Physics 361 - Machine Learning in Physics

Lecture 7 – Decision Trees, Random Forrests, Gradient Boosted Trees

Feb. 11th 2025

Moritz Münchmeyer (with slides from Gary Shiu)

Decision trees

1. Motivation

Power of decision tree methods

- We will spend one lecture discussing an almost ancient class of machine learning techniques: decision tree methods. Why are we bothering with that in 2025?

- **Decision trees methods are still the state-of-the-art for many tabular data applications.**
    - An advanced version, Gradient boosted decision trees (GBDTs) are the current state of the art on tabular data.
    - They are used in many Kaggle competitions and are the go-to model for many data scientists, as they tend to get better performance than neural networks while being easier and faster to train.
    - Neural networks, on the other hand, are the state of the art in many other tasks, such as image classification, natural language processing, and speech recognition.

- https://arxiv.org/abs/2207.08815 Why do tree-based models still outperform deep learning on tabular data?

    While deep learning has enabled tremendous progress on text and image datasets, its superiority on tabular data is not clear. We contribute extensive benchmarks of standard and novel deep learning methods as well as tree-based models such as XGBoost and Random Forests, across a large number of datasets and hyperparameter combinations.… Results show that tree-based models remain state-of-the-art on medium-sized data (~ 10K samples) even without accounting for their superior speed. To understand this gap, we conduct an empirical investigation into the differing inductive biases of tree-based models and Neural Networks (NNs).

Why do tree-based models still outperform deep learning on tabular data?

Léo Grinsztajn
Soda, Inria Saclay
leo.grinsztajn@inria.fr

Edouard Oyallon
ISIR, CNRS, Sorbonne University

Gaël Varoquaux
Soda, Inria Saclay

Abstract

While deep learning has enabled tremendous progress on text and image datasets, its superiority on tabular data is not clear. We contribute extensive benchmarks of standard and novel deep learning methods as well as tree-based models such as XGBoost and Random Forests, across a large number of datasets and hyperparameter combinations. We define a standard set of 45 datasets from varied domains with clear characteristics of tabular data and a benchmarking methodology accounting for both fitting models and finding good hyperparameters. Results show that tree-based models remain state-of-the-art on medium-sized data (∼10K samples) even without accounting for their superior speed. To understand this gap, we conduct an empirical investigation into the differing inductive biases of tree-based models and Neural Networks (NNs). This leads to a series of challenges which should guide researchers aiming to build tabular-specific NNs: **1.** be robust to uninformative features, **2.** preserve the orientation of the data, and **3.** be able to easily learn irregular functions. To stimulate research on tabular architectures, we contribute a standard benchmark and raw data for baselines: every point of a 20 000 compute hours hyperparameter search for each learner.

Reasons for their success

- **Structure of Tabular Data**: Tabular data often contain a mix of categorical and numerical features. Tree-based models can inherently handle these different types of data and their interactions effectively.

- **Non-Linearity**: Tree ensembles are particularly good at capturing non-linear relationships and interactions between variables without needing to explicitly engineer these features. Deep learning models can also capture non-linearities but often require large amounts of data and complex architectures to do so effectively.

- **Efficiency with Small to Medium-Sized Datasets**: Deep learning models excel in domains with abundant data (like images, text, and audio) where they can learn complex patterns and representations. However, many tabular datasets are relatively small or medium-sized, where deep learning models might overfit or may not have enough data to adequately learn.

- **Other advantages:**
  - Interpretability
  - Speed
  - Feature Importance is easy to evaluate
  - Robust to outliers and missing data
  - Simplicity

Decision trees

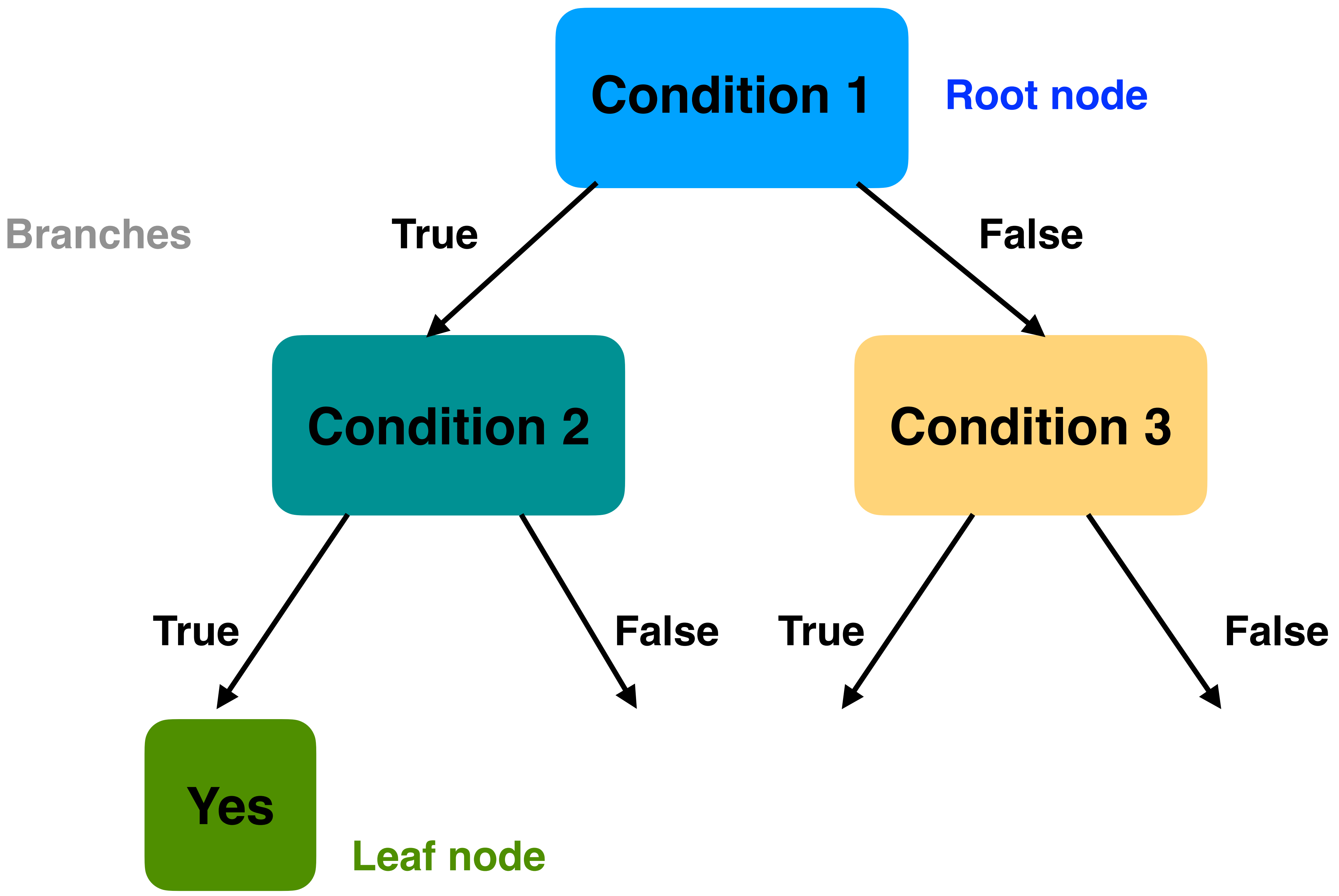
2. Introduction

Reference: “Machine Learning for Physics and Astronomy” by Viviana Acquaviva, Princeton University Press, Chapter 2.

DECISION TREES

- Work by splitting data on different values of features
- Simplest trees are binary trees
- If categorical features, the split would be on yes/no
- If numerical, the split would be on a certain value (e.g. $x > 100$ or $x < 100$)

Decision Trees



Decision Trees

- Depth of tree = maximum number of splitting conditions.
- Stop growing the tree when 1) all items on a branch have the same features (values) or 2) other stopping criterion is met.
- Usually have maximum criterion to avoid overfitting.
- At each splitting node, look for features which provide the best splitting condition. How do we quantify best?
- **Maximize “information gain”** or **maximize decrease in impurity**. (defined more precisely later)

EXAMPLE: THIS 2-FEATURE DATA SET.

HOW SHOULD WE SPLIT?

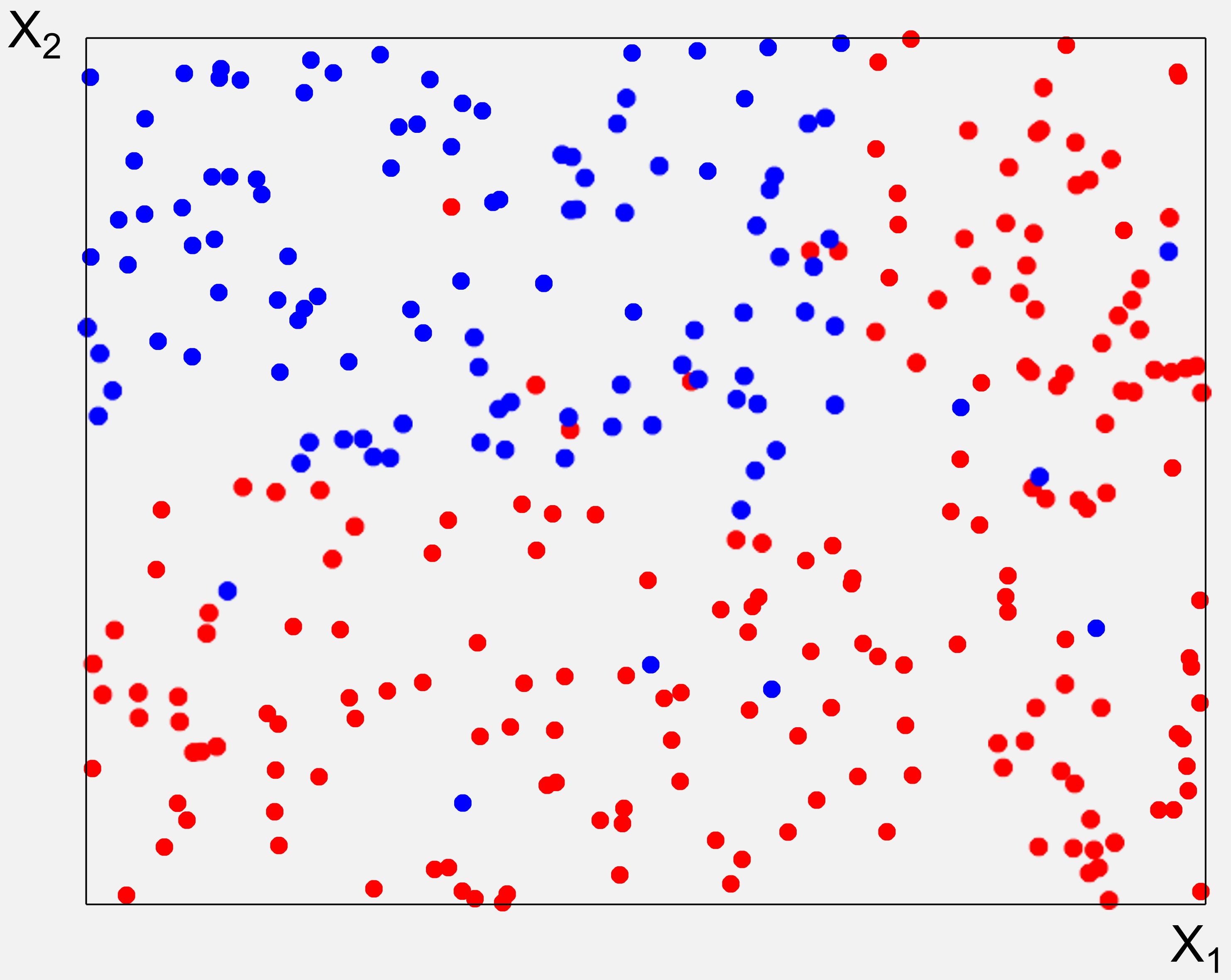


Figure credit:
Gilles Louppe

EXAMPLE: THIS 2-FEATURE DATA SET.

HOW SHOULD WE SPLIT?

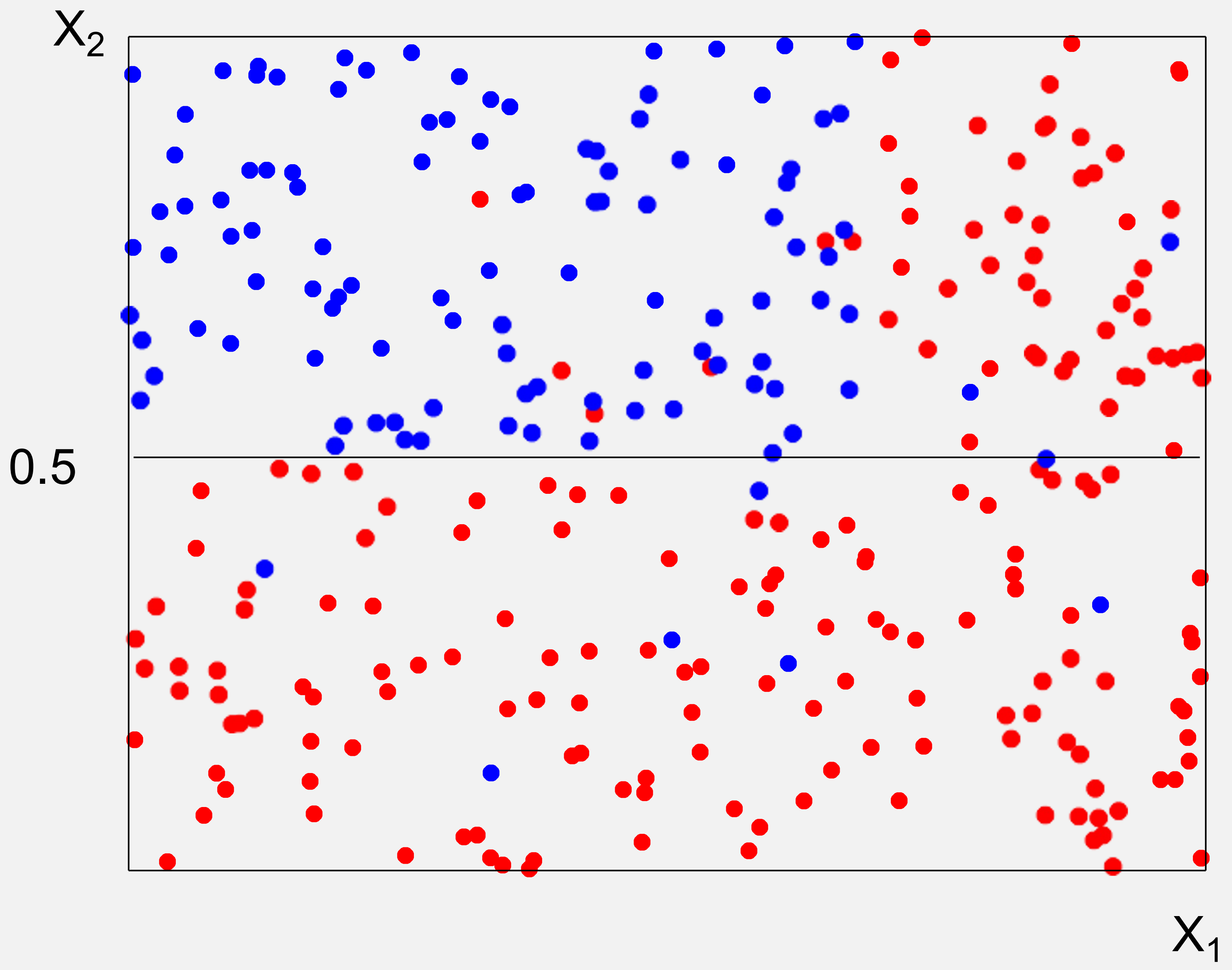


Figure credit:
Gilles Louppe

SHOULD WE STOP?

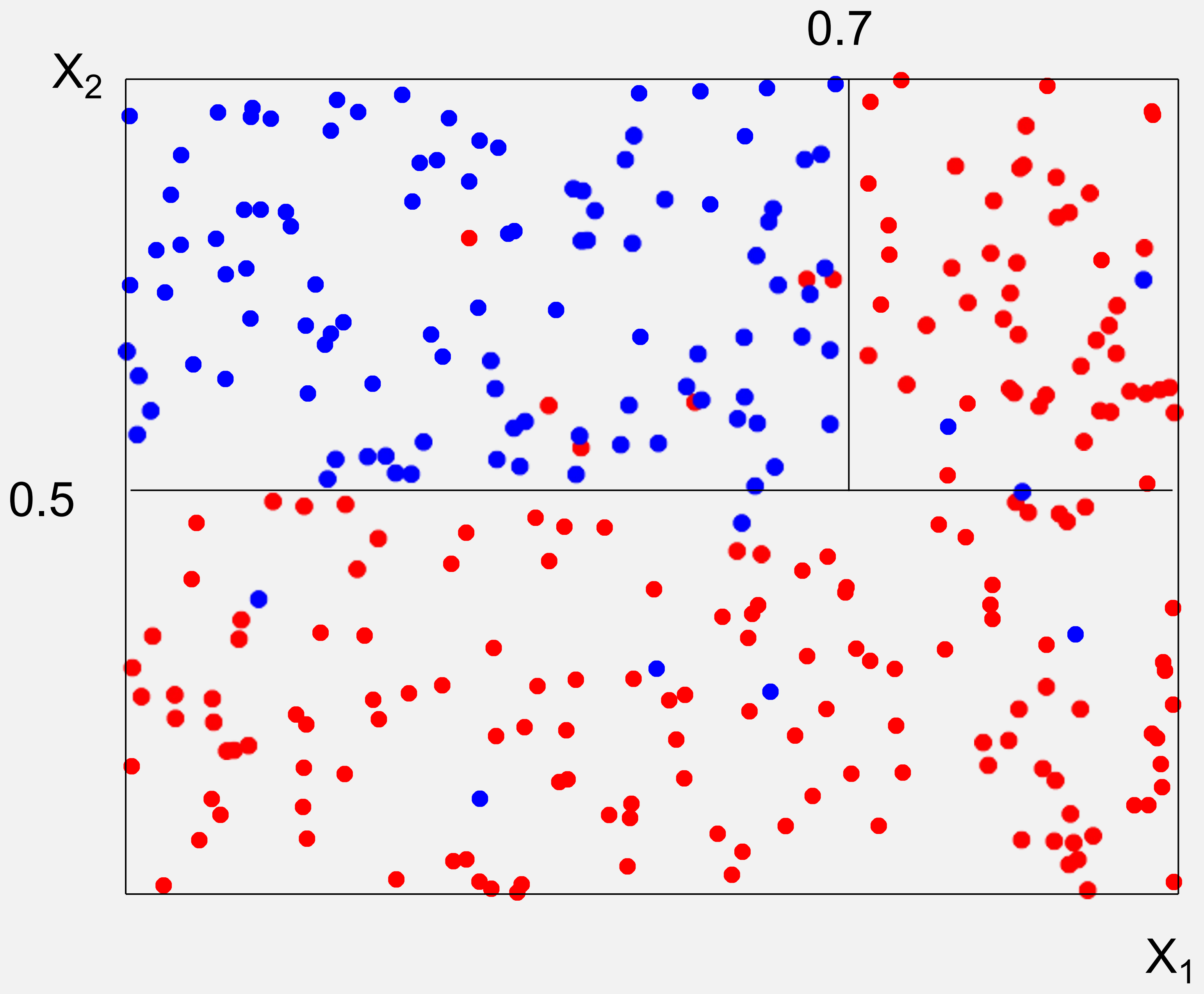


Figure credit:
Gilles Louppe

NODES (SPLITS AND LEAVES) DEFINE THE DECISION TREE

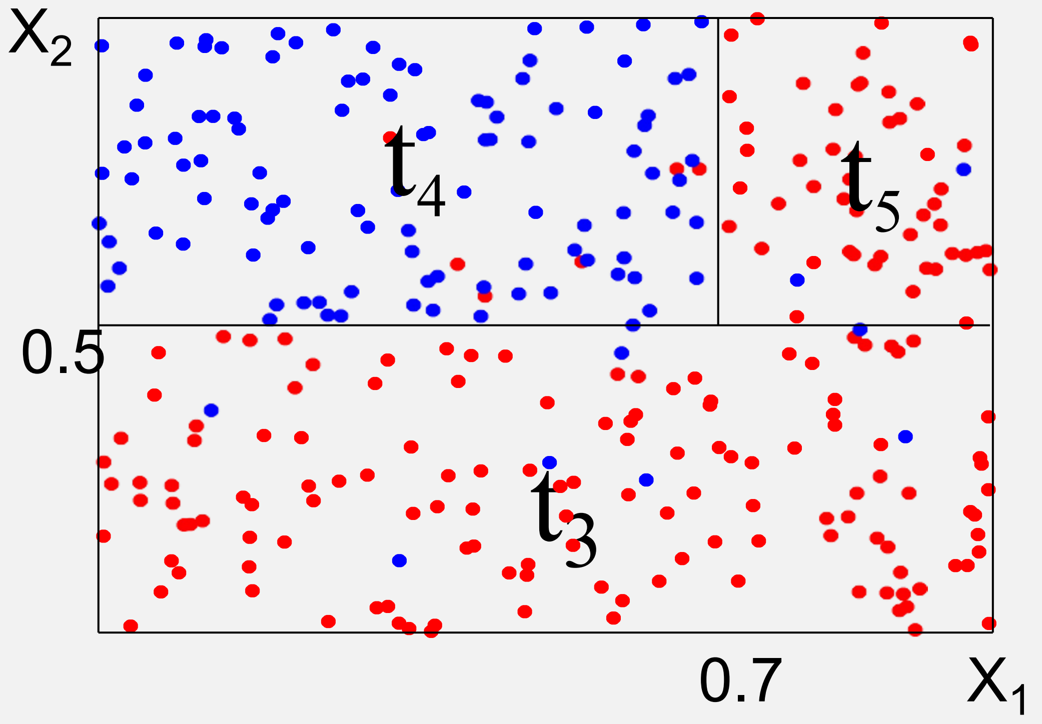
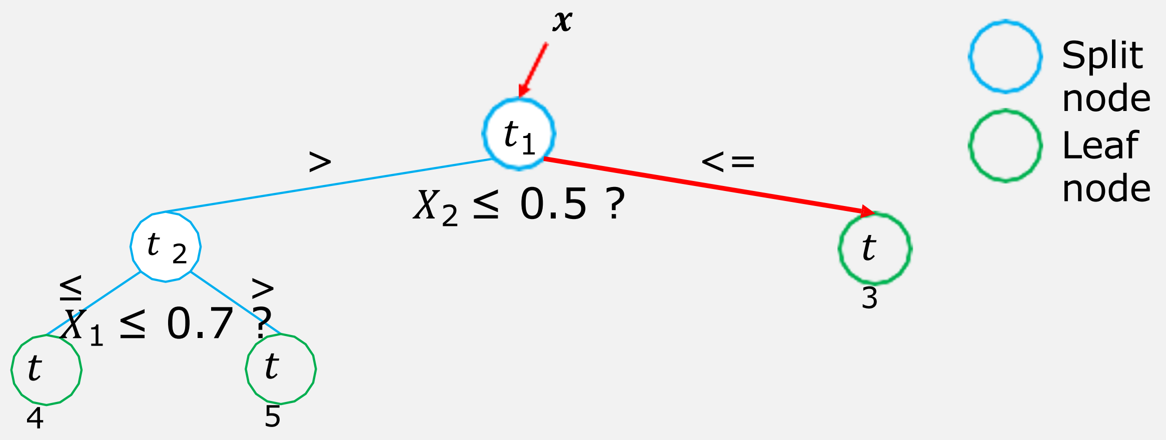


Figure credit:
Gilles Louppe



In a terminal node (leaf),
the model is ready to output a classification
(and all objects in that leaf have the same class)

Important questions:

How do we decide which splits to make, among the many possible ones?

How do we decide whether we should stop?

Measure of impurity: Gini impurity

Gini impurity is a metric used in decision trees to measure how "impure" or "mixed" a dataset is. It quantifies the likelihood that a randomly chosen element from the dataset would be incorrectly classified if it were randomly labeled according to the class distribution.

Formula for Gini Impurity

For a dataset with C different classes, the Gini impurity is defined as:

$$G = 1 - \sum_{i=1}^{C} p_i^2$$

where:

- p_i is the proportion of elements in class i.

Intuition

- **Pure Node**: If a node contains only one class (e.g., all samples belong to class A), then $p_A = 1$ and $G = 1 - 1^2 = 0$, meaning zero impurity.
- **Maximal Impurity**: If the classes are evenly distributed (e.g., 50% class A and 50% class B), then:

$$G = 1 - (0.5^2 + 0.5^2) = 1 - 0.5 = 0.5$$

  More evenly mixed distributions have higher Gini impurity.

Example Calculation

Imagine a dataset with three classes: A, B, and C, with the following proportions:

- $p_A = 0.5$
- $p_B = 0.3$
- $p_C = 0.2$

Then:

$$G = 1 - (0.5^2 + 0.3^2 + 0.2^2)$$

$$G = 1 - (0.25 + 0.09 + 0.04) = 1 - 0.38 = 0.62$$

A higher Gini impurity means the data is more mixed.

Building a decision tree

1. **Calculate Gini Impurity for the Root Node**

   Compute the Gini impurity of the entire dataset before splitting.

2. **Consider Possible Splits**

   For each feature, evaluate possible split points.

3. **Compute Gini Impurity for Each Split**

   - Partition the dataset into left and right child nodes.
   - Compute the weighted Gini impurity of the split:

   $$G_{\text{split}} = \frac{N_{\text{left}}}{N} G_{\text{left}} + \frac{N_{\text{right}}}{N} G_{\text{right}}$$

   where N_{left} and N_{right} are the sizes of the left and right subsets.

4. **Choose the Best Split**

   The split that **minimizes** G_{split} is chosen.

5. **Recursively Split Until a Stopping Condition is Met**

   Stop splitting when:

   - All samples in a node belong to one class.
   - A maximum tree depth is reached.
   - A minimum number of samples per node is reached.

Decision trees

3. Example: Habitable planets with scikit-learn

Reference: “Machine Learning for Physics and Astronomy” by Viviana Acquaviva, Princeton University Press, Chapter 2.

Search for Habitable Planets

- **Our first example will be a supervised classification problem, in which we attempt to predict if a planet is habitable, based on its distance from parent star, mass of parent star, and orbital period. This is not really a practically useful example, but it lets us visualize decision trees on a very small data set.**

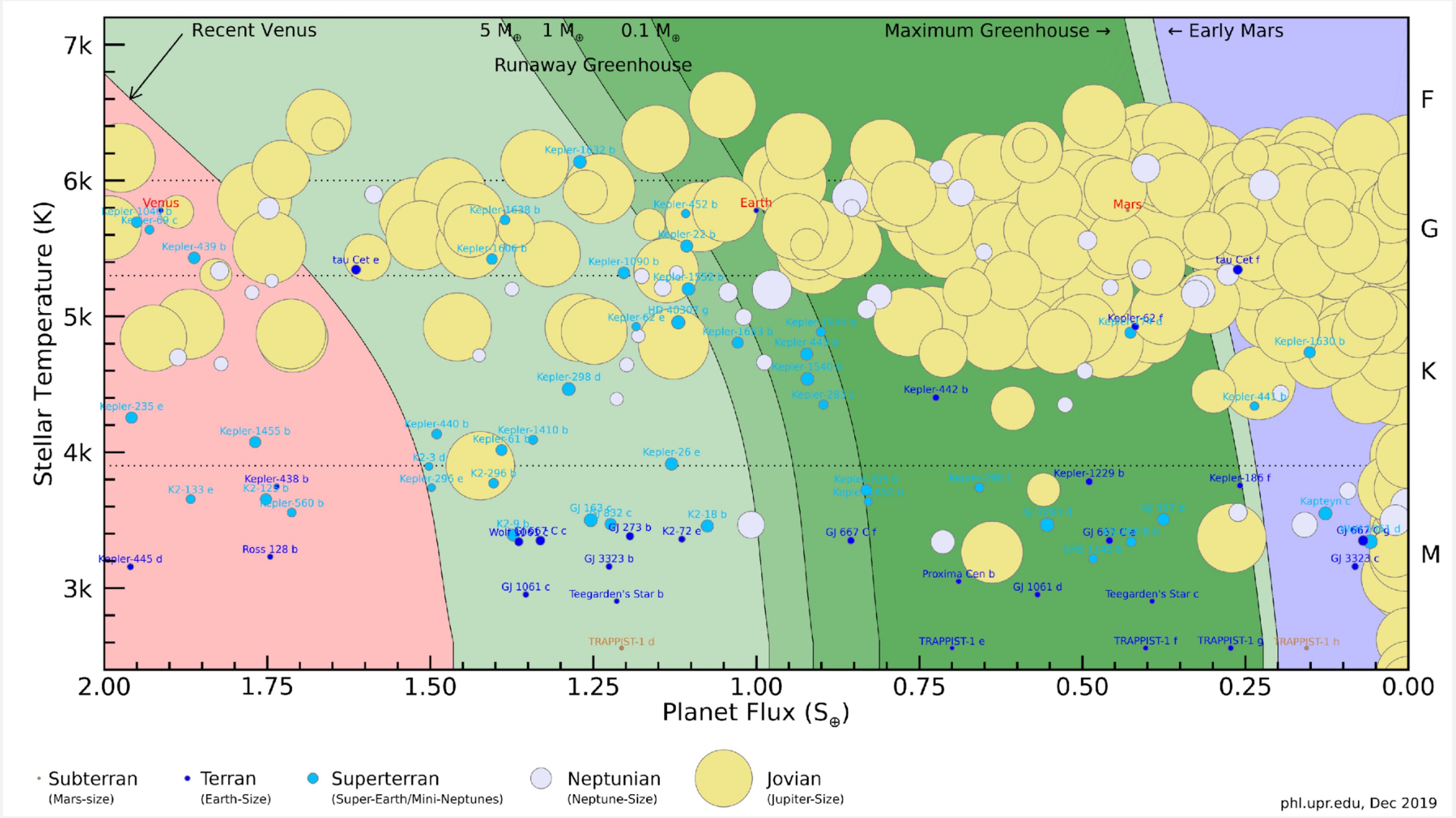
- More than 5000 exoplanets: https://exoplanets.nasa.gov/alien-worlds/historic-timeline/#first-transiting-exoplanet-observed

- Finding habitable planets (density and temperature conditions etc that are compatible with the development of life).

- Planet Habitability Laboratory website: https://phl.upr.edu/projects/habitable-exoplanets-catalog contains data for thousands of planets and collects a variety of features.

- We consider a learning set composed of 18 instances and their 3 features.

DATA FROM THE PLANET HABITABILITY LAB AT ARECIBO OBSERVATORY



Name	Stellar Mass ($M_\odot$)	Orbital Period (days)	Distance (AU)	Habitable?
Kepler-736 b	0.86	3.60	0.0437	0
Kepler-636 b	0.85	16.08	0.1180	0
Kepler-887 c	1.19	7.64	0.0804	0
Kepler-442 b	0.61	112.30	0.4093	1
Kepler-772 b	0.98	12.99	0.1074	0
Teegarden's Star b	0.09	4.91	0.0252	1
K2-116 b	0.69	4.66	0.0481	0
GJ 1061 c	0.12	6.69	0.035	1
HD 68402 b	1.12	1103	2.1810	0
Kepler-1544 b	0.81	168.81	0.5571	1
Kepler-296 e	0.5	34.14	0.1782	1
Kepler-705 b	0.53	56.06	0.2319	1
Kepler-445 c	0.18	4.87	0.0317	0
HD 104067 b	0.62	55.81	0.26	0
GJ 4276 b	0.41	13.35	0.0876	0
Kepler-296 f	0.5	63.34	0.2689	1
Kepler-63 b	0.98	9.43	0.0881	0
GJ 3293 d	0.42	48.13	0.1953	1

Table 2.1: Learning set for the habitable planets problem.

Comments on the dataset

- Well balanced: 10 examples in one category (not habitable) and 8 in the other (habitable).
- A factor that determines whether a planet is habitable is temperature, which likely depends on the energy it receives from its parent star.
- The planet’s temperature therefore depends on the star’s luminosity and its **distance** from the planet.
- **Mass** of a star is a decent tracer of its luminosity (as is the case for main sequence stars).
- Reality is more complicated: the energy budget of each planet also depends on other features, e.g., properties of its atmosphere, & whether it has an internal energy source.
- Mass/luminosity relationship is monotonic only for main sequence stars, which make up only about 90% of the total.

Predicting planet habitability

- On canvas, under DT-kNN-Notebooks, you will find a Jupyter Notebook: Intro_DT_HabPlanets.ipynb and a dataset: HPLearningSet.csv.
- Split into a training and a test set:

```
LearningSet = pd.read_csv (`HPLearningSet.csv')
TrainSet = LearningSet.iloc[;13,:]
TestSet = LearningSet.iloc[13:,:]
```

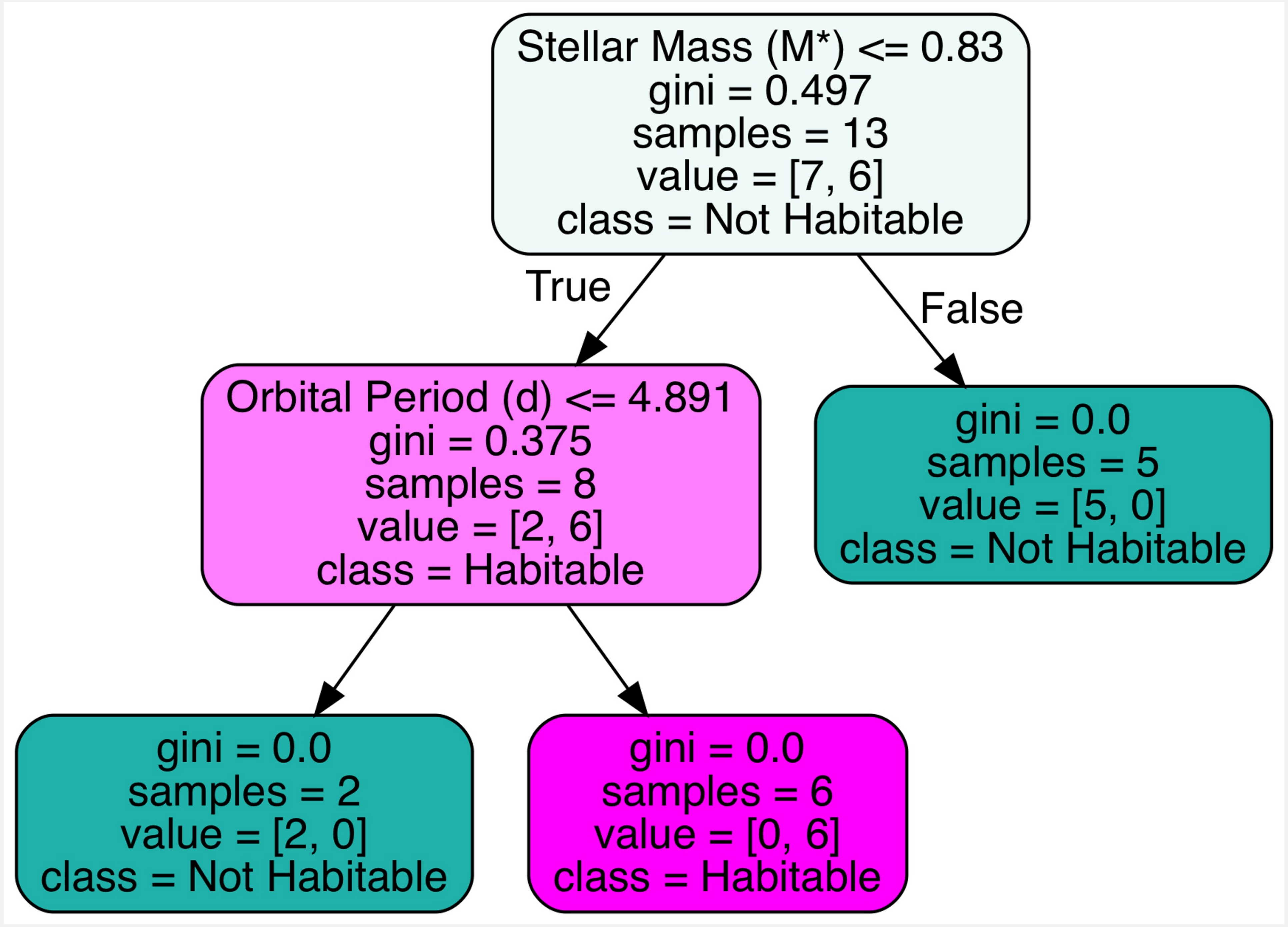
- The file contains both features & labels, separate them into 4 arrays:

```
Xtrain = TrainSet.drop([`P_NAME',`P_HABITABLE'], axis = 1)
XTest = TestSet.drop([`P_NAME',`P_HABITABLE'], axis = 1)
ytrain = TrainSet.P_HABITABLE
ytest = TestSet.P_HABITABLE
```

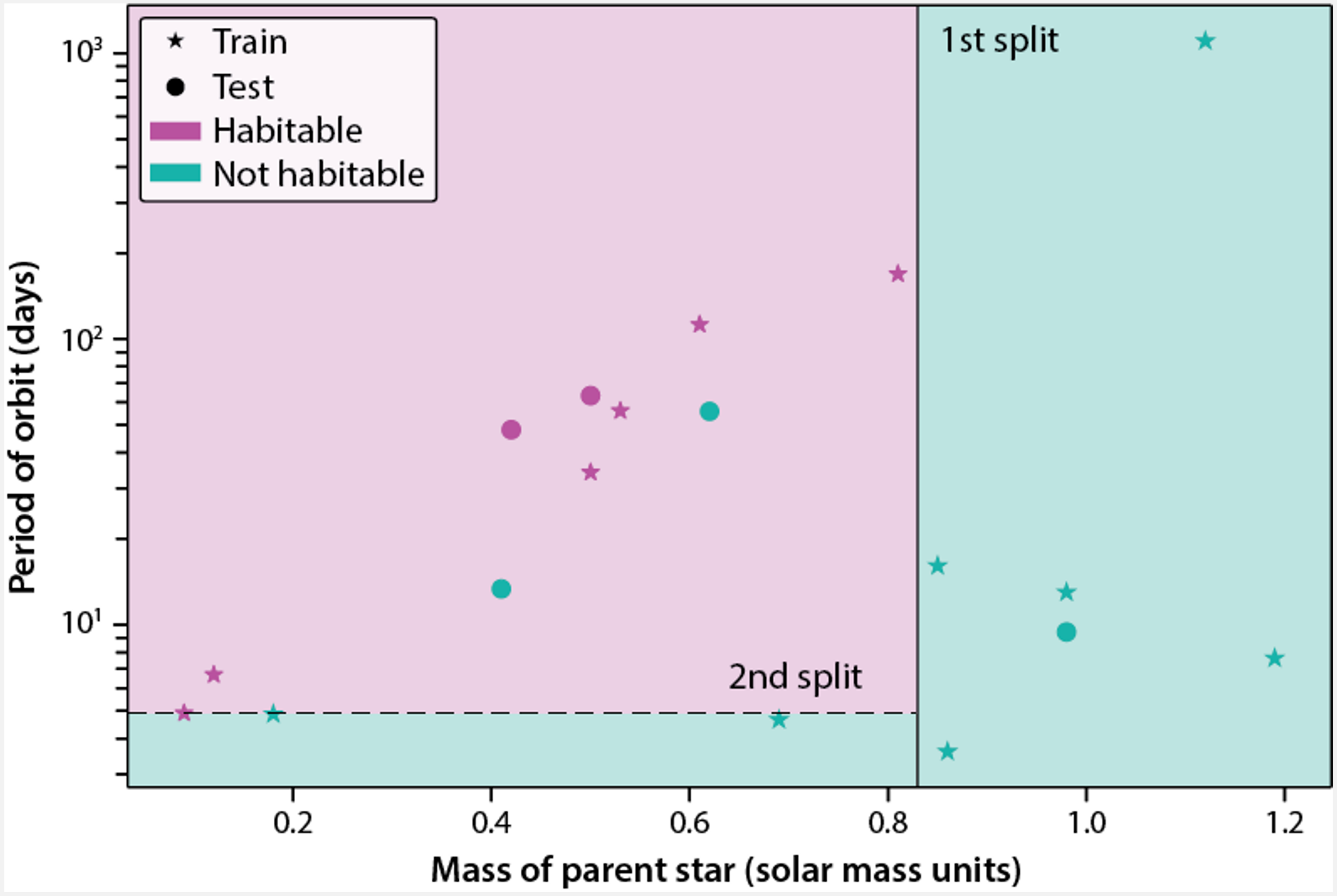
- Import the Decision Tree Classifier from sklearn and build the decision tree using the “fit” method.

```
model = DecisionTreeClassifier (random_state = 3)
model.fit(Xtrain, ytrain)
```

LET'S SEE WHAT SKLEARN SAYS

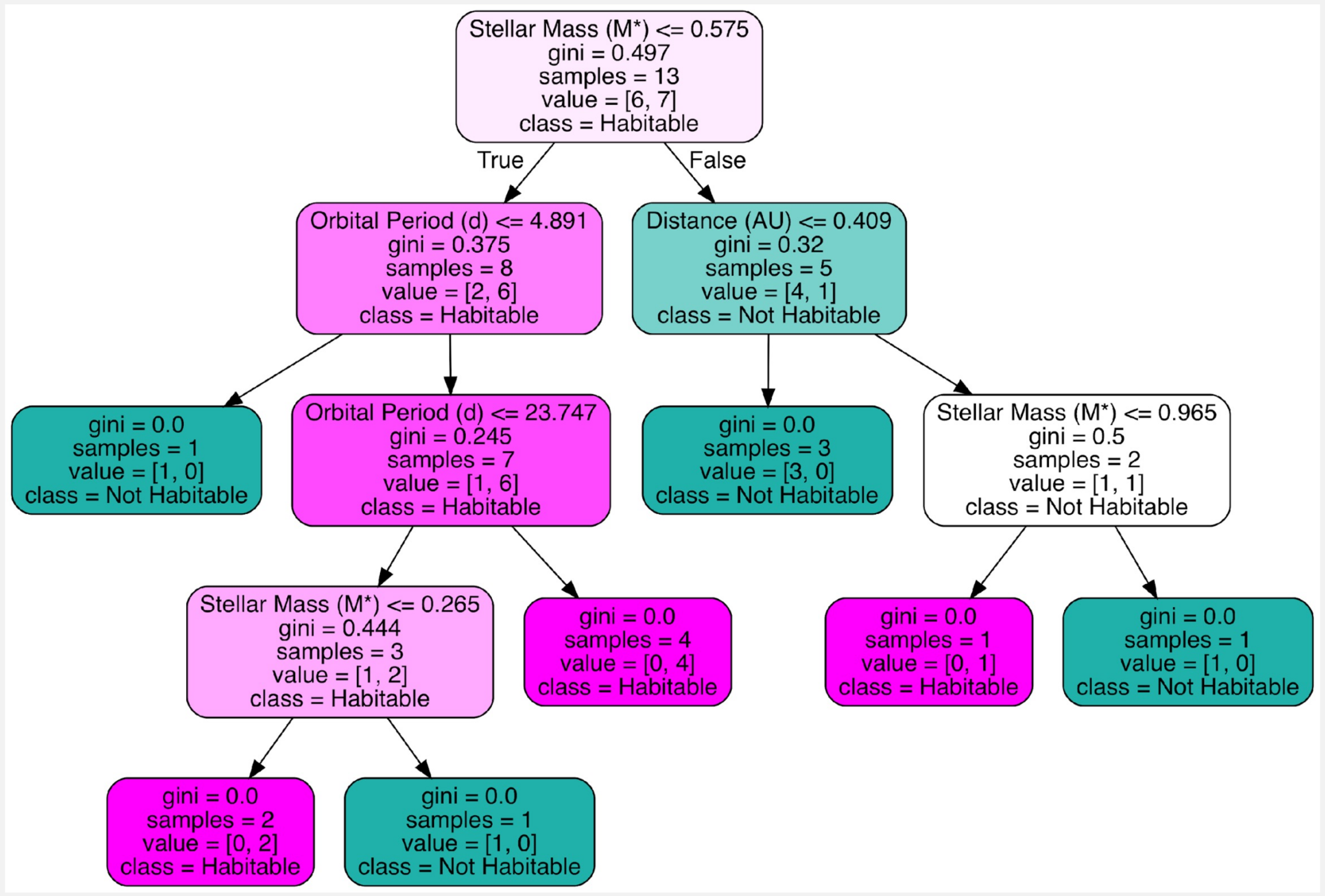


AND VISUALIZE THE CRITERIA THAT WE FOUND!



Can you figure out the accuracy on the test set?

NOTE (AND WE'LL SEE CODE FOR IT): IF YOU USE THE LAST 13 ROWS FOR TRAINING AND THE FIRST 5 FOR TESTING, YOU GET THIS TREE:



and 100% accuracy on test set

Morale: Different train/test split might give significantly different performances, especially when data sets are small.

Decision tree ensembles

1. Random Forrests
("Bagging")

Ensemble methods

- One way to boost the performance (both for classification and regression) is to aggregate the response of several models.

- For example in **classification we could take the majority vote, perhaps weighted by some factor** if different models have different precision.

- This combined model often gives better combined accuracy than the single constituents.

- **Reasons:**
  - Aggregating several base learners generally **reduces the variance.**
  - Single models may get **stuck in different local minima**.
  - The combined model has a higher capacity than the constituents and may fit the data better.
  - **Single models may be biased** in opposite directions so the biases may cancel out in some situations.

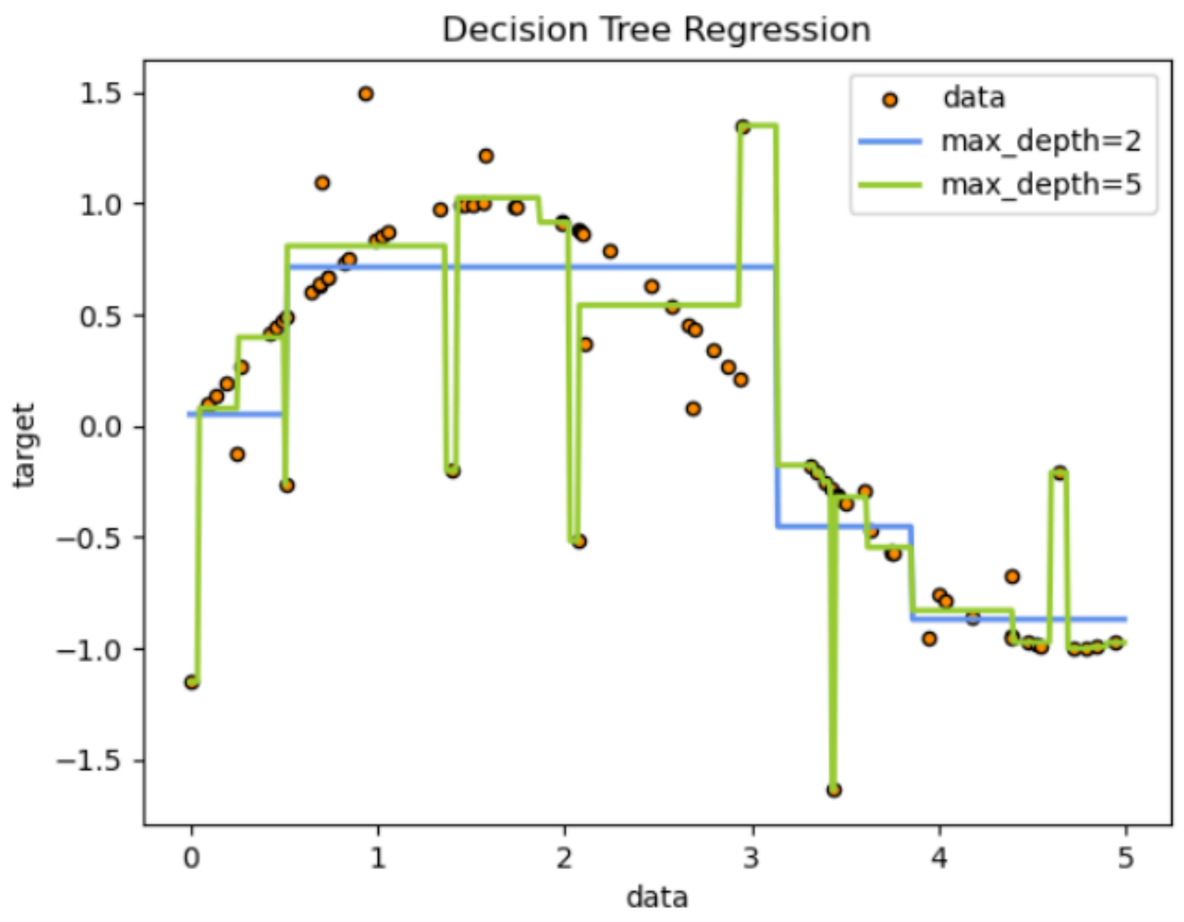
- A popular set of models for ensemble training are decision trees. **A set of decision trees is called a forest :)**

Trees for regression problems

“Decision trees” are also useful for regression problems. They are then often called “Regression trees”.

Regression trees assign a continuous value to each leaf.

They thus approximate the function as piecewise constant.



https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html

Random Forests

- Random Forests are a **collection of randomized decision trees**.

- **Randomization ("Bootstrapping") occurs in two ways:**
  - Take many different random subset of the training data (where elements can repeat).
  - Take random subsets of the features.

- We train many random trees based on these randomized data sets.

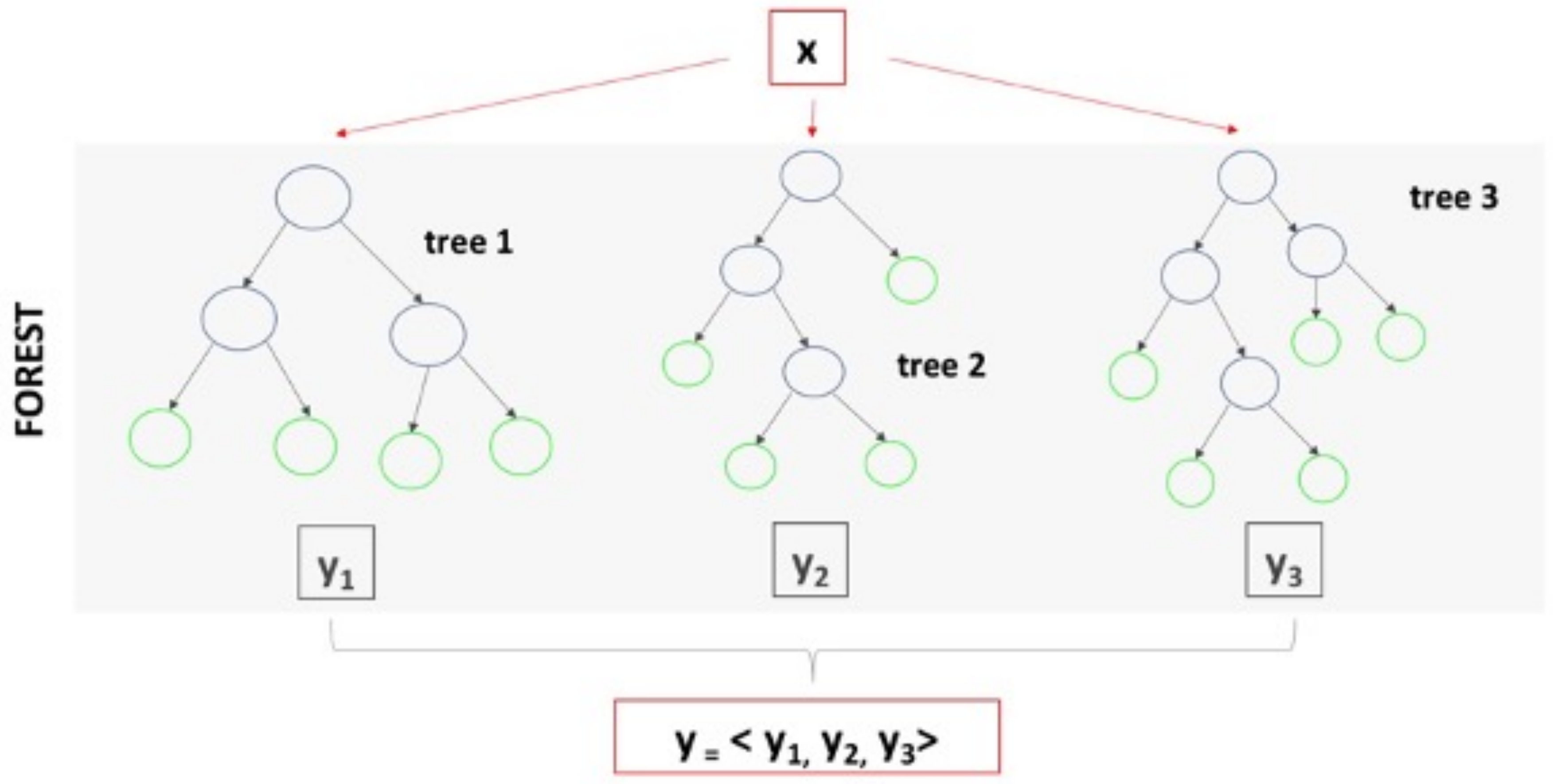
- The final outcome is the "mean" of the many trees.

- The approach we just described is called "**bagging**". The name comes from Bootstrap AGGregating.

- Typical hyper parameters: number of trees and the number of features in the bootstrap subset.

- Implementation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

How to combine trees



- The final prediction (class, or number) is typically the
  - average of all predictions (for a regression problem),
  - or the majority vote (for a classification problem)

Feature Importance

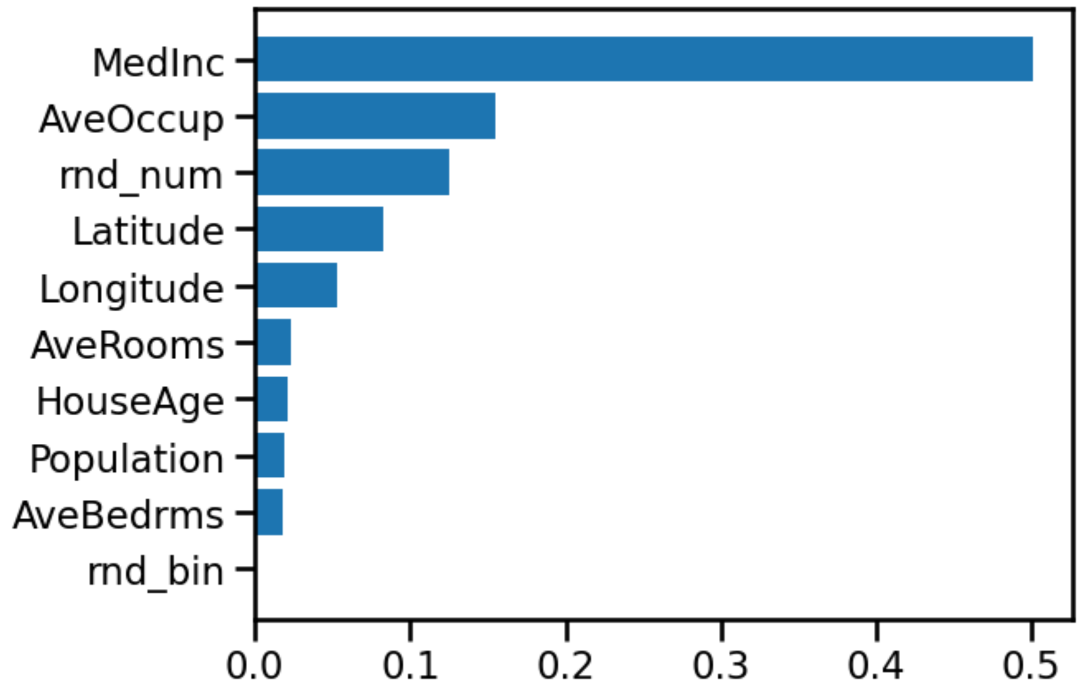
A nice feature of random forests and other ensembles of decision trees is that one can evaluate which features of the data are more important than others.

The result of this analysis is somewhat algorithm dependent but is still informative.

More important features appear earlier in the tree, since they lead to a large decrease in impurity.

To quantify the importance **we can sum up the impurity improvements of all the splits associated with a given variable**. One can then rank the features. Correlation can complicate the interpretation.

Example: predicting the **median house value** (target) given some information about the neighborhoods, as the average number of rooms, the latitude, the longitude or the median income of people in the neighborhoods (block).
https://inria.github.io/scikit-learn-mooc/python_scripts/dev_features_importance.html



Median income is still the most important feature.

Decision tree ensembles

2. Gradient Boosted Decision Trees (“Boosting”)

Boosting vs Bagging

- **Bagging means that we average over many weaker models**, which are trained independently.

- **Boosting works sequentially**: A weak (simple) learner is created to make prediction. Then more weak learners are iteratively added to get problematic examples right ("boosting the success rate").

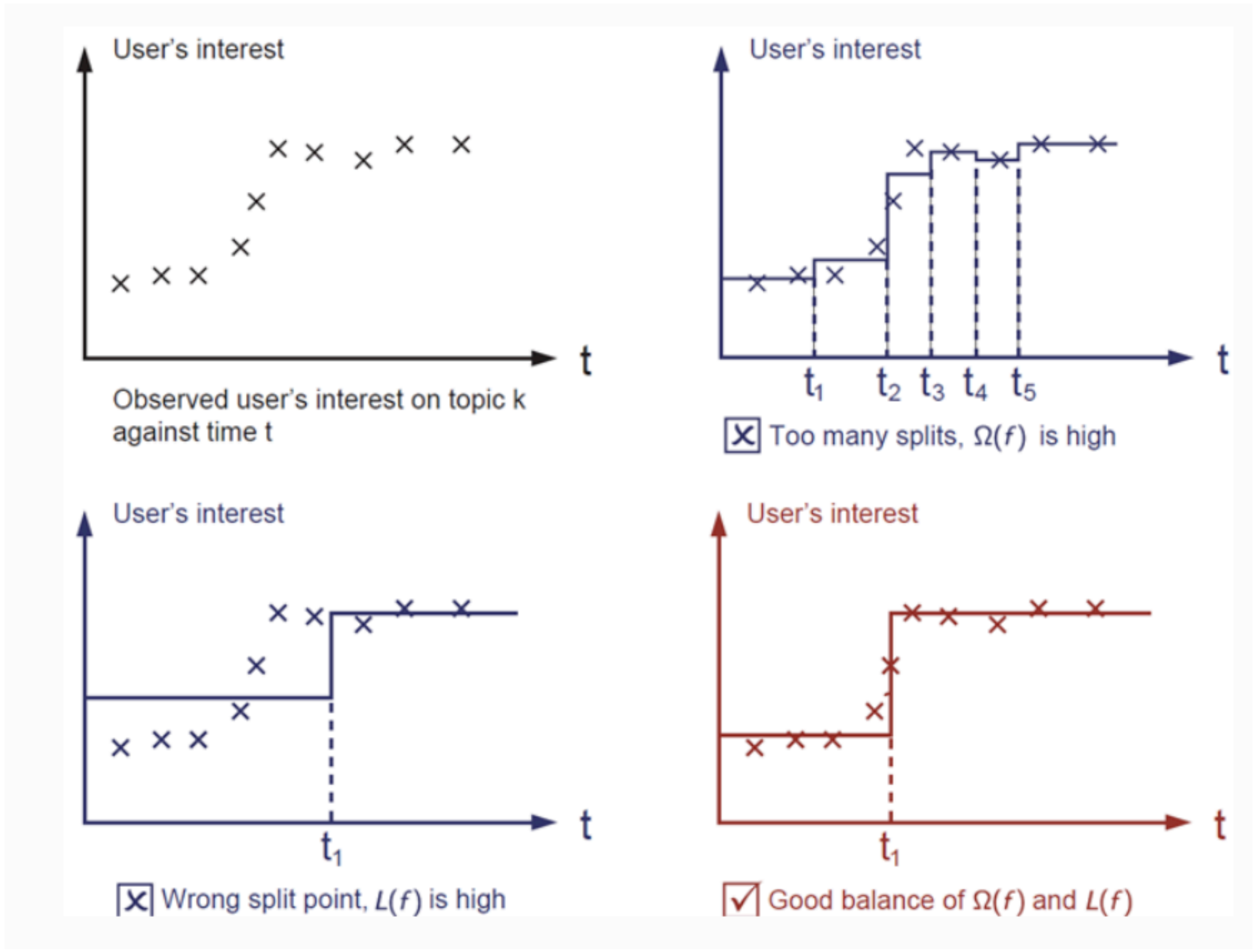
- Two popular algorithms are Adaptive Boosting (AdaBoost) and Gradient Boosting.

- I will focus on Gradient Boosting, which seems to be the dominant method. The **leading software implementations of Gradient Boosting are currently XGBoost and LightGBM**. They have **won countless kaggle competitions (**https://www.kaggle.com/competitions**).**

XGBoost

XGBoost stands for “Extreme Gradient Boosting”, where the term “Gradient Boosting” originates from the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman.

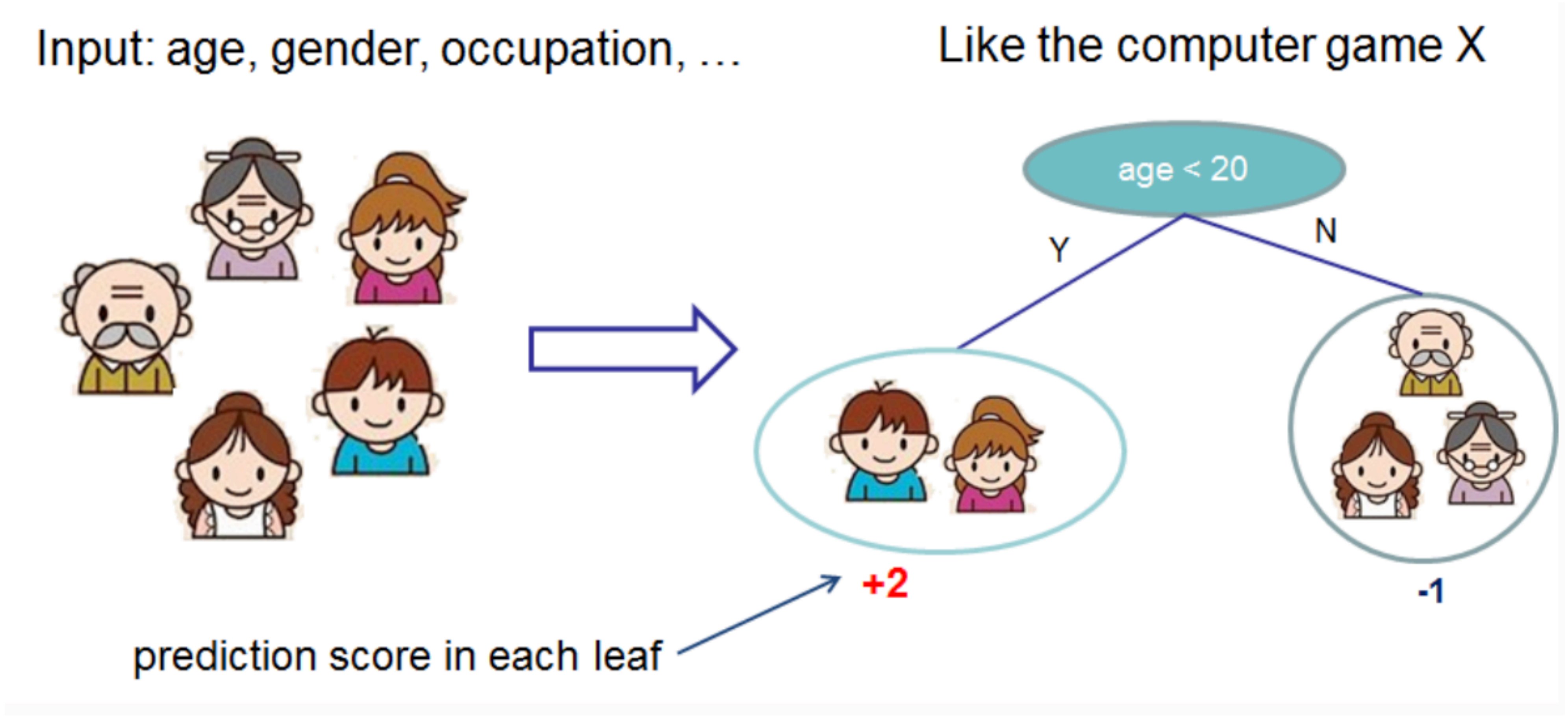
My introduction is based on https://xgboost.readthedocs.io/en/stable/tutorials/model.html (which contains mathematical details we have to skip over for time reasons)



Example: function fitting with a tree, finding the optimal tree complexity

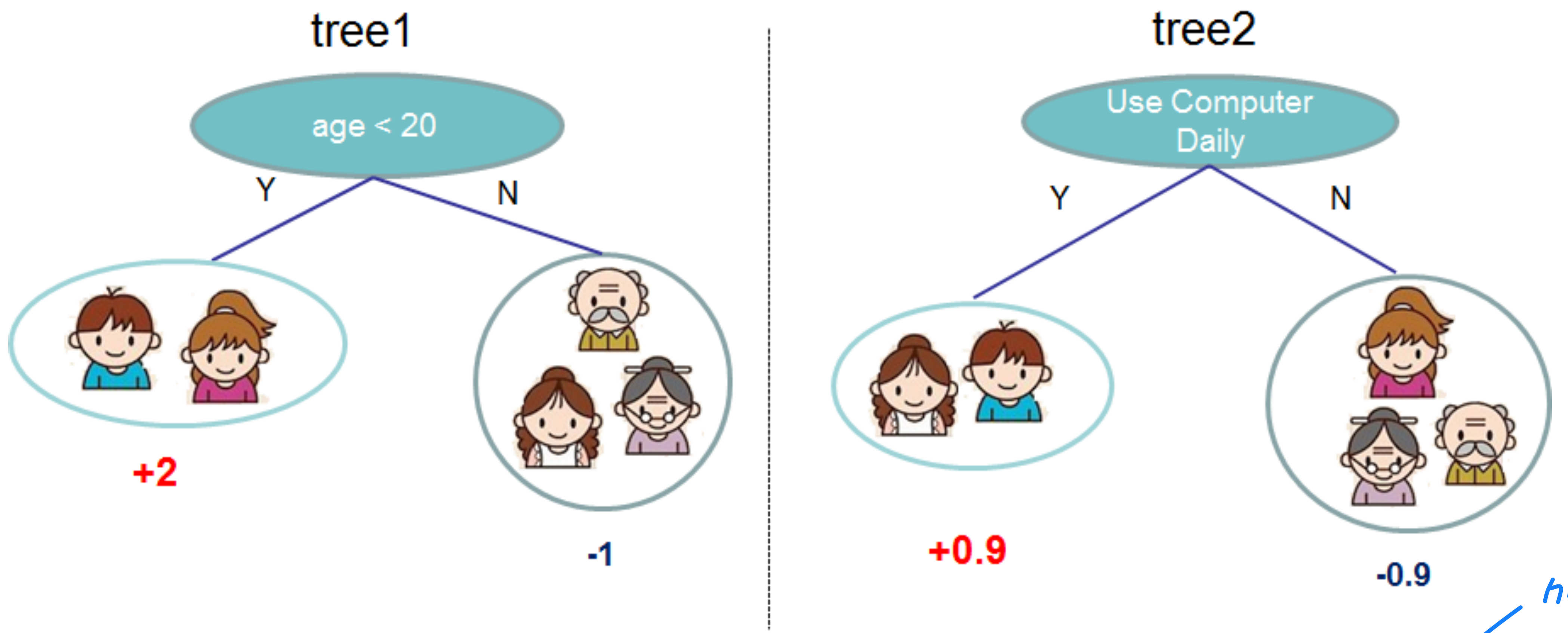
CART in XGBoost

The tree ensemble model of XGBoost consists of a set of **classification and regression trees (CART)**. A CART is a bit different from decision trees, in which the leaf only contains decision values. In CART, a real score is associated with each of the leaves, which gives us richer interpretations that go beyond classification.


Input: age, gender, occupation, …
Like the computer game X
age < 20
Y
N
+2
-1
prediction score in each leaf

We want an objective function for training the trees.

Usually, a single tree is not strong enough to be used in practice. What is actually used is the ensemble model, which sums the prediction of multiple trees together.



f( ) = 2 + 0.9= 2.9   f( )= -1 - 0.9= -1.9

nr of trees

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

output of tree k for example i

example losses:

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

MSE

$$L(\theta) = -\sum_i \sum_{j=1}^{C} y_{i,j} \ln p_{i,j}$$

cross-entropy

$$\underbrace{\text{obj}(\theta)}_{\text{objective}} = \underbrace{L(\theta)}_{\text{training loss}} + \underbrace{\Omega(\theta)}_{\text{regularization}}$$

Tree Boosting

- We want to optimize the loss function by adjusting the parameters of the trees.
- What are the parameters of trees? the structure of the tree and the leaf scores.
- Learning tree structure is much harder than traditional optimization problem where you can simply take the gradient. It is intractable to learn all the trees at once. Instead, we use an additive strategy: fix what we have learned, and add one new tree at a time **("greedy algorithm")**.
- It remains to ask: which tree do we want at each step? A natural thing is to add the one that optimizes our objective, i.e. the sum of the loss function and the regularization.

$$\text{obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \omega(f_i)$$

- The regularization is given by the complexity of the tree. One way to measure this is

$$\omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Where T is the number of leaves and w are the scores of the leaves.

Tree Boosting: Greedy algorithm

The iteratively added tree is found with a greedy algorithm:

Step 1: Initialization

- The algorithm starts with all training instances in the root node.
- At each node, it evaluates all possible splits across all features.

Step 2: Evaluating Splits

- For each feature, the potential splits are considered. The algorithm sorts the values of the feature and then iteratively evaluates the possible split positions between these sorted values. The "gain" from making a split is calculated based on how much it would reduce the loss function.

Step 3: Choosing the Best Split

- The algorithm selects the split with the highest gain. If no split results in a gain that meets the regularization criteria, the node is not split and becomes a leaf.

Step 4: Recursion

- This process is recursively applied to each resulting subset of the data (corresponding to each branch of the split) until one of the stopping criteria is met (max depth of the tree OR does not improve the loss by a significant amount OR having too few samples in a node)

Step 5: Outputting the Leaf Values

- Once the tree is fully grown and no more splits are made, the algorithm calculates the optimal output value for each leaf.

XGBoost improves upon this basic greedy algorithm by introducing several optimizations.

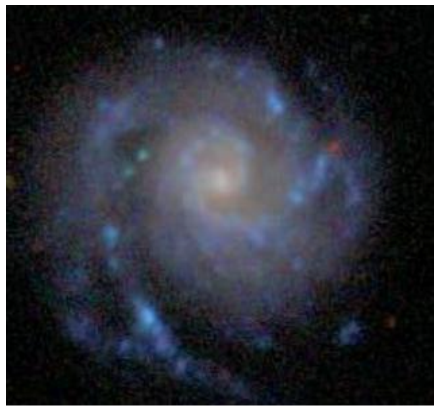
Decision tree ensembles

3. Application: Redshift estimation

Slides and python notebook from: Viviana Acquaviva - Machine Learning for physics and Astronomy chapter 6

Redshift of a galaxy

distance?



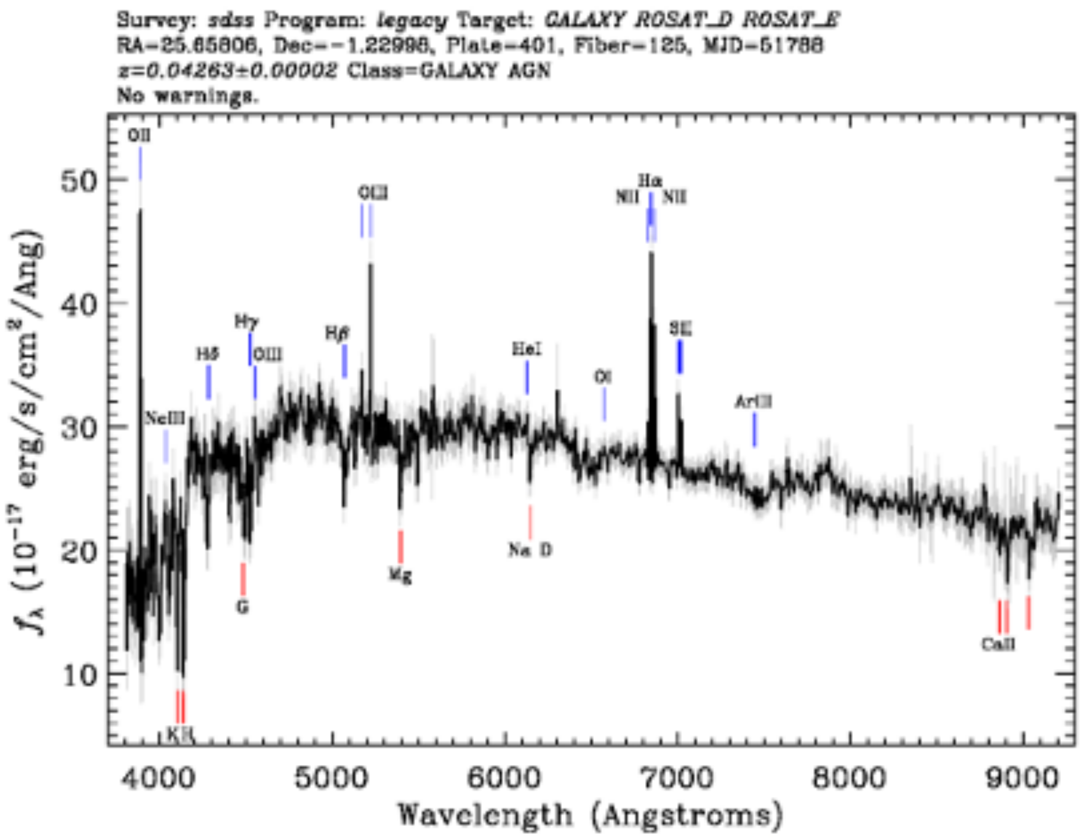
Astronomers measure the distance of a galaxy using its redshift. Because the universe is expanding, galaxies that are farther away have a higher redshift.

A spectrum is a high-resolution chart of brightness vs wavelength.

For galaxies that are further away, the spectrum is stretched
(all the wavelengths are longer).

Spectra contains spikes and dips, which corresponds to known transition in basic atoms (e.g., H, O).

If we can identify the emission lines we see (from the structure – one is not enough!), we can calculate the amount of stretch, which is 1 + z (a Doppler effect, essentially!). z is called the “redshift parameter”.



Example spectrum from the SDSS survey

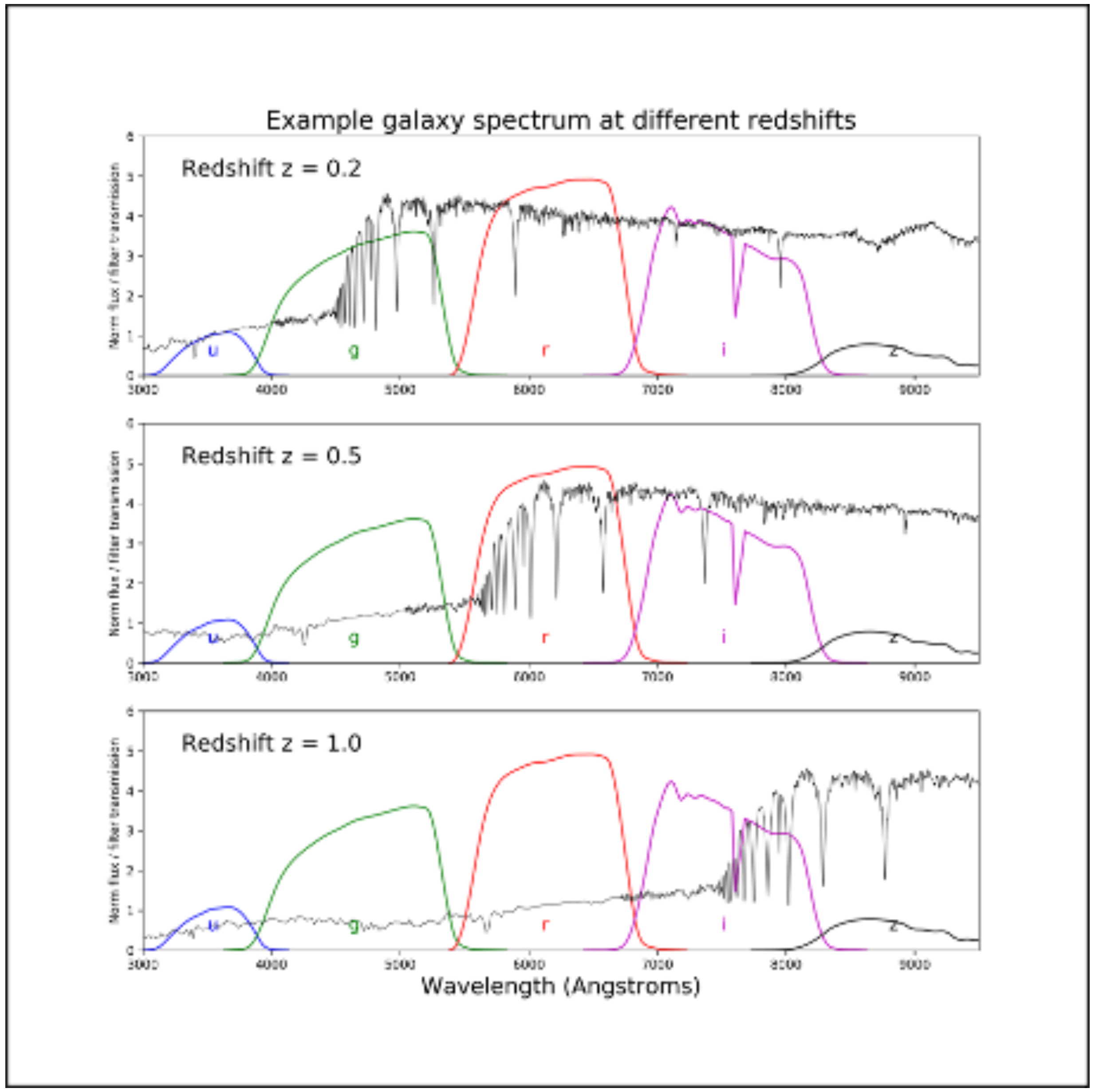
Photometric redshifts

In this case, we only have the average brightness over wide range of wavelengths, called filters or bands (1000s of Angstroms).

Much more challenging/less accurate than spectroscopic redshifts, but a lot cheaper to obtain!

Photo-z can be derived for billions of galaxies.

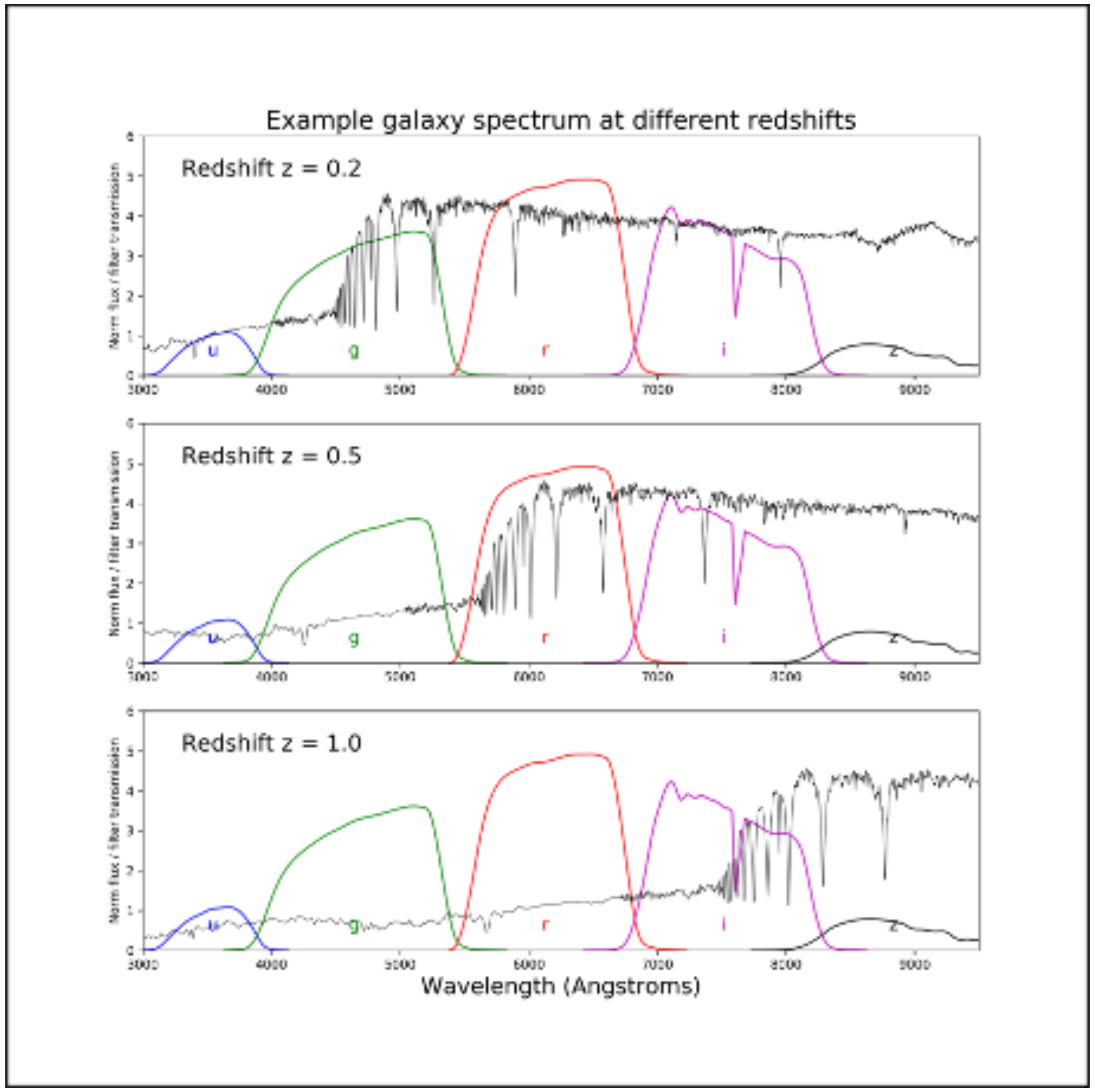
Spectroscopic redshifts (derived from line identification) can be used as a learning set for photometric redshifts.



Learning task

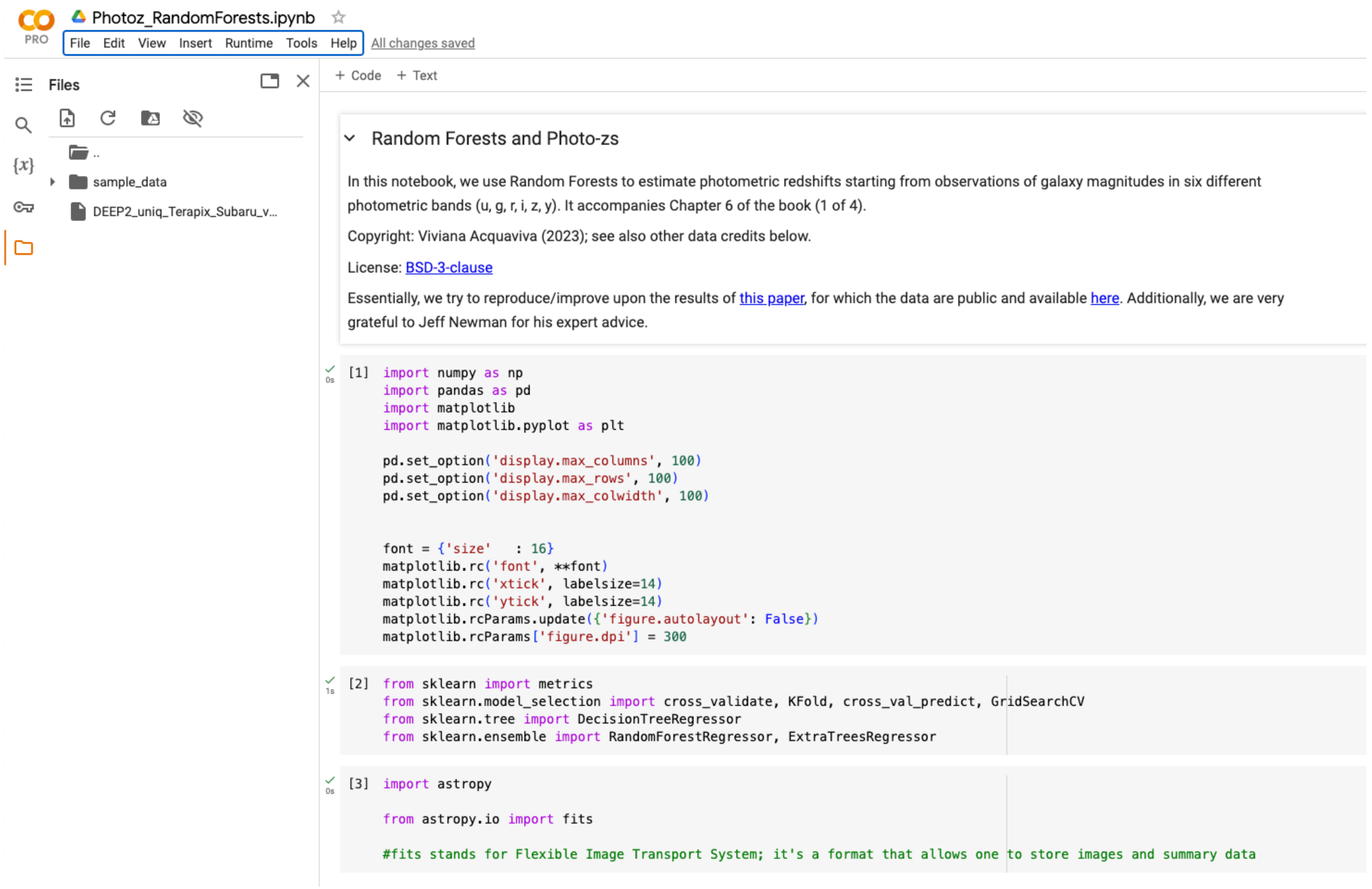
Input data:
Collection of photometric intensity in 6 bands (i.e. 6 numbers per galaxy)

Target data:
True redshift of the galaxy obtained from more expensive spectroscopy. 1 number called z.



Colab notebook

The rest of this lecture will be on Colab. We will use notebooks from the book Viviana Acquaviva "Machine learning for Physics and Astronomy". The notebooks can be downloaded on the course website, and on the book website https://press.princeton.edu/books/paperback/9780691206417/machine-learning-for-physics-and-astronomy.



CO PRO
Photoz_RandomForests.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
+ Code + Text
sample_data
DEEP2_uniq_Terapix_Subaru_v...

Random Forests and Photo-zs

In this notebook, we use Random Forests to estimate photometric redshifts starting from observations of galaxy magnitudes in six different photometric bands (u, g, r, i, z, y). It accompanies Chapter 6 of the book (1 of 4).

Copyright: Viviana Acquaviva (2023); see also other data credits below.

License: BSD-3-clause

Essentially, we try to reproduce/improve upon the results of this paper, for which the data are public and available here. Additionally, we are very grateful to Jeff Newman for his expert advice.

```
[1] import numpy as np
    import pandas as pd
    import matplotlib
    import matplotlib.pyplot as plt

    pd.set_option('display.max_columns', 100)
    pd.set_option('display.max_rows', 100)
    pd.set_option('display.max_colwidth', 100)


    font = {'size'   : 16}
    matplotlib.rc('font', **font)
    matplotlib.rc('xtick', labelsize=14)
    matplotlib.rc('ytick', labelsize=14)
    matplotlib.rcParams.update({'figure.autolayout': False})
    matplotlib.rcParams['figure.dpi'] = 300
```

```
[2] from sklearn import metrics
    from sklearn.model_selection import cross_validate, KFold, cross_val_predict, GridSearchCV
    from sklearn.tree import DecisionTreeRegressor
    from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
```

```
[3] import astropy

    from astropy.io import fits

    #fits stands for Flexible Image Transport System; it's a format that allows one to store images and summary data
```

Course logistics

- **Reading for this lecture:**
  - This lecture was based mostly on
    - Viviana Acquaviva “Machine learning for Physics and Astronomy” chapter 6.
    - https://xgboost.readthedocs.io/en/stable/tutorials/model.html