Applied Machine Learning in Engineering

Lecture 07, May 24, 2023

Prof. Merten Stender

Cyber-Physical Systems in Mechanical Engineering, Technische Universität Berlin

www.tu.berlin/cpsme
merten.stender@tu-berlin.de

X-Student Research Groups

- Research teams of 15 students (BUA) and young researchers
- Seminar (6 ECTS, free choice modules) for one semester (winter 23)

more info

Proposal for Research Group: **Physical Reservoir Computing**

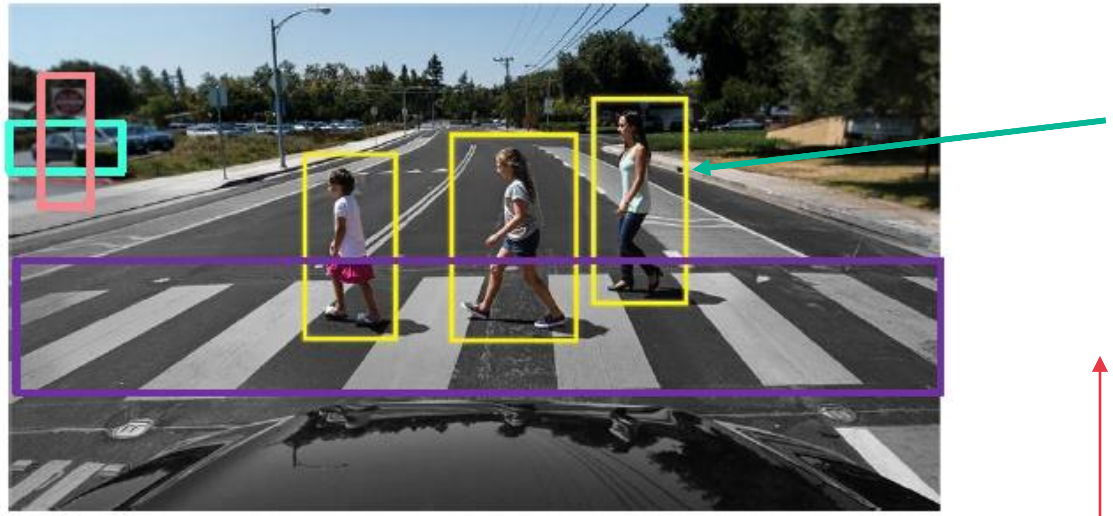
Use a bucket of water for building a machine learning computer

- **Build a demonstrator** (electronics, micro-controllers, computer vision, coding, ML)
- Show a **proof of concept** for time series prediction or natural language processing
- **Present results** at scientific conference or publish scientific paper

Spread the word! If interested: email to M. Stender!

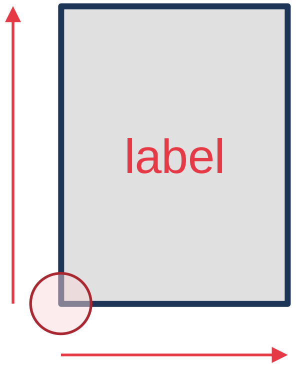
Recap: Lecture 06

- **Supervised learning**: the crucial role of high-quality labels. Example: labels for object detection



Bounding box

- Coordinates of lower-left corner
- Width & height
- Class label



Recap: Lecture 06

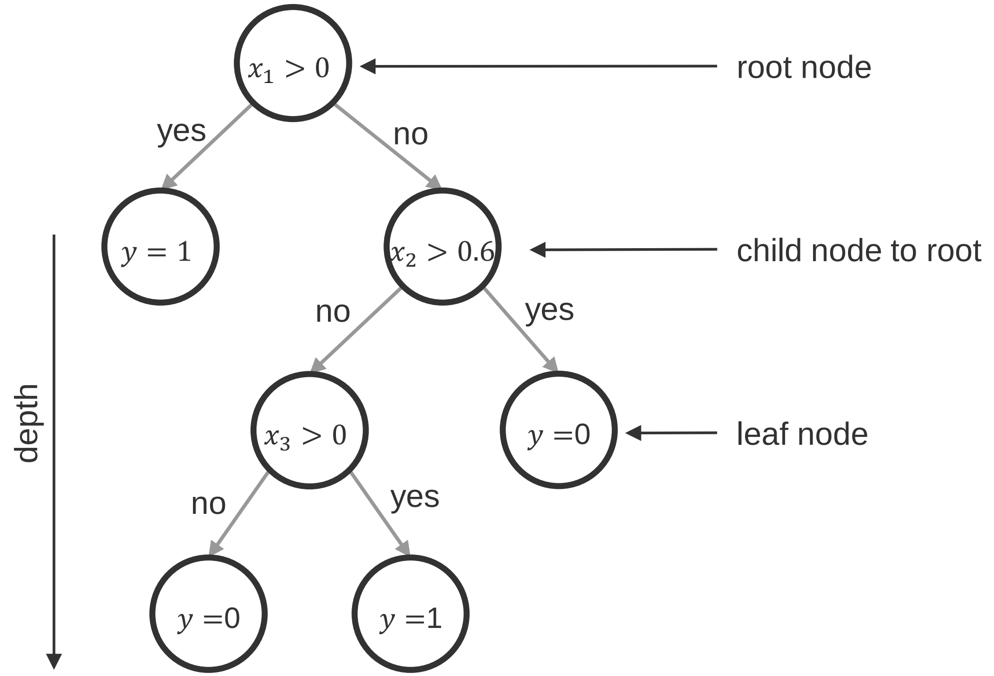
Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- **Decision trees: sequential binary feature space segmentation**

- Splitting of parent nodes to increase purity in child nodes
  - Measure of purity: **entropy** $H(x)$
  - Increased purity from parent to child nodes: **information gain** $I(x)$

$$H(x) = -\sum_{i \in C} P(x_i) \log P(x_i)$$

$$I(x) = H_{\mathrm{P}}(x) - \left(\frac{N_{\mathrm{L}}}{N_{\mathrm{P}}} \cdot H_{\mathrm{L}}(x) + \frac{N_{\mathrm{R}}}{N_{\mathrm{P}}} \cdot H_{\mathrm{R}}(x)\right)$$



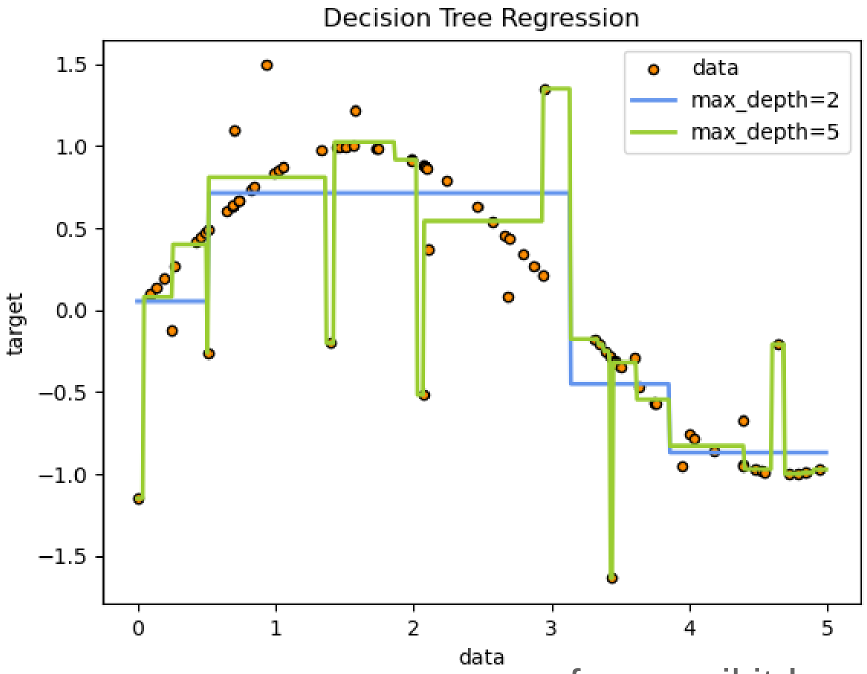
Recap: Lecture 06

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- **Stopping criteria** for avoiding overfitting
  - Max. tree depth
  - Min. samples per node

- **Regression trees**: binning of continuous variables

- Generally:
  - Decision trees are *prone to overfitting*
  - Issues arise as the number of features grows



from: scikit-learn

Today

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Understand different types of ensemble methods
- Differentiate bagging and boosting algorithms
- Explain the difference between bagging and random forests

Agenda

Machine Learning:

- Ensemble methods
- Random forests

Python:

- Recursive functions

Ensemble Methods

Ensemble Methods

Ensemble methods combine different weak learners into a larger meta-model that exhibits increased robustness against overfitting while obtaining more accurate predictions.

Why? Models can suffer the *curse of dimensionality* – as the number of feature dimensions grow, models get too complex and get prone to overfitting or suffer from high bias and underfit the data

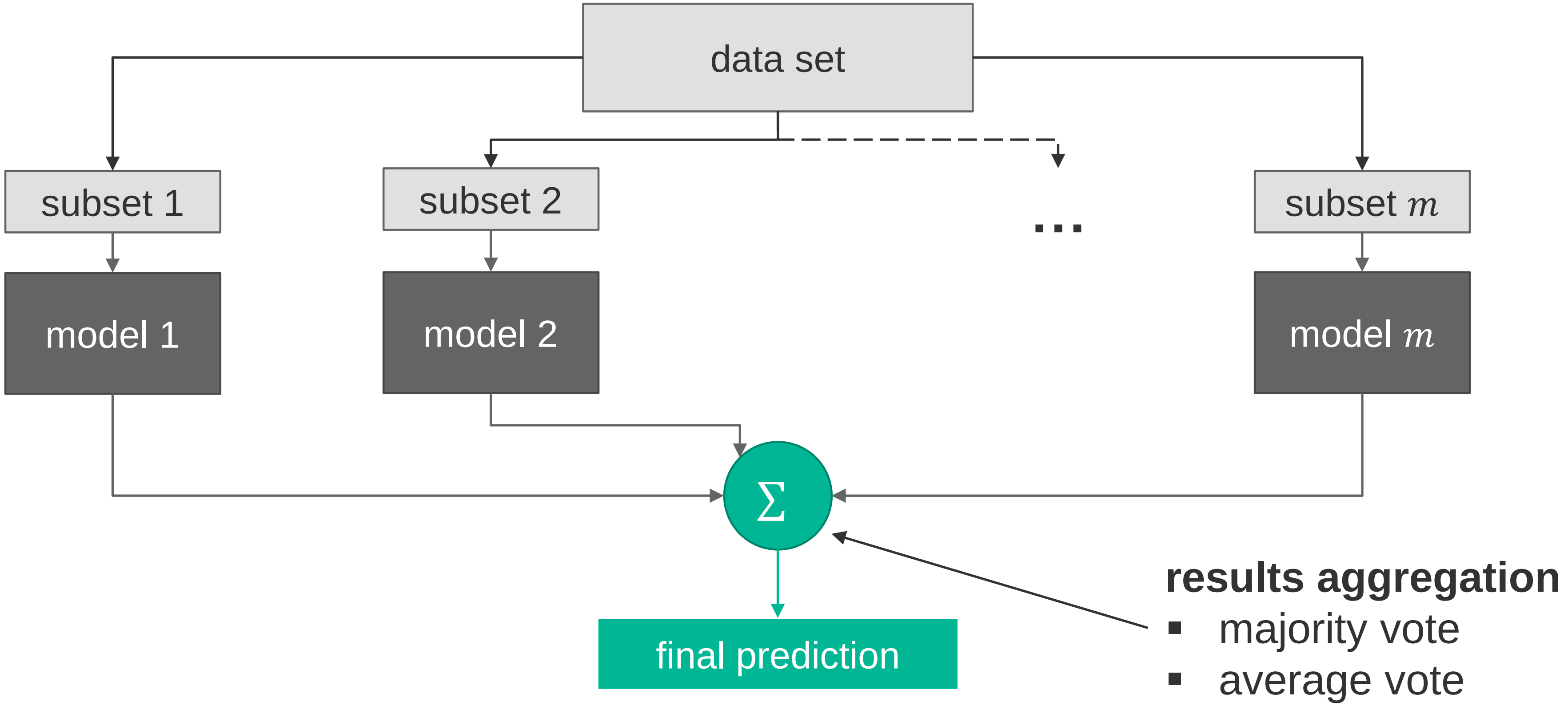
Two main approaches to build ensemble methods:

1. **Bagging** – **B**oostrap **Agg**regating: base models on bootstrapped data subsamples → avoid overfitting
2. **Boosting** – Increase the complexity of models where their performance is weak → avoid underfitting

Note: ensemble methods can be used for any kind of estimator, not only for decision trees!

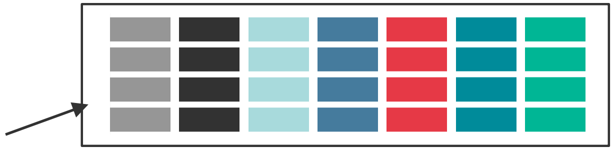
Bagging

- m different data subsets are obtained from bootstrapping and 1 model is formed on each subset
- Final prediction is obtained by aggregating the predictions of all m models (all of which independent of each other)



Data Sets and Sampling

- Data set contains N samples and n feature dimensions

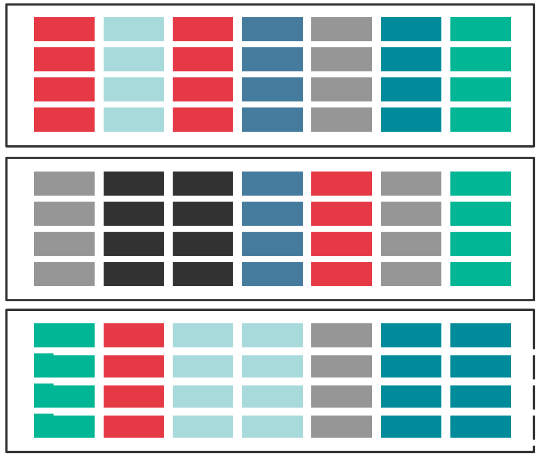


$n = 4$

$N = 7$, one color for each sample

- **Bootstrapping**: drawing N_B samples at random with replacement for m times

$N_B = 7, m = 3$



$N_B = 5, m = 3$

Random Forests

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

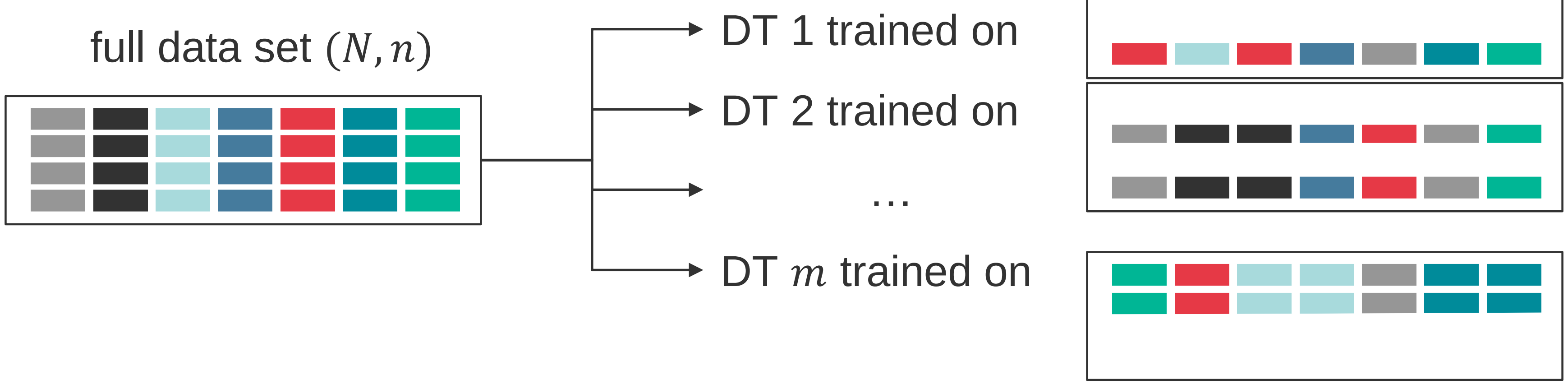
- Ensemble method for decision trees using a variant of bagging

- m decision tree classifiers are trained on m bootstrapped data subsamples of the complete data set using $N_B = N$ (as many samples in subsets as there are in the complete data set)

→ Bagged Trees

- Difference to classical bootstrapping: random choice of $n_B < n$ feature dimensions for each subset

→ Random Forest



$n_B = 2$

sklearn.ensemble.RandomForestClassifier

class `sklearn.ensemble.RandomForestClassifier`(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

bootstrap : *bool, default=True*

Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

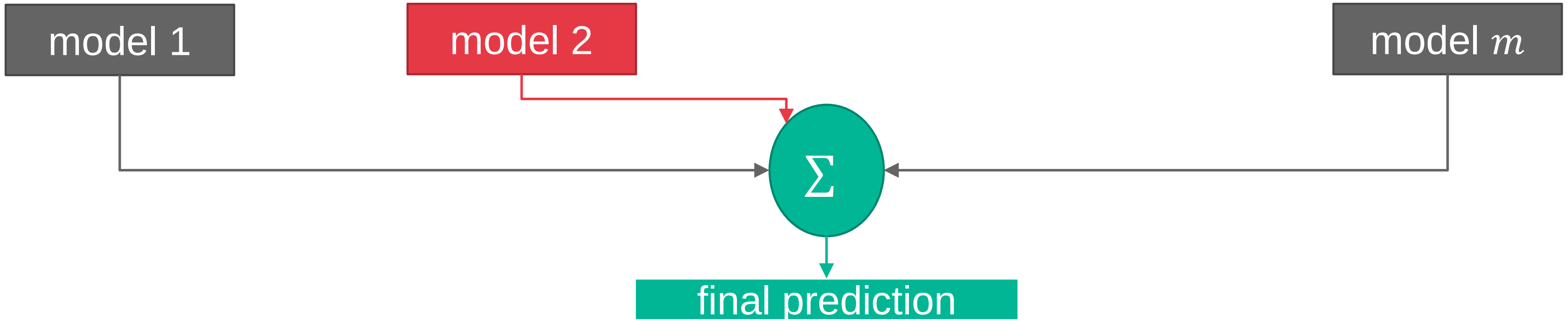
max_features : *{"sqrt", "log2", None}, int or float, default="sqrt"*

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `max(1, int(max_features * n_features_in_))` features are considered at each split.
- If "auto", then `max_features=sqrt(n_features)`.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

Boosted Trees

- Bagged Trees / decision trees cannot deal well with errors made by individual trees. Weak predictions will always enter the result aggregation

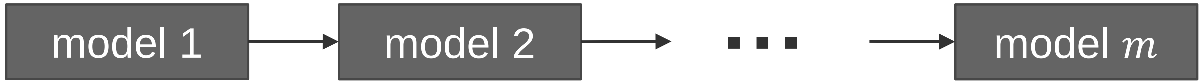


- Boosting: adaptive learning from mistakes and improving models where different base learners in the ensemble did not perform well

- Examples: XGBoost

Boosting

- **Boosting is used to incrementally increase complexity** of a model in regions of the data set where the model‘s performance is poor, i.e. performing adaptive learning
- Boosting is based **assigning weights to individual data points**, and increasing the weights of data points that were misclassified by the model (adaptive boosting ***AdaBoost***)
- Very simple classifiers (so-called **weak learners**) are combined into an ensemble by **sequential learnin**g: the current weak learner is dependent on the previous learner

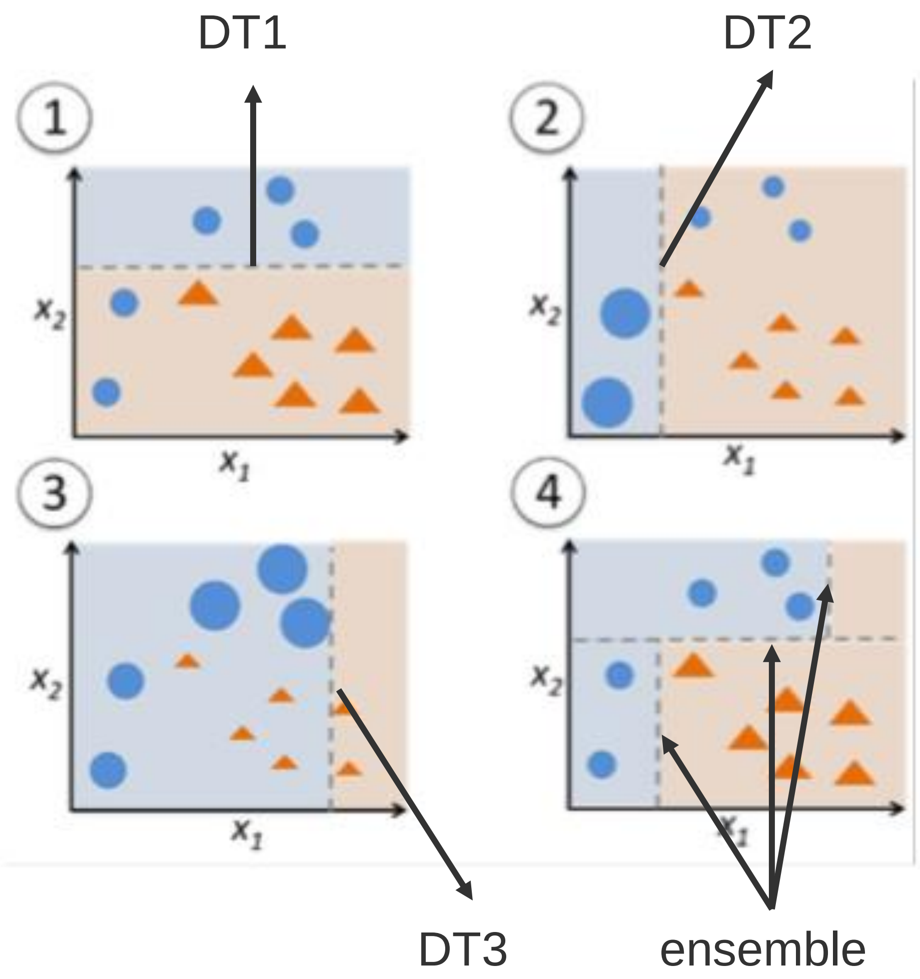


Boosted Trees: AdaBoost

- Weak learners: decision tree (DT) with only 1 splitting rule (two leaf nodes)
  decision tree stump model

1. Build first tree, find data set regime where is performs weak
2. Put larger weight on mis-classified data points
3. Build second tree using weighted data set

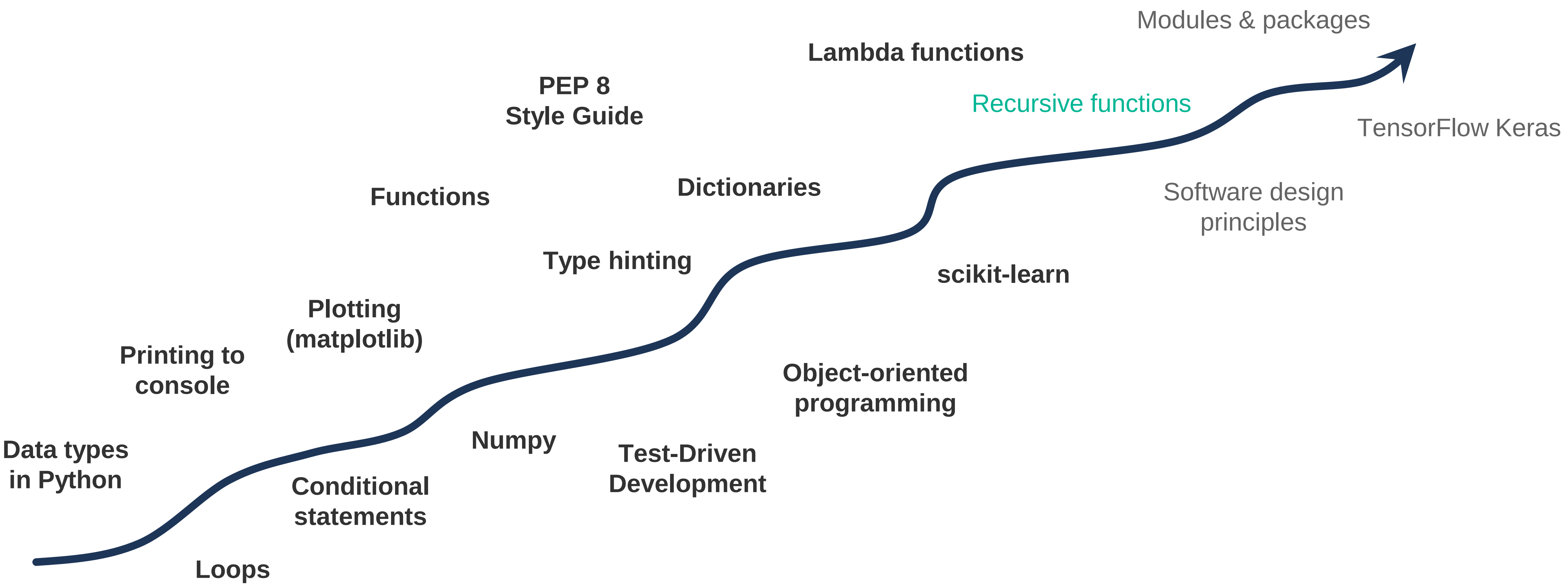
- Continue (without weight reduction)



Python

Learning Curve

Cyber-Physical Systems
in Mechanical Engineering TU Berlin



Recursive Functions

- Functions can call different functions
- Recursive functions call themselves

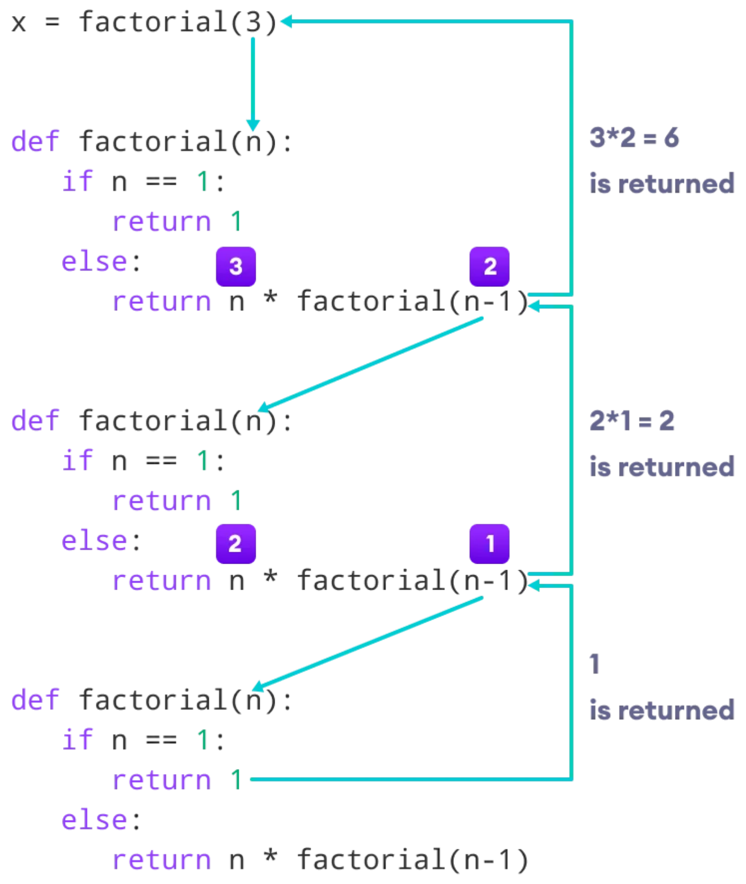
- Example: finding the factorial of a number, e.g. $3! = 1 * 2 * 3 = 6$

```
def factorial(x: int) -> int:
    """Compute the factorial of an
    integer using a recursive function"""
    if x == 1:
        return 1
    else:
    return x * (factorial(x-1))


print(f'factorial of x=3 is {factorial(3)}')
```

stopping condition (‚base condition‘)

Recursive Functions

```python
def factorial(x: int) -> int:
    """Compute the factorial of an
    integer using a recursive function"""
    if x == 1:
        return 1
    else:
    return x * (factorial(x-1))


print(f'factorial of x=3 is {factorial(3)}')
```



From https://www.programiz.com/python-programming/recursion

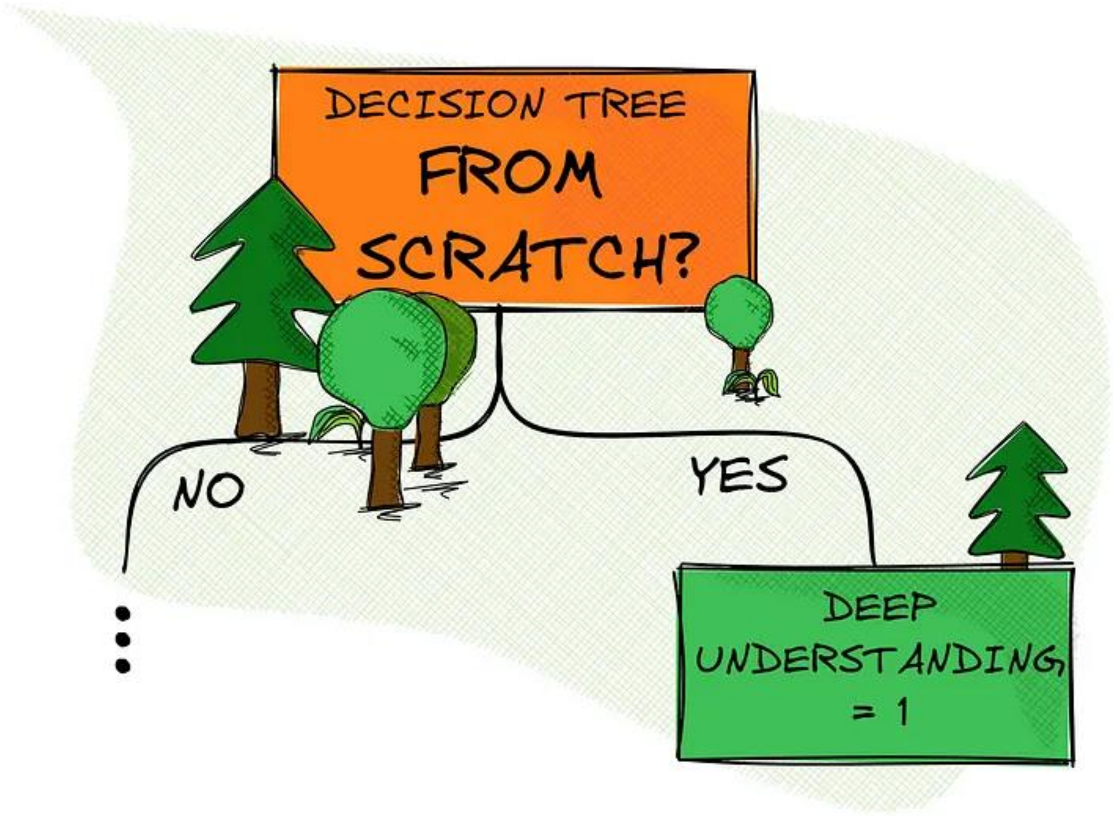
Exercise 07

May 31, 2023

Exercise 07

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Implementation of a decision tree from scratch
- Implement a recursive function to grow the tree as long as no stopping condition is met
- Implement a method for traversing data through the final tree
- Test implementation on sample data set



Questions?

Prof. M. Stender | merten.stender@tu-berlin.de | tu.berlin/cpsme