicted value for each region is always the **average target values of the instances** in that region



*Code to reproduce the graphs can be seen at **UUID - #S6AC5***

Decision Trees

Regression – CART cost function

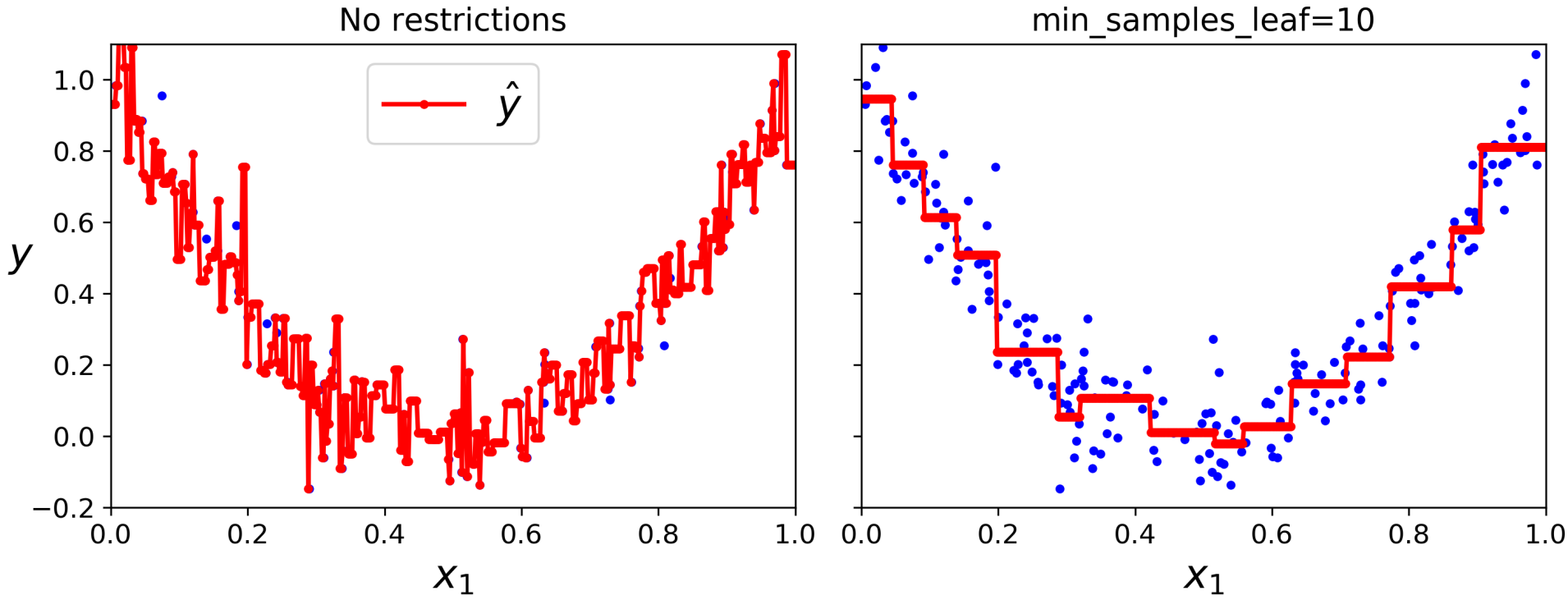
- The CART works in similar manner, but instead of splitting the training set for minimizing impurity (either through Gini or entropy)…
  - Now tries to minimize the MSE
- In the `DecisionTreeRegressor` class there are a couple of other criterions, i.e. “friedman_mse” and “mae”
  - **MSE (default)** – minimizes L2 loss using the mean of each terminal node
  - **Friedman MSE** – it uses MSE with an added improvement score for potential splits
  - **MAE (Mean Absolute Error)** – minimizes the L1 loss using the median of each terminal node

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

Decision Trees

Regression regularization

- Regression is also susceptible to overfit
- The model on the left extremely overfits the dataset
- The model on the right generalizes better after regularization

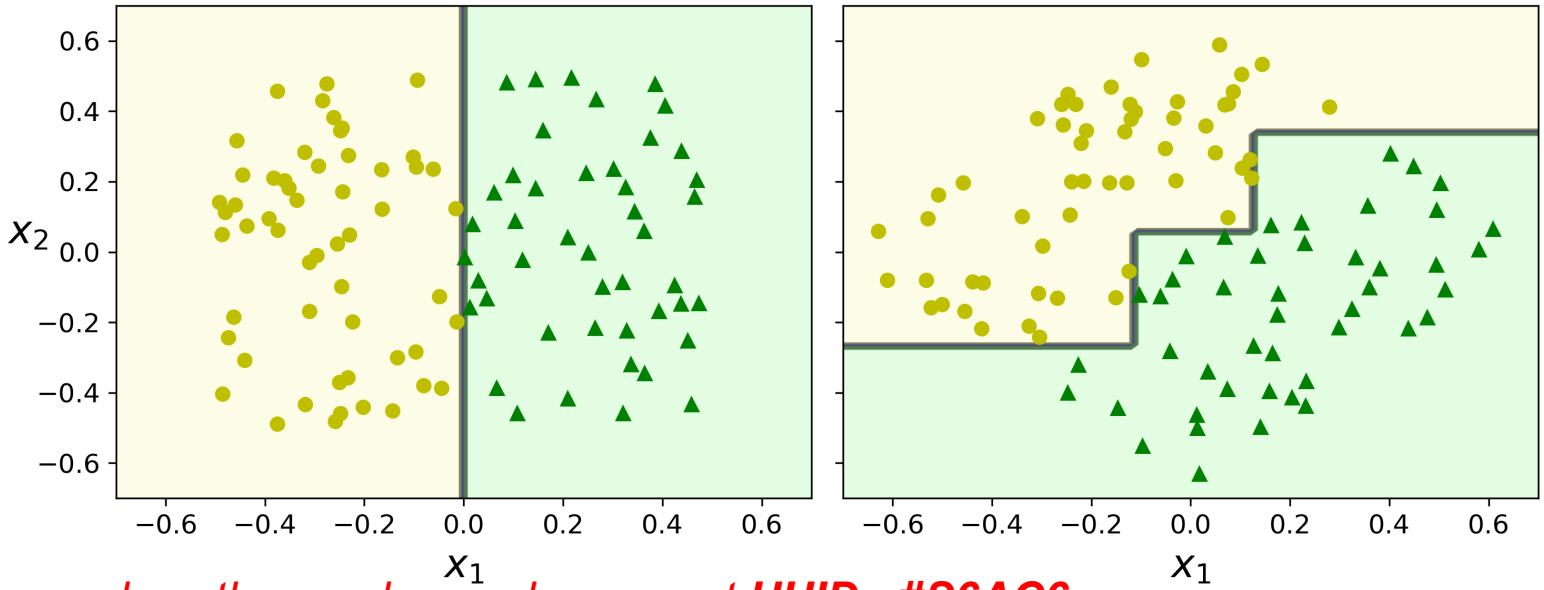


*Code to reproduce the graphs can be seen at **UUID - #S6AC5***

Decision Trees

Instability – orthogonal decision boundaries

- DTs have some clear limitations:
  - They love orthogonal decision boundaries (all splits are always perpendicular to an axis)
  - They are very sensitive to variations in the training data
- Below a dataset that is easily separable, when rotated 45 degrees, the decision boundary is too crooked, when it could be linear (but is not perpendicular to an axis)
  - So, this might not generalize well

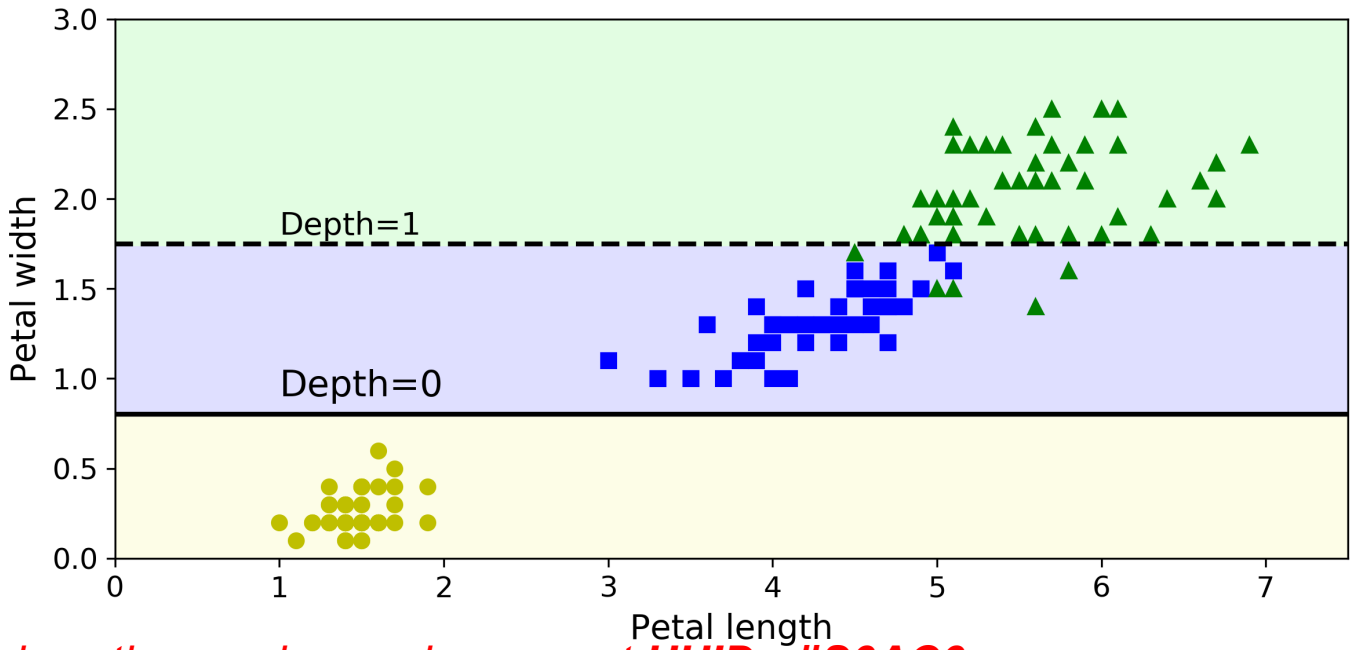


*Code to reproduce the graphs can be seen at **UUID - #S6AC6***

Decision Trees

Instability – sensitive to dataset variation

- If we remove a sample from the Iris dataset which has the widest Iris Versicolor (petals 4.8cm long and 1.8cm wide)…
  - After removing only one sample, we get the model below…
  - **Hint**: actually, because CART is stochastic (randomnly selects the set of features to evaluate at each node) you might get different model splits unless you set always the same `random_state` hyperparameter



*Code to reproduce the graphs can be seen at **UUID - #S6AC6***

Decision Trees

Sneak peak to Random Forest and Ensemble Methods

Random Forests can limit DTs model instability
by averaging predictions over many trees

BTS | Barcelona Technology School

MBDS 2020-2021

Decision Trees

Summary and Hyperparameters

Decision Trees

Tips on usage

- Hyperparameters have been already discussed above
- It is good to perform some sort of dimensionality reduction (PCA) to give the tree a better chance to find discriminative features
- Start by max_depth=3 to get a feel of how data is being split, continue from there
- For regression, use min_samples_leaf to guarantee that each leaf node has the minimum size, avoiding low-variance, over-fit leaf nodes in regression problems.
- Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant.

Decision Trees

In class exercises

Decision Trees

In class exercises

- What is the approximate depth of a Decision Tree trained (without restrictions) on a training set with 1 million instances?

Decision Trees

In class exercises

- Is a node's Gini impurity generally lower or greater than its parent's?

Decision Trees

In class exercises

- What should we do with max_depth if a DT is overfitting?

Decision Trees

In class exercises

- Should I try to scale the features before passing it to the DT? What about dimensionality reduction?

Decision Trees

In class exercises

Code exercise

UUID #S6AC7

BTS | Barcelona Technology School

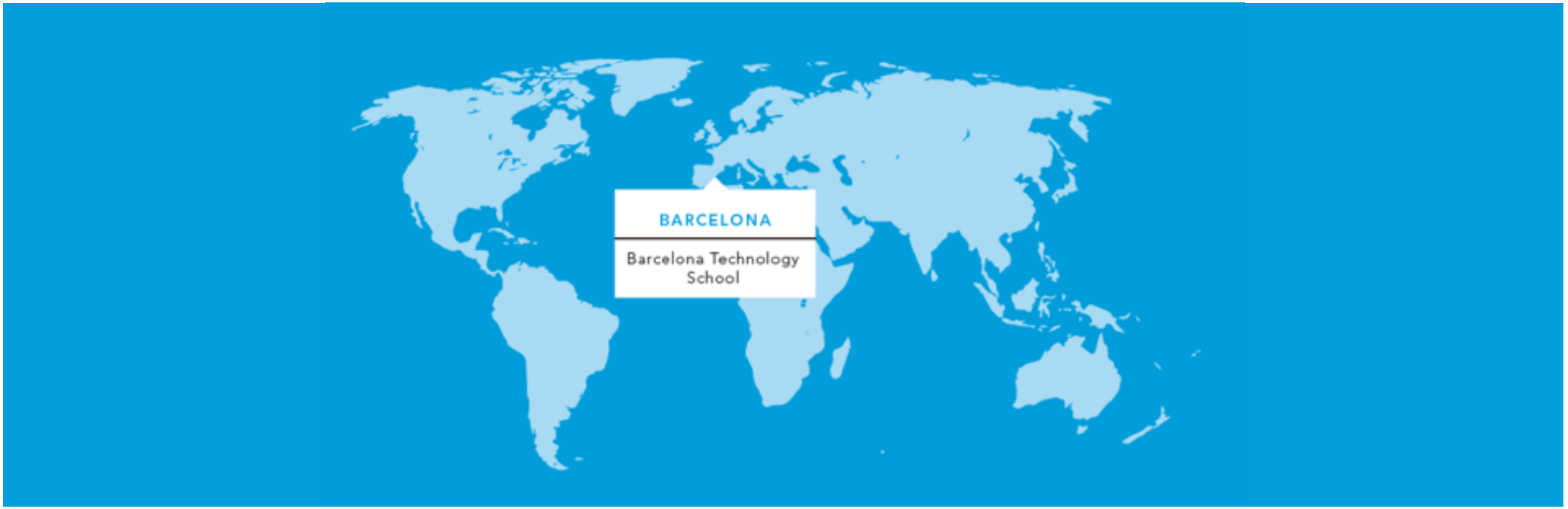
MBDS 2020-2021

RECAP

Resources

Important resources

- Lex Friedman Series (MIT)
- Sklearn docs
- Aurelien Geron, “Hands-on machine learning with scikit-learn, Keras & Tensorflow”
- Chris Smith, “Decision Trees and Random Forests: A Visual Introduction for Beginners”


BARCELONA
Barcelona Technology School