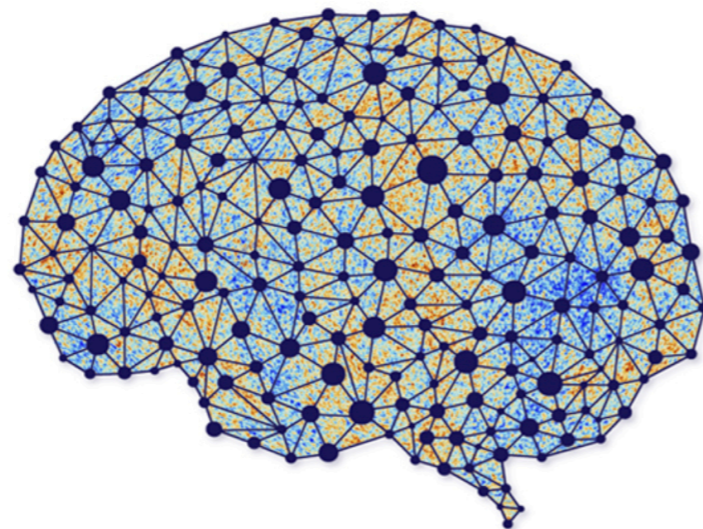


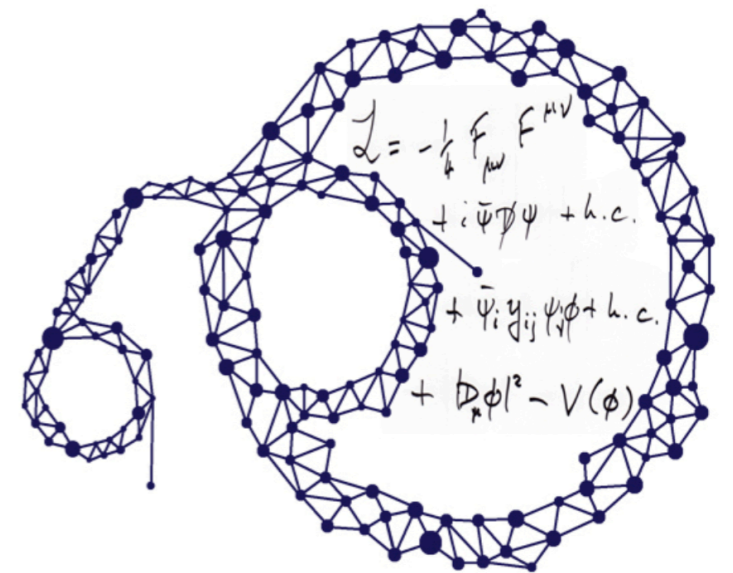
Physics 361 - Machine Learning in Physics

Lecture 17 – Random Forests and Gradient Boosted Trees

Mar. 19th 2024



AI
∩
Universe



Moritz Münchmeyer

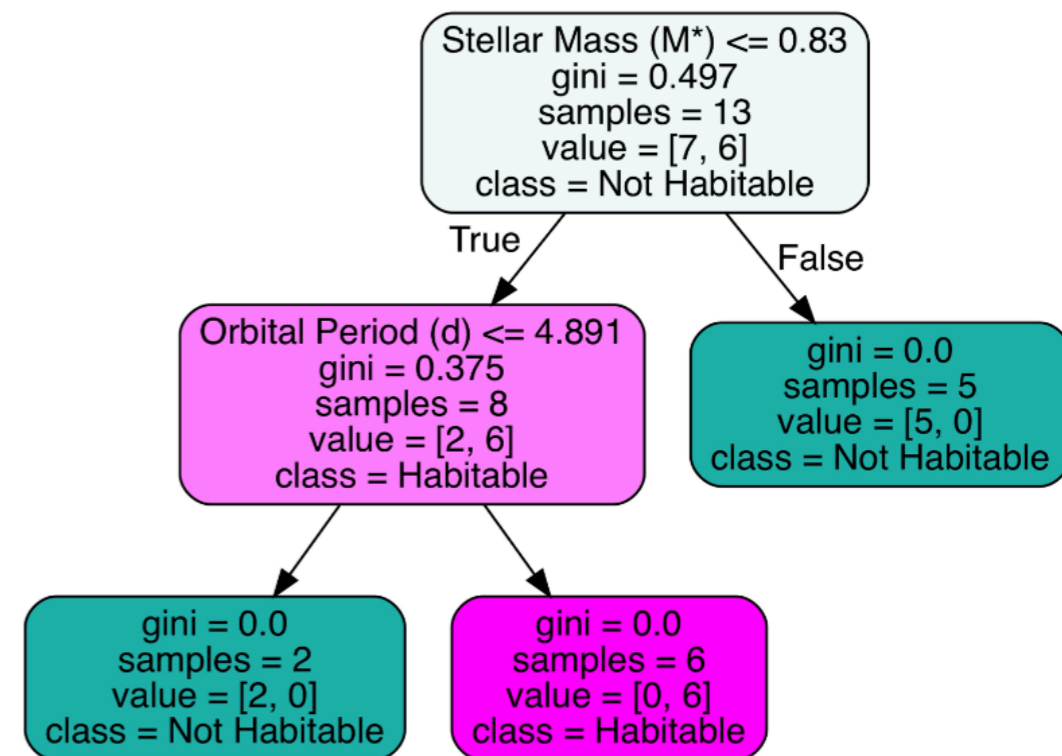
Decision tree ensembles

1. Motivation

Recall decision trees

- At each step we consider all possible splits on all possible features and chose the one that leads to the highest reduction in impurity of there resulting branches.
- Classification: common metric is Gini impurity
- Regression: common metric is the MSE
- Typical Hyperparameters:
 - Minimum reduction of impurity to do a split
 - Minimum number of samples required in leaf node
 - Maximum depth of the tree

Recall example from Gary's lecture:



The Gini impurity of a dataset is defined by the equation:

$$I_G(p) = 1 - \sum_{i=1}^J p_i^2$$

where:

- $I_G(p)$ is the Gini impurity for a set of items,
- J is the number of classes,
- p_i is the proportion of items labeled with class i in the set.

Power of decision tree methods

- **Decision trees methods are still the state-of-the-art for many tabular data applications.**
 - 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).

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 tree ensembles

2. Random Forests ("Bagging")

Ensemble methods

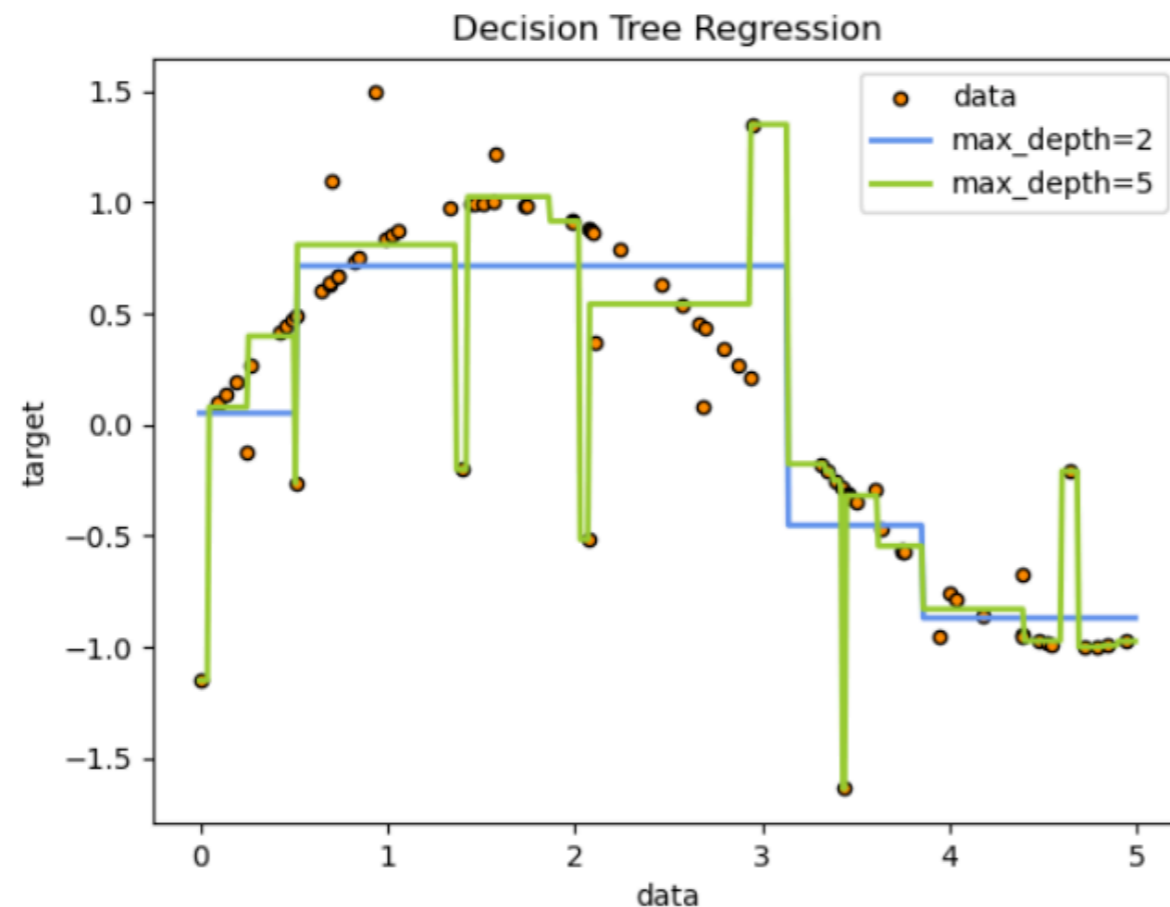
- One way to boost the performance (both for classification and regression) is to aggregate the response of several models.
- For example in classification we could take the majority vote, perhaps weighted by some factor if different models have different precision.
- This combined model often gives 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.

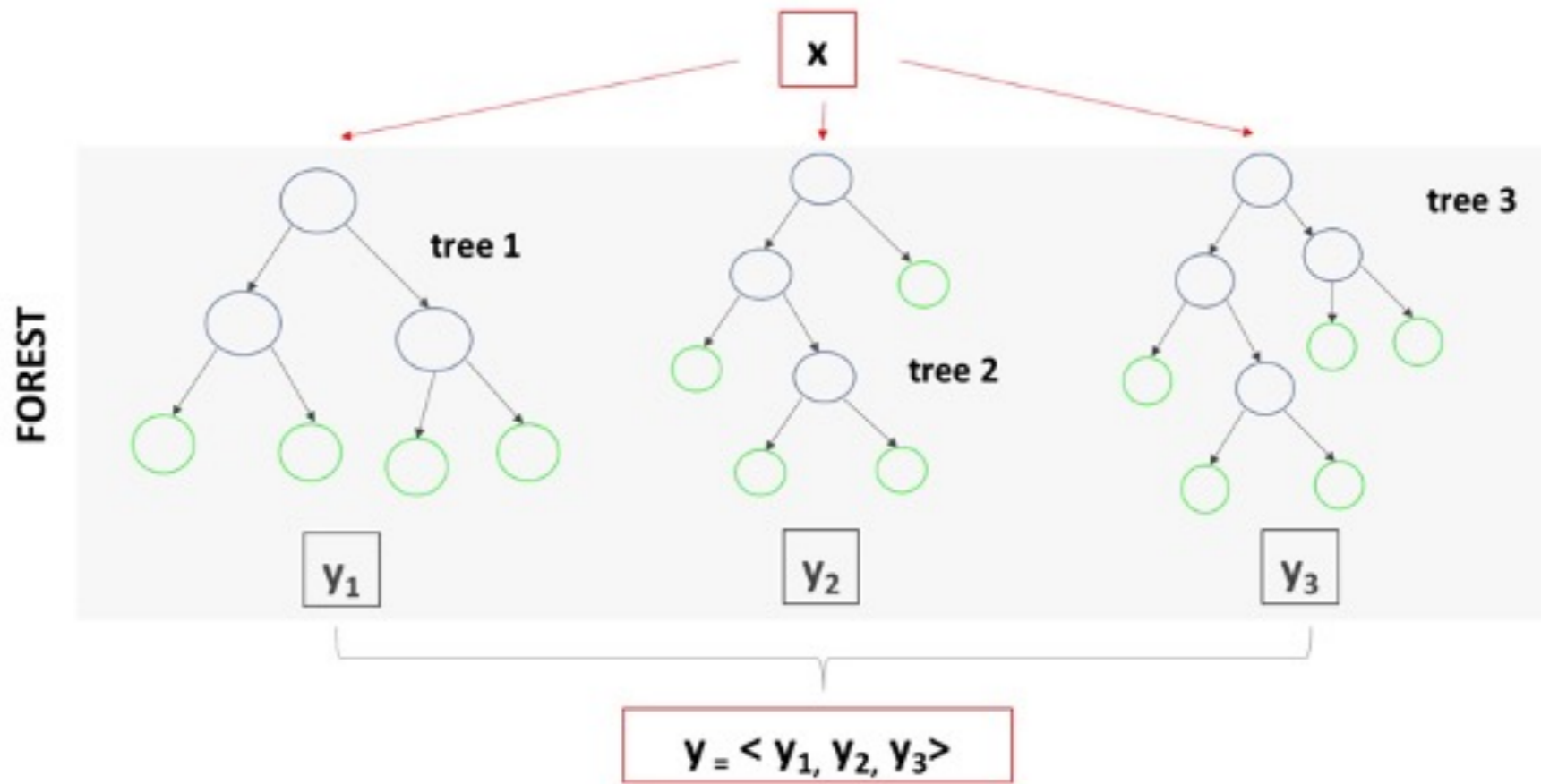


https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html

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.

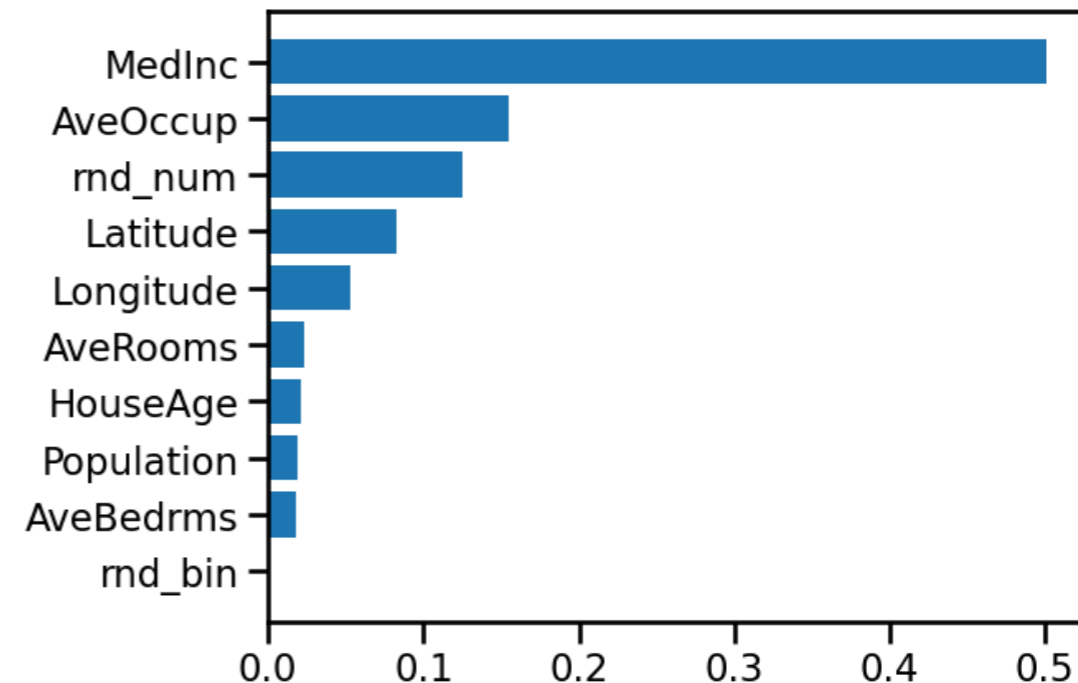
The result of this analysis is somewhat algorithm dependent but is still informative.

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”)

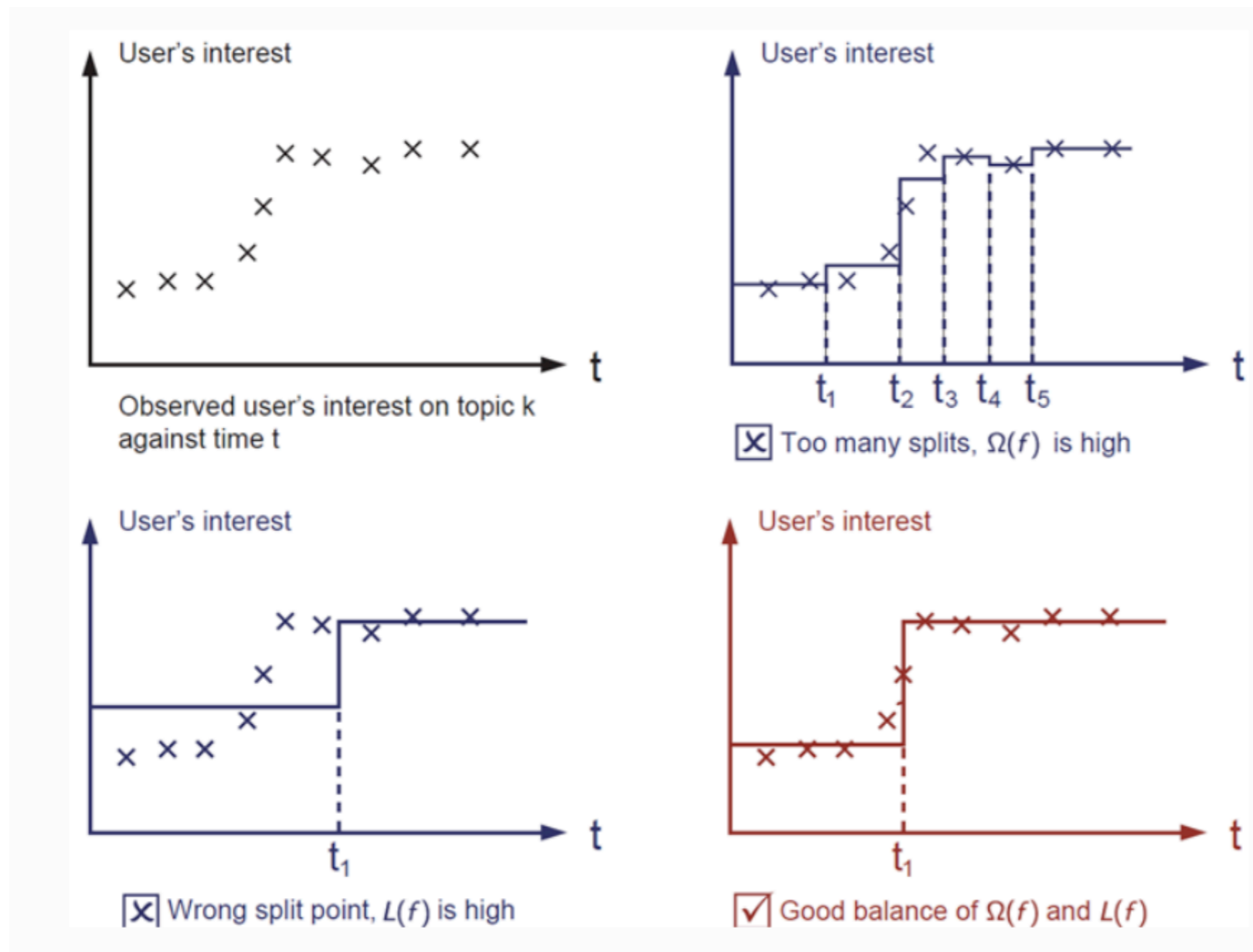
Boosting vs Bagging

- 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 the process is progressively iterated to get the 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.

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)



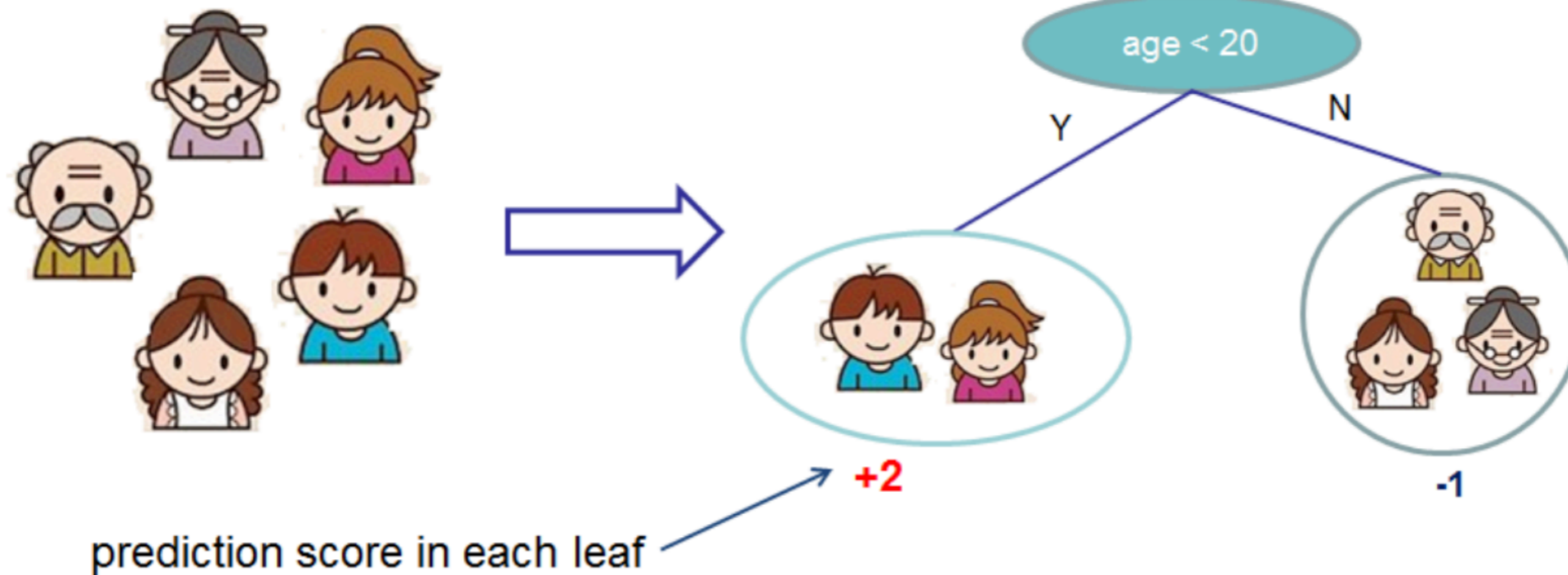
Example: function fitting with a tree, finding the optimal tree complexity

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.

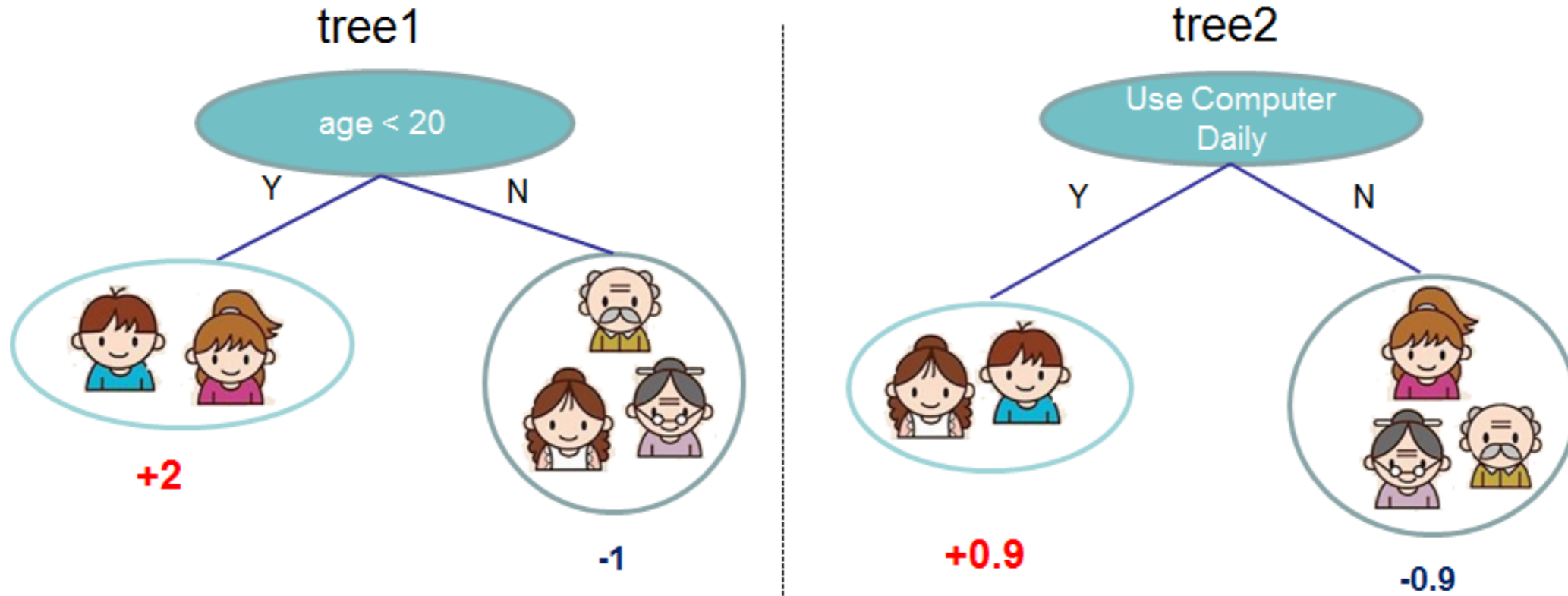
Input: age, gender, occupation, ...


Like the computer game X




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

no. 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

MSE Loss

example Losses:

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

$$L(\theta) = \sum_i [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})]$$

Logistic regression Loss

$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$

objective ↑ *training Loss* ↑ *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.
- 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.

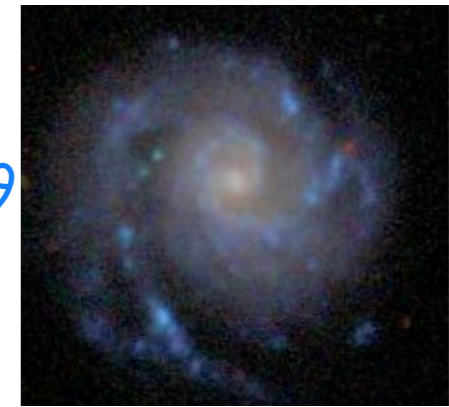
Decision tree ensembles

3. Application: Red Shift estimation

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

Redshift of a galaxy

distance $\propto z$ →



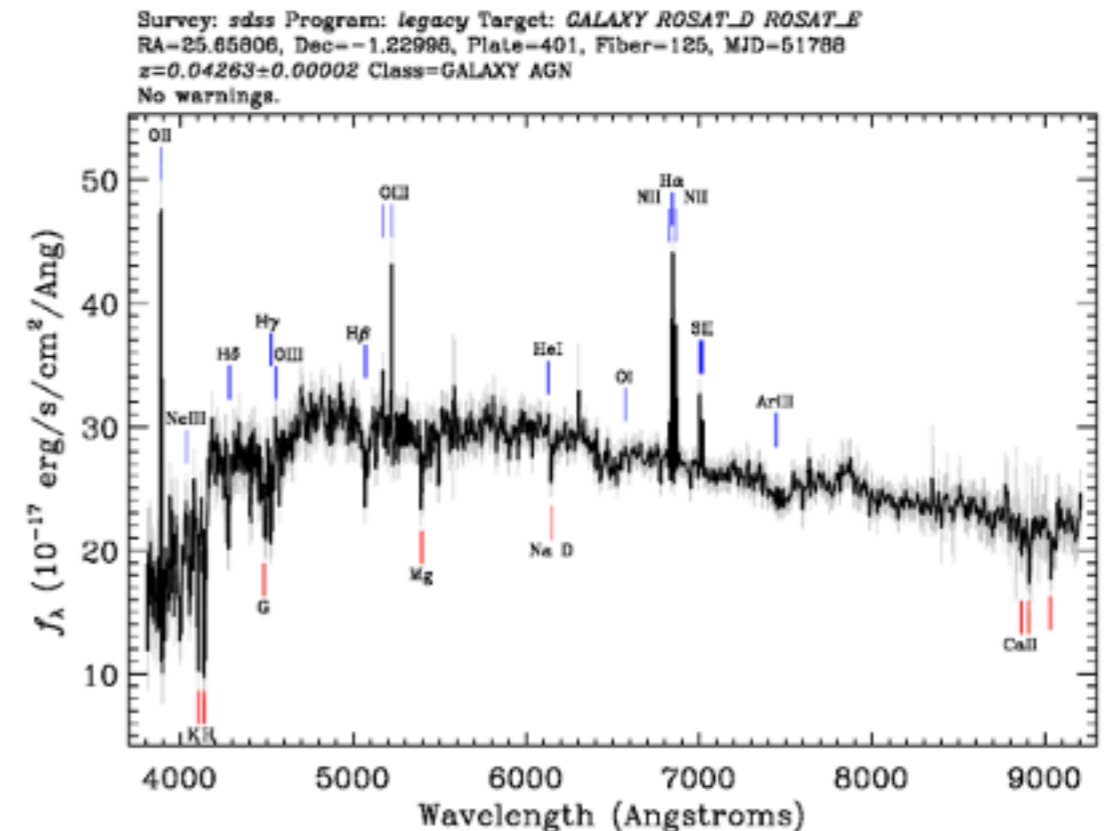
Astronomers measure the distance of a galaxy using its redshift. Because the universe is expanding, galaxies that are farther away have a higher redshift.

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 I can identify the emission lines I see (from the structure – one is not enough!), I 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

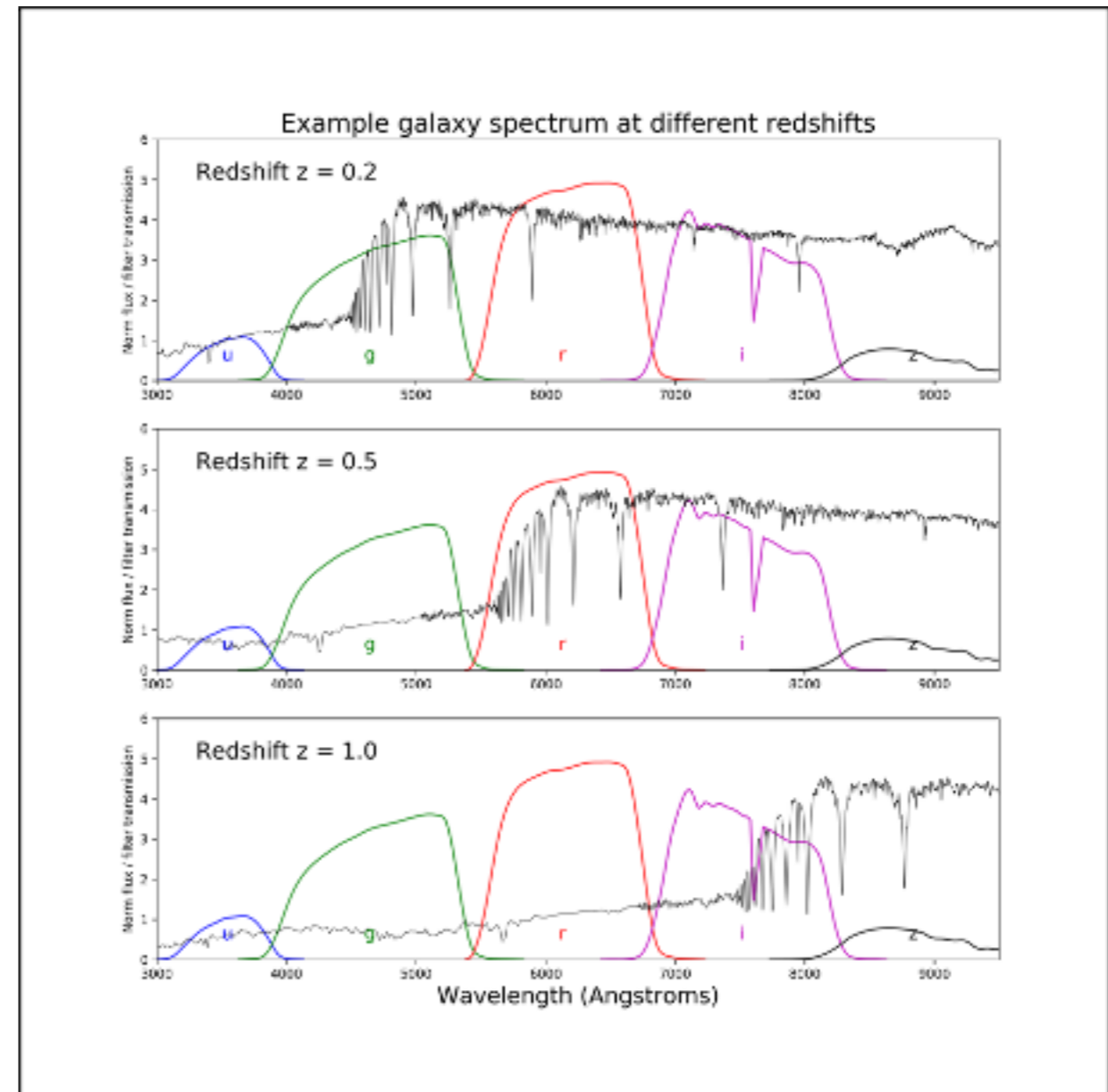
Photometric redshifts

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, 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.



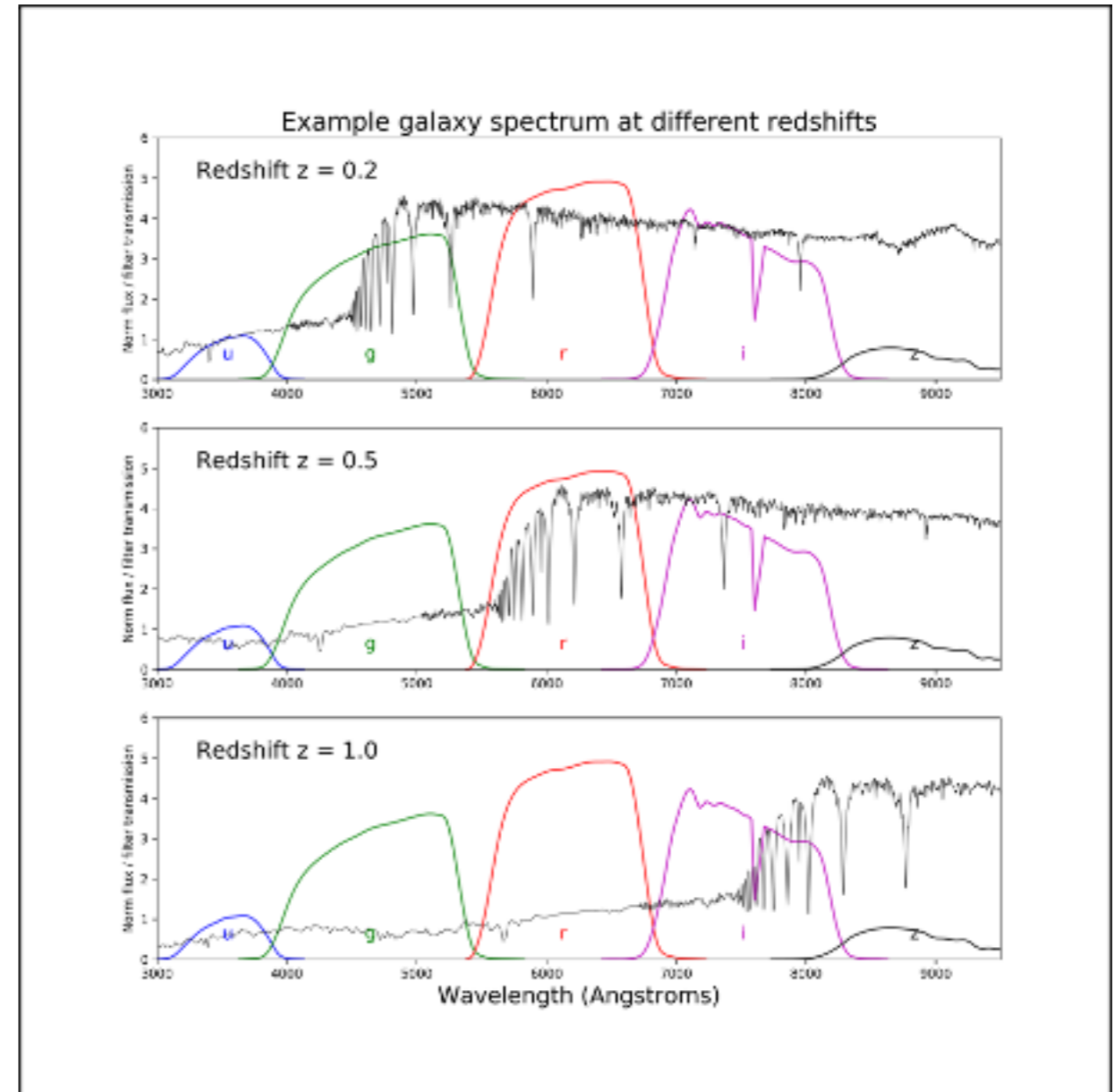
Learning task

Input data:

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

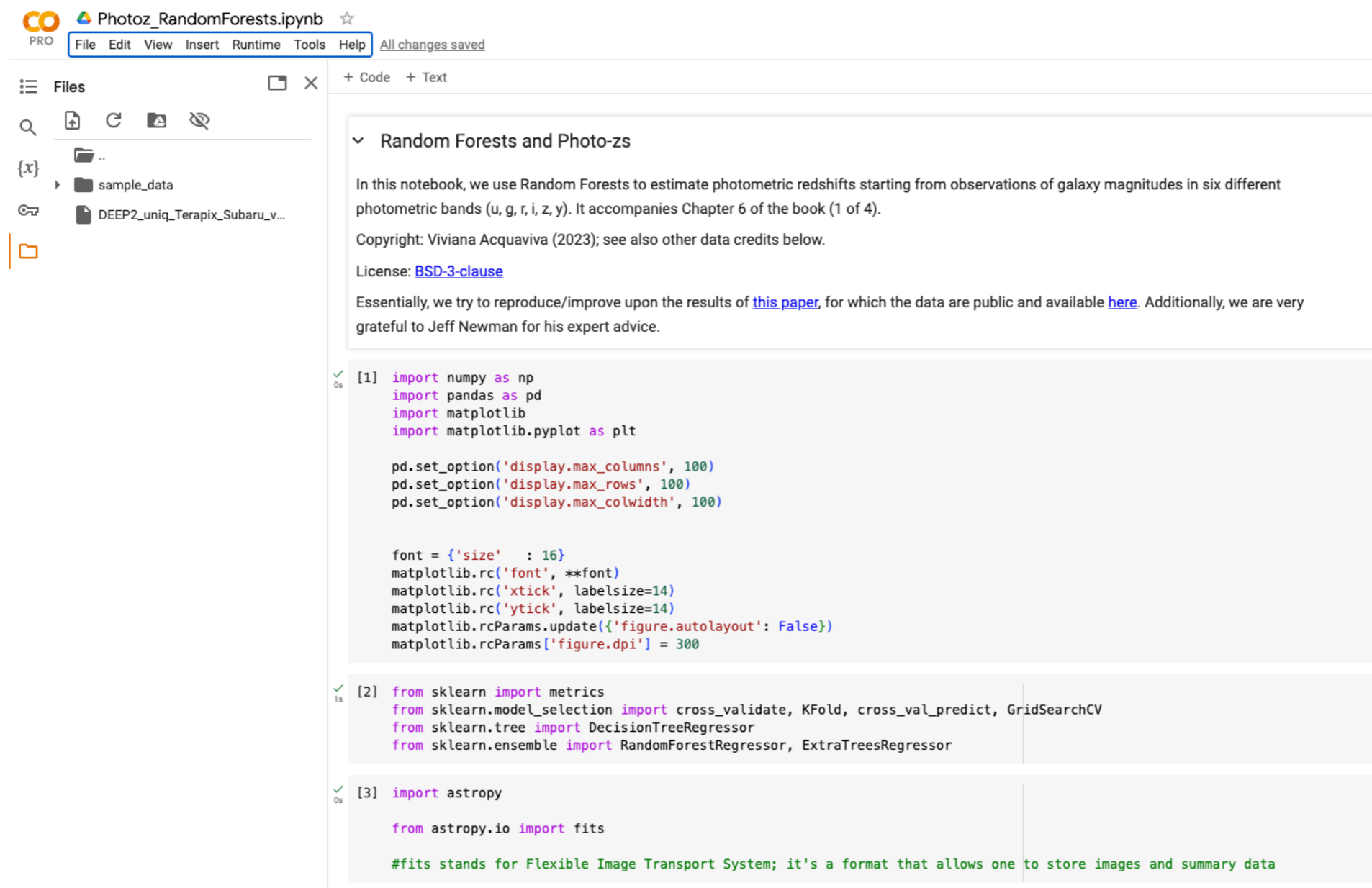
Target data:

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>.



Photoz_RandomForests.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

Files

- 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', labels=14)
matplotlib.rc('ytick', labels=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.