

HFT Summer School 2025

Fundamentals of Machine Learning

Michael Mommert

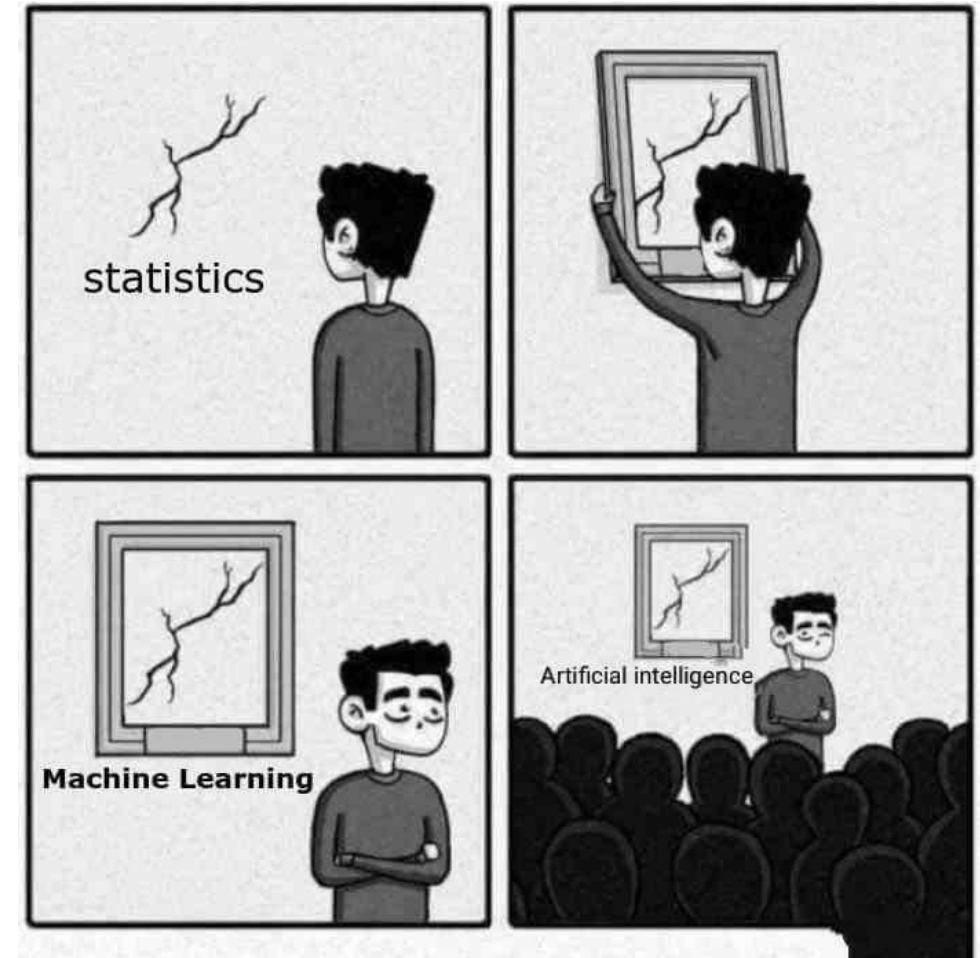
Stuttgart University of Applied Sciences

Learning Objectives

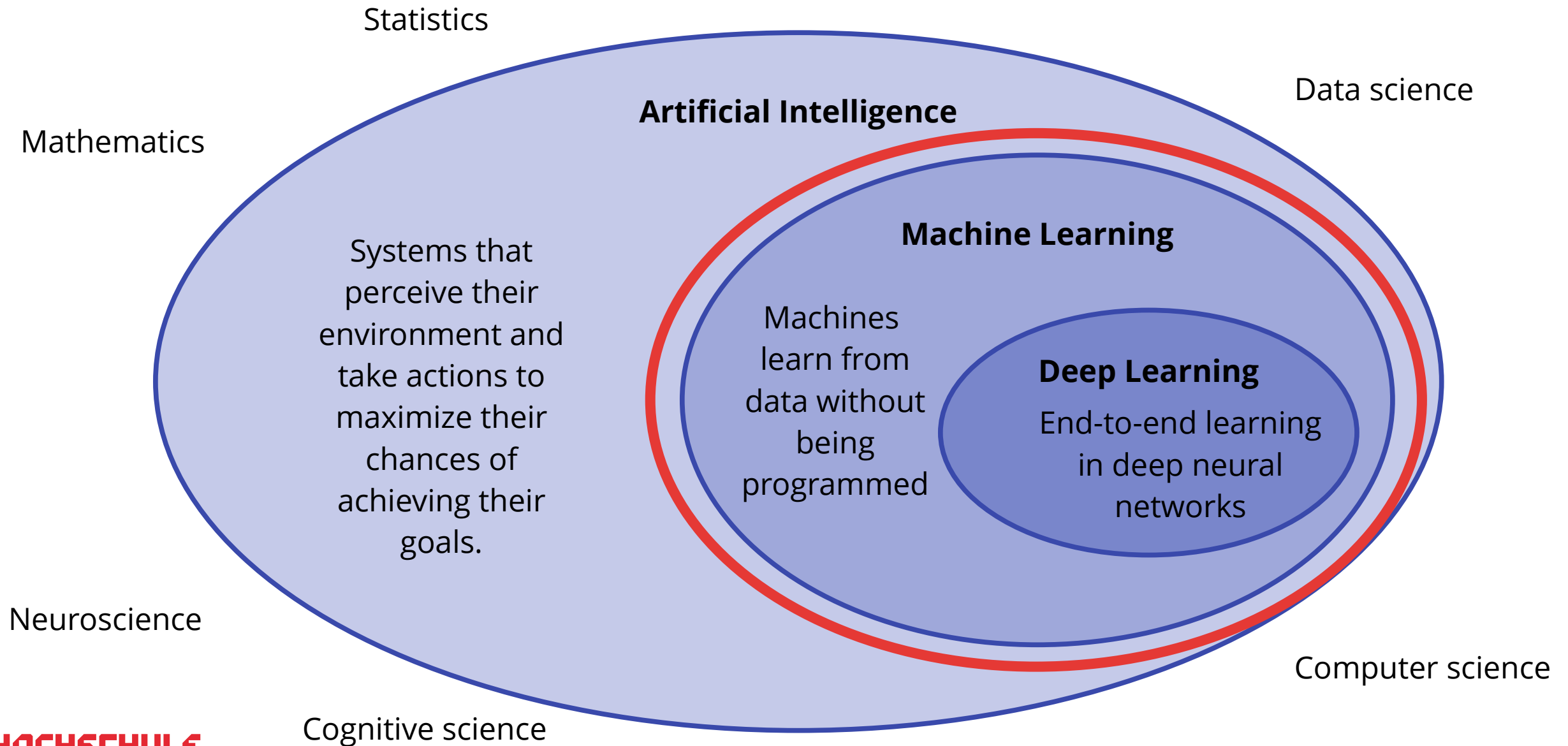
In this lecture you will learn the following:

- What is machine learning?
- What is supervised learning?
- What is feature engineering?
- What is overfitting, underfitting and generalization?
- How to benchmark machine learning models?
- How to use a linear regression model?
- How to use a k-Nearest Neighbor classifier?

What is Machine Learning?



Mapping terminology



What is Machine Learning (ML)?

"The field of study that gives computers the ability to learn without being explicitly programmed."
- Arthur Samuel (1959)

Different approaches:

- **Supervised learning**

Find a function that relates input data to output data by learning a specific task.

- **Unsupervised learning**

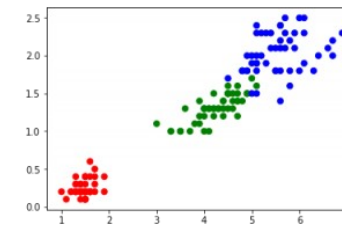
Find structure within a data set.

- **Reinforcement learning**

Learn a task in a dynamic and responsive environment.



Iris Versicolor



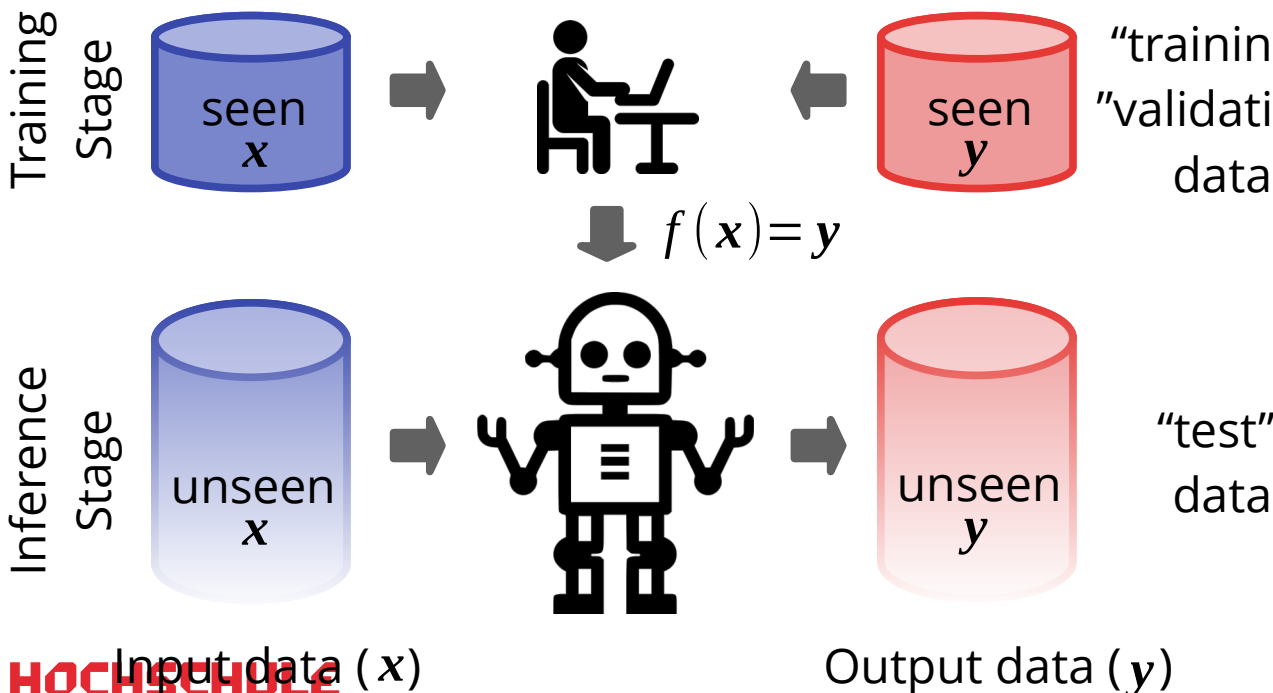
Supervised Machine Learning

General goal for supervised problems:

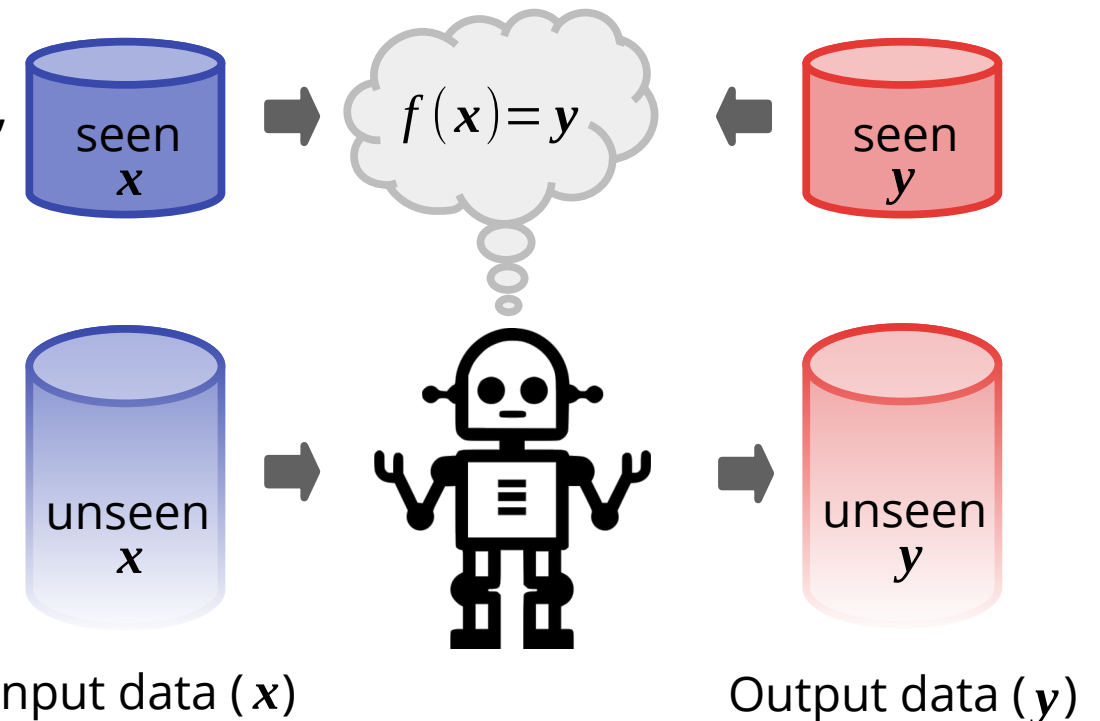
Find a function ("task") that relates input data (x) to output data (y) such that: $f(x) = y$

This function has to be "trained" before it can be applied to previously unseen data ("inference").

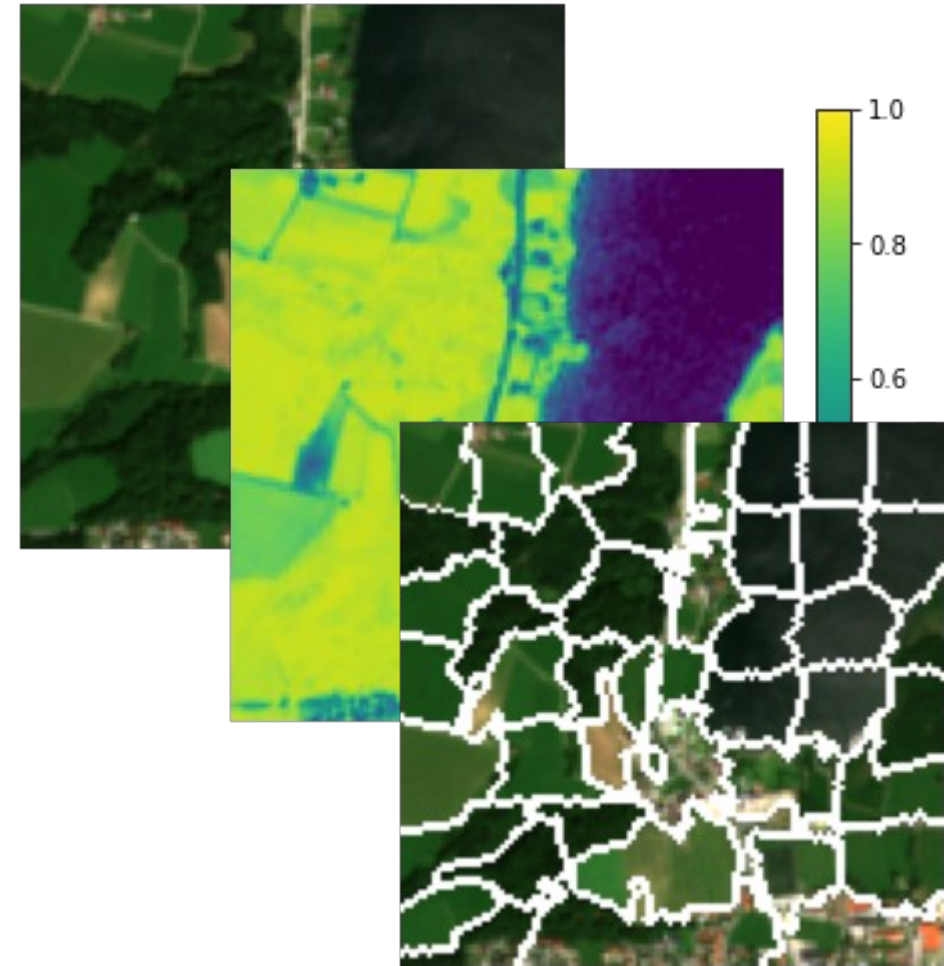
Traditional (Rule-based) Approach:



Machine-Learning Approach:



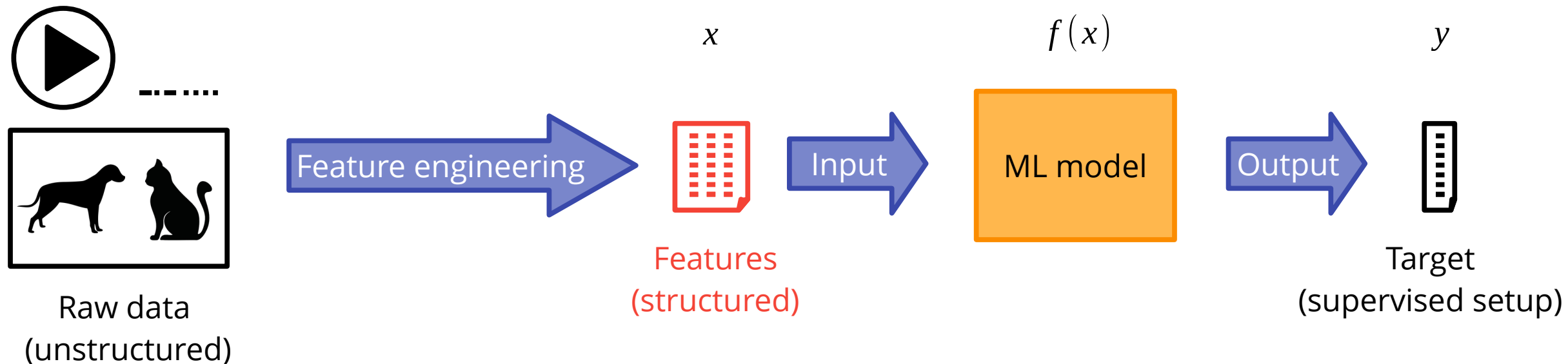
Data and Features



Data for Machine Learning

A wide range of data modalities exists.

Not all data modalities can be readily used in Machine Learning models: most models require **numerical features** (quantitative data, see next slide) as input. This means that suitable and meaningful features have to be extracted from the data modality.



Structured vs unstructured data

Structured data

Preprocessed and formatted data that is easily queryable.

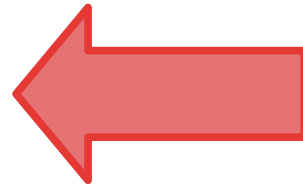
Quantitative data



Only continuous/
discrete numbers:
"features"

Each sample
represented as a
feature vector

Processing unstructured
data into structured data
is called **feature
extraction** or **feature
engineering**



Unstructured data

Unprocessed and unformatted data is not easily queryable.

Qualitative
data

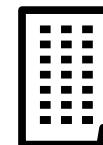
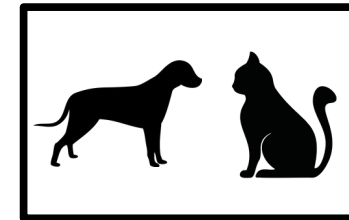


Image data



Video data



Textual data



Data complexity

Data stream

.....

Audio data

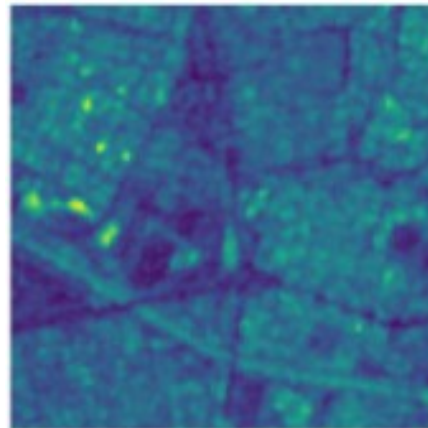


Image data

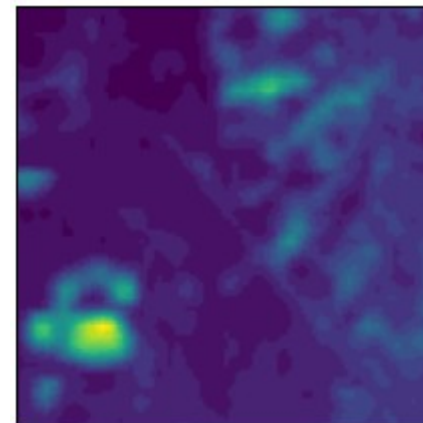
Images or **image-like data representations** are very common as model input. Image-like data are rasterized (projected to a regular); rasterization happens at the time of acquisition (e.g., as part of the imaging process) or synthetically (e.g., by projecting GIS polygons on a grid).



Multispectral Imaging
(e.g., Sentinel-2)



SAR Polarization
(e.g., Sentinel-1)

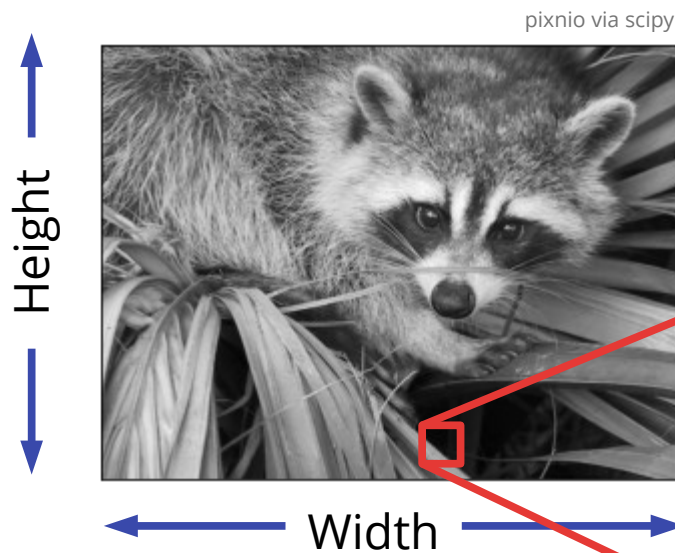


Digital Elevation Model
(e.g., GTOPO30)

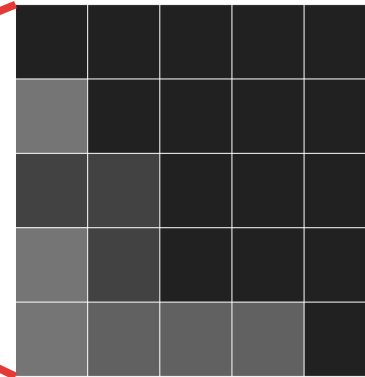


Land Use/Land Cover
(e.g., ESAWorldcover)

How are images represented?



Images consist of quadratic “Picture elements”, **pixels**. An image of height H and width W contains $H \times W$ pixels.



In a **greyscale** image, the brightness of each pixel is represented by a single value $[0, 1]$ (8-bit encoding: $[0, 256]$): 0 refers to a black pixel and the max value to a white pixel.

0	0	0	0	0
0.3	0	0	0	0
0.1	0.1	0	0	0
0.3	0.1	0	0	0
0.3	0.2	0.2	0.2	0

Remote sensing data may use a higher dynamic range: e.g., Sentinel-2 MSI: 16bit ($[0, 65536]$)

How are images represented?

Greyscale



pixnio via scipy

Color (RGB)



```
[7] image.shape  
0s  
"2-d" (600, 400)
```

```
[5] image.shape  
0s  
"3-d" (600, 400, 3)
```

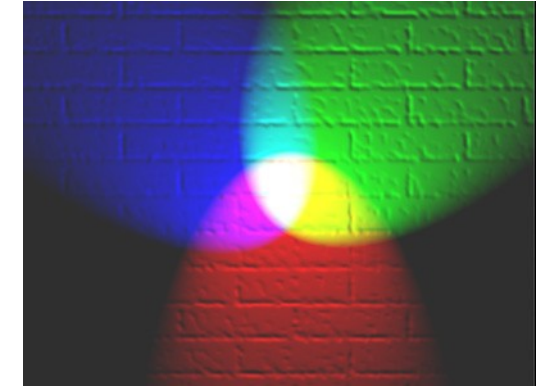


=



Why RGB?

Every color can be synthesized through additive mixing of red, green and blue.



Bb3cxv @ wikipedia

Color images typically consist of **three channels** (RGB), each of which is a grayscale image in itself.

More on color spaces later in this course...

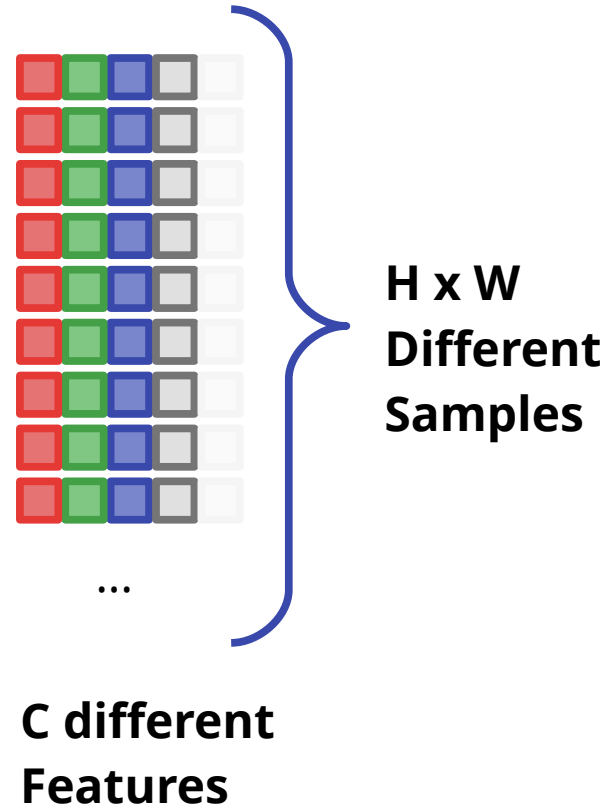
Feature engineering – image data

How can we feed image data into ML models?



[86, 183, 106, ...]

Number of bands: C



Each pixel is represented by its channel-wise values.

Very common in remote sensing applications (e.g., multiband, hyperspectral data).

Downside: each pixel is treated separately (neighborhood is ignored).

Feature engineering – image data

An **object-based image analysis approach** finds “objects” in the image data. Objects are image areas that share common properties, such as spectral properties, size, shape or texture. Unsupervised methods such as SLIC find such objects (“superpixels”) using a segmentation approach.



Properties derived from these “superpixels” can now be fed into a Machine Learning model as features.

Other features:

- Edges
- Lines
- Keypoints (SIFT)

Final data set nomenclature

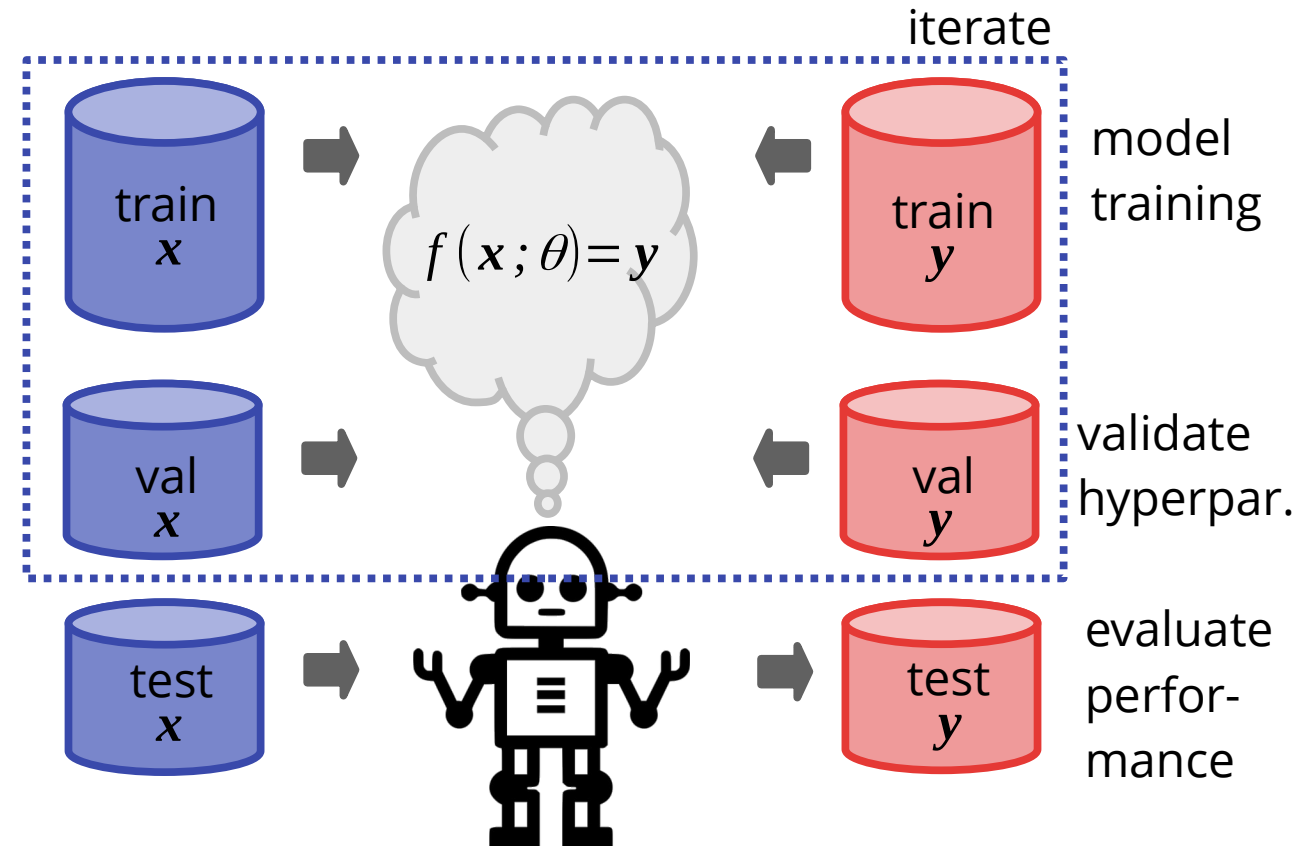
Feature engineering results in a compilation of features that we can use to train our ML models.

Remote Sensing-related Example: Land-use/land-cover classification

Features/Attributes (input variables, x)					$f(x) = y$	Targets/Labels (output variables, y)	
						Ground-Truth	
Samples/Instances = Image Pixels	R	G	B	NIR	NDVI	LULC Class	
	0.12	0.16	0.83	0.21	0.03	water	} classes of label "LULC Class"
	0.23	0.18	0.76	0.18	0.07	water	
	0.51	0.73	0.48	0.81	0.93	forest	
	0.43	0.76	0.27	0.76	0.83	forest	
	0.73	0.68	0.61	0.23	0.01	built-up	
	...	...	...	...	...	...	

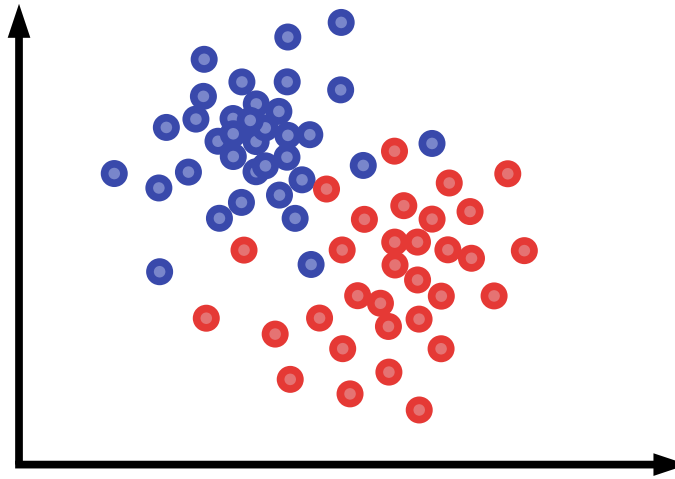
(This is a multi-class classification problem)

Model Training



Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:

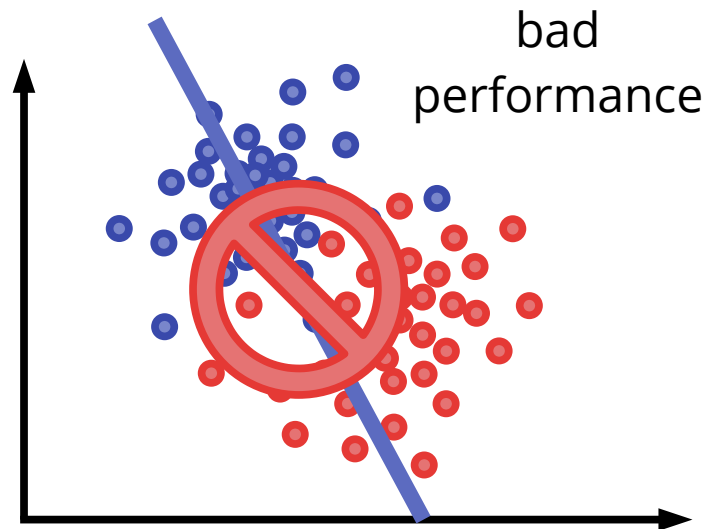


What would be a good decision boundary between the two classes?

The **decision boundary** separates the different classes as learned by the trained model.

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:

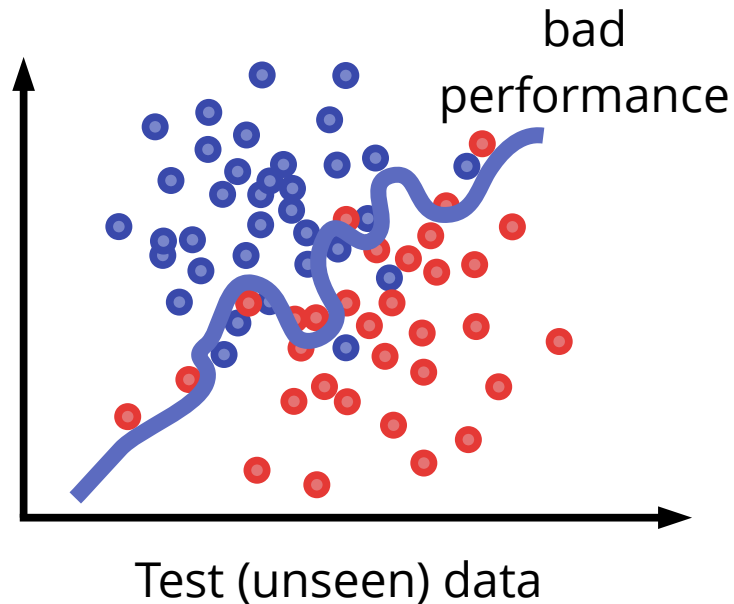


This would not be a good decision boundary as it barely allows to distinguish the two classes.

This model clearly **underfits** the data and leads to **poor performance**.

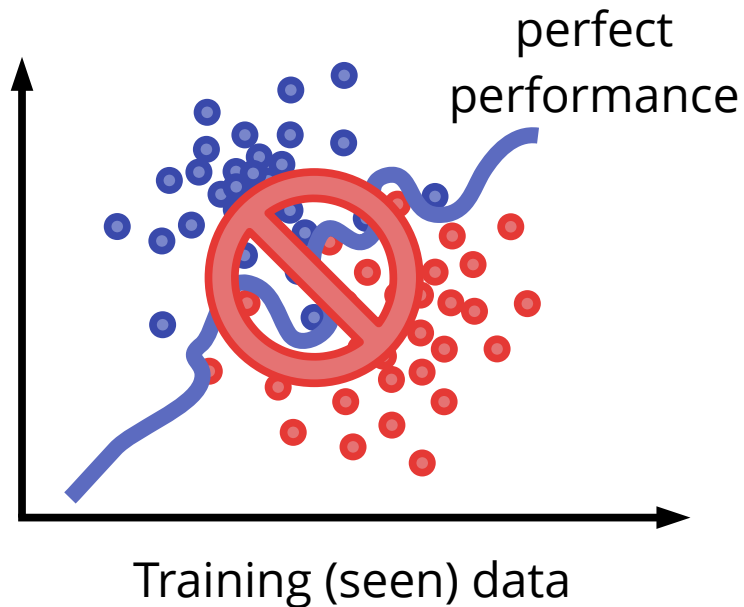
Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:



This model is **overfitting**: it memorizes the structure of the training (seen) data and as a result **generalizes badly** on the overall data distribution.

We can improve its performance through **regularization** methods.



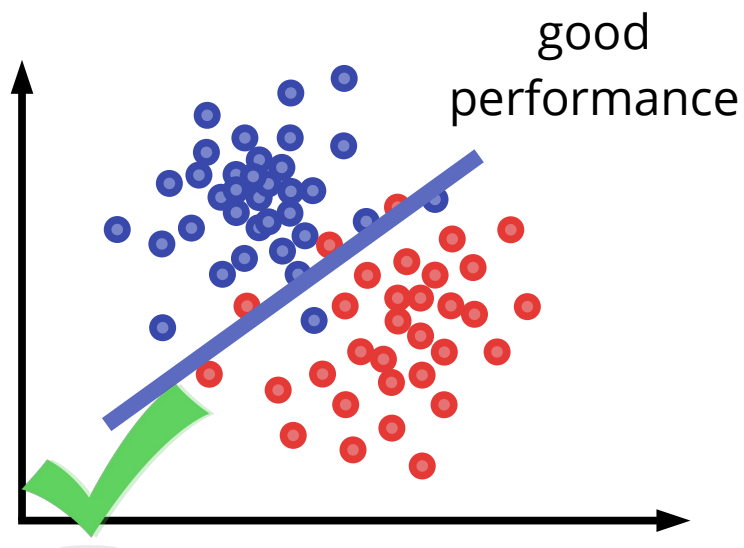
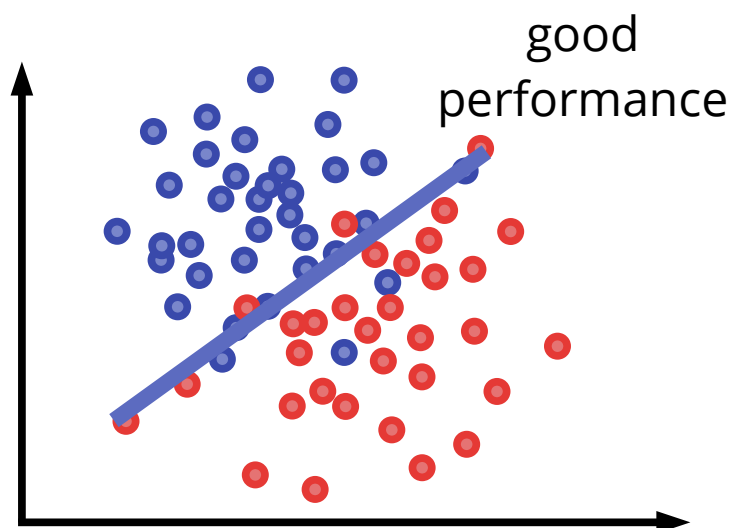
This decision boundary perfectly delineates the two classes in our data set.

Is this good?

No, see what happens if we apply the model to previously unseen data: while it seems to perform perfectly on the seen data, it performs much worse on the unseen data.

Generalization, regularization, overfitting and underfitting

Consider the following data set on which we train a ML model to classify two distinct class:



This decision boundary leads to **equally good performance on both data sets**.

The model therefore **generalizes well** on previously unseen data.

This is the desired situation as it provides a realistic view on the expected performance of your model.

How can we check how well the model generalizes?

How can we force the model to generalize?

Performance metrics

In order to measure how well our model generalizes, we have to define a **performance metric**. A performance metric provides a quantitative assessment of how well our model performs based on:

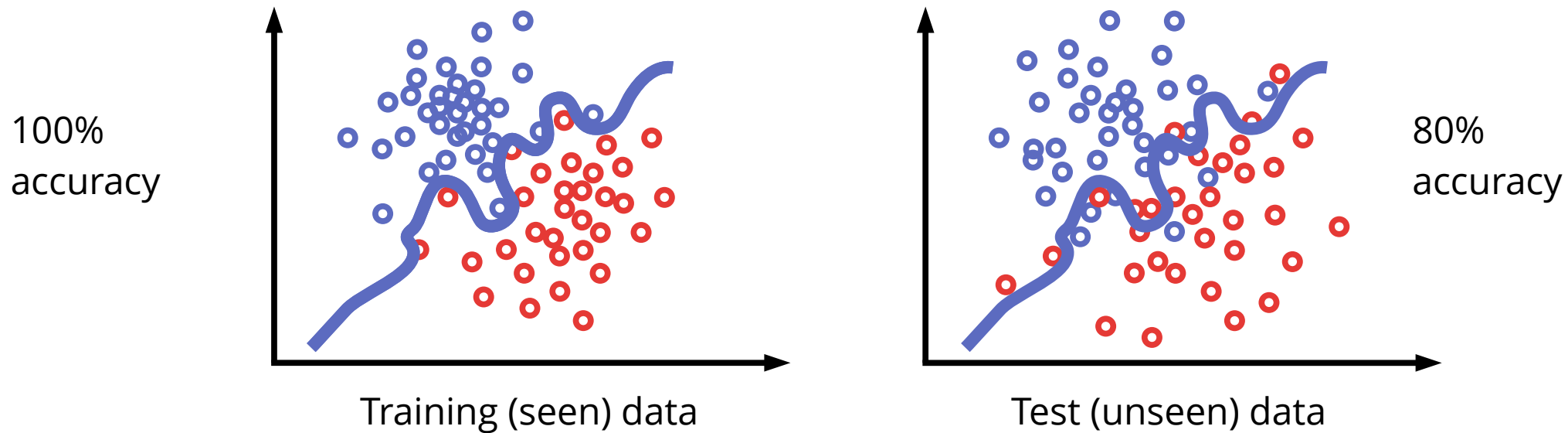
- the machine learning implementation (model type and loss function),
- the dataset,
- the task to be learned.

Performance metrics have to be tailored to the method and task. We will discuss them in detail in the final part of this lecture (Evaluation).

Assuming that we have such a performance metric implemented, how can we identify overfitting?

Identifying overfitting

Overfitting occurs if the model memorizes patterns specific to the training data instead of learning the general patterns of the overall dataset.



We can identify overfitting by comparing the performance on the seen (training) data and some previously unseen (test) data. But **where do we get unseen data from?**

Train/Validation/Test splits

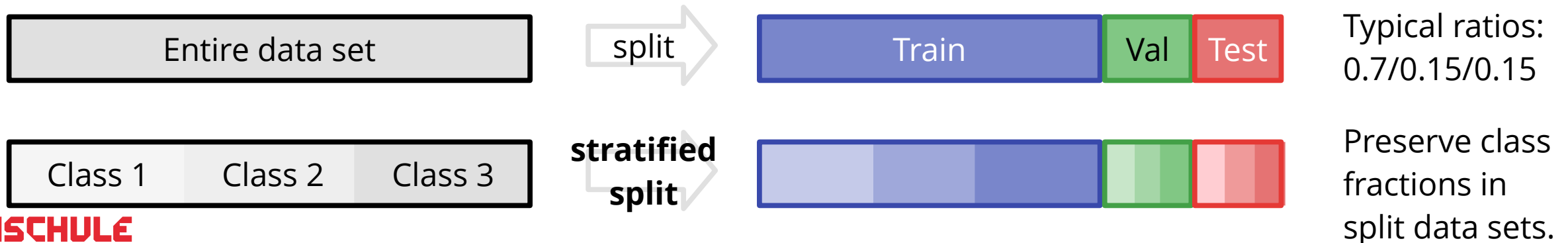
How can we check how well the model generalizes? We synthesize our own unseen data set.

In supervised learning, an existing data set is typically randomly split into three parts:

Training data set – the model is trained exclusively on this data set

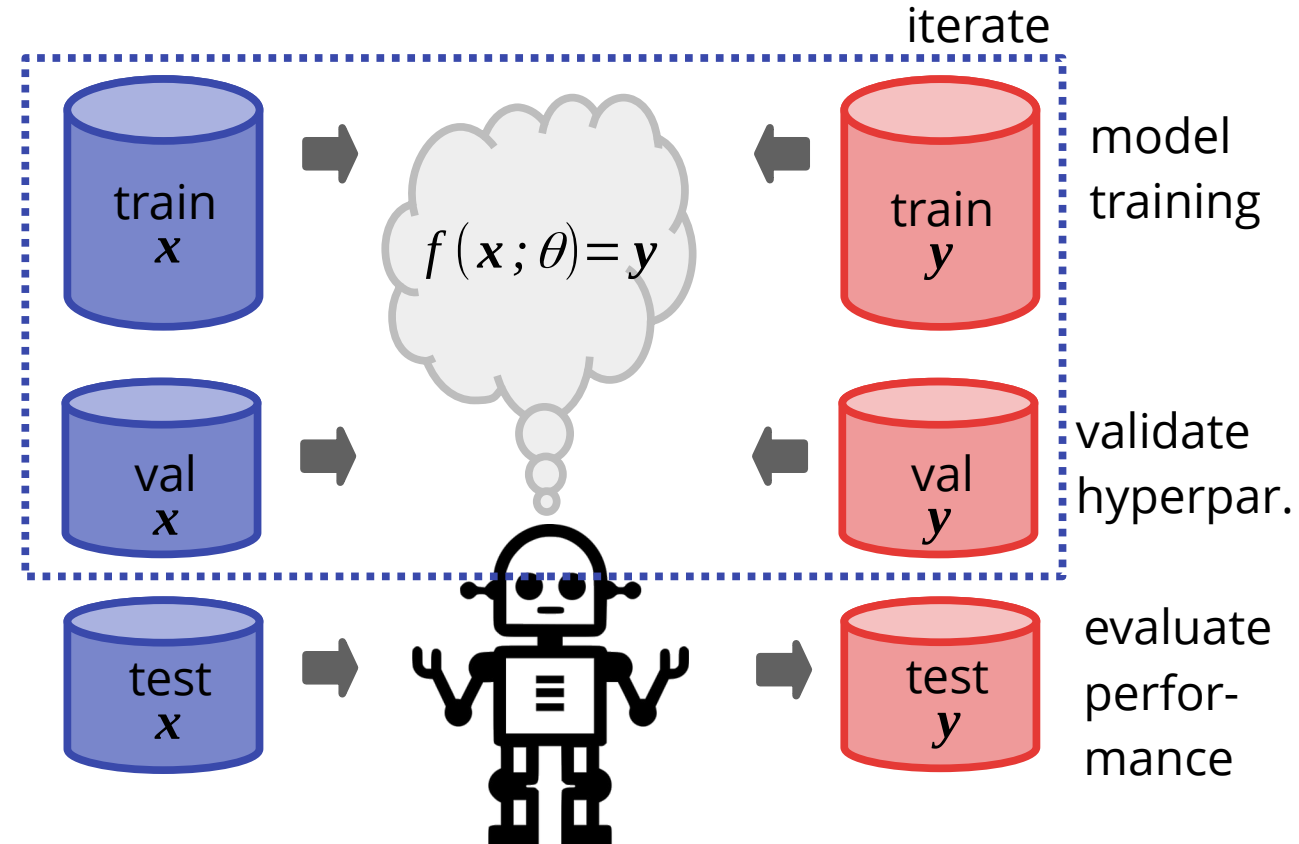
Validation data set – this data set is used to tune our model's hyperparameter

Test data set – this data set is used to evaluate the model's performance on unseen data

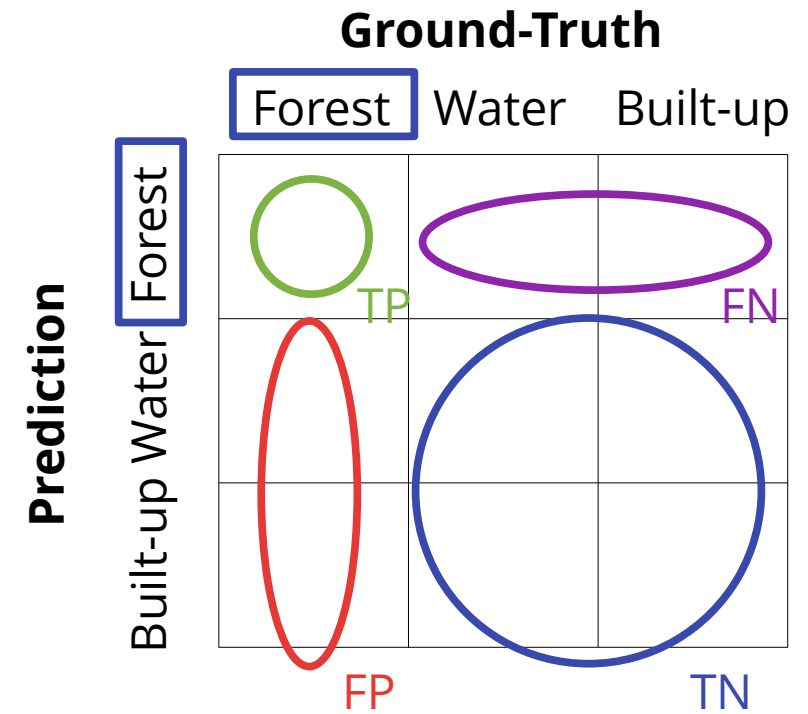


General supervised learning pipeline

- 1) Feature engineering: raw data → features
- 2) (Data scaling)
- 3) Data splitting → training, validation, test data
- 4) Define hyperparameters
- 5) Train model on training data for fixed hyperparameters
- 6) Evaluate model on validation data
- 7) Repeat 4) to 6) until performance on validation data maximized
- 8) Evaluate trained model on test data
→ report test data performance



Model Evaluation



In order to know whether our model training was successful or not, we have to evaluate it.

Performance is measured based on pre-defined metrics and the available ground-truth data; a **metric** can be thought of as a measure for how well an ML model performs on a specific task and data set.

For a meaningful evaluation of the performance of your model, two conditions must be fulfilled:

- the model must be evaluated on a metric **suitable** to the chosen task and
- the model must be evaluated based on an independent **test** dataset (not the training dataset!)

Only if both conditions are met, the reported results mirror the performance of the model on previously unseen data.

Regression

Regression task metrics:

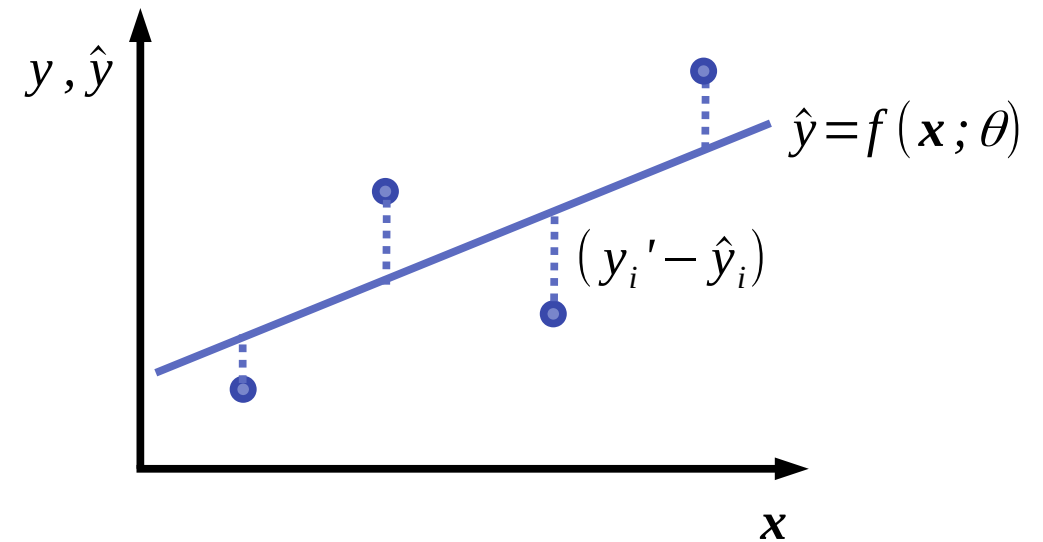
MAE (Mean Absolute Error)

$$MAE = \frac{1}{N} \sum_i^N |y_i' - \hat{y}_i|$$

RMSE (Root Mean Square Error)

$$RMSE = \sqrt{\frac{1}{N} \sum_i^N (y_i' - \hat{y}_i)^2}$$

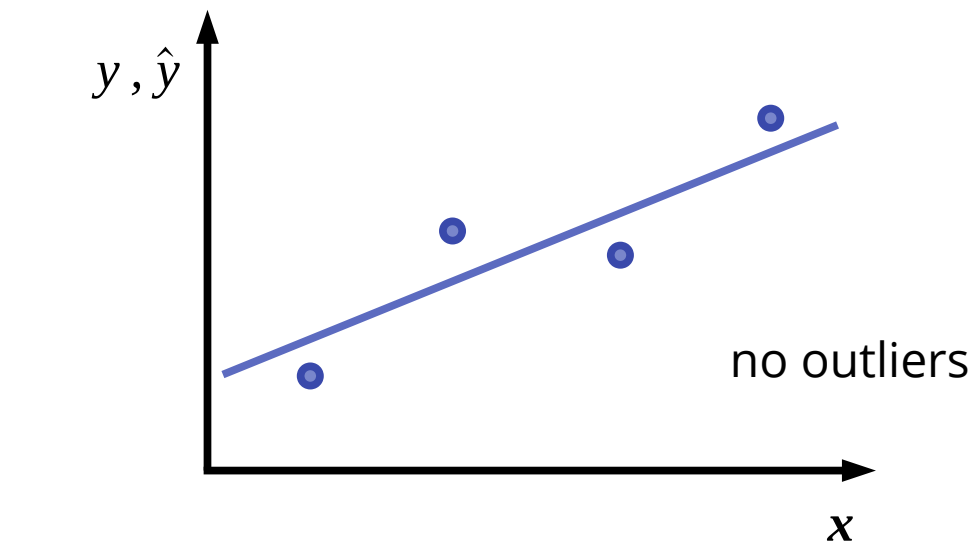
Input data: $\mathbf{x}_i, i \in \{1 \dots N\}$
Target ground-truth: y_i'
Target prediction: $\hat{y}_i = f(\mathbf{x}_i; \theta)$



Intuition: by how much deviates your model prediction from the ground-truth on average.

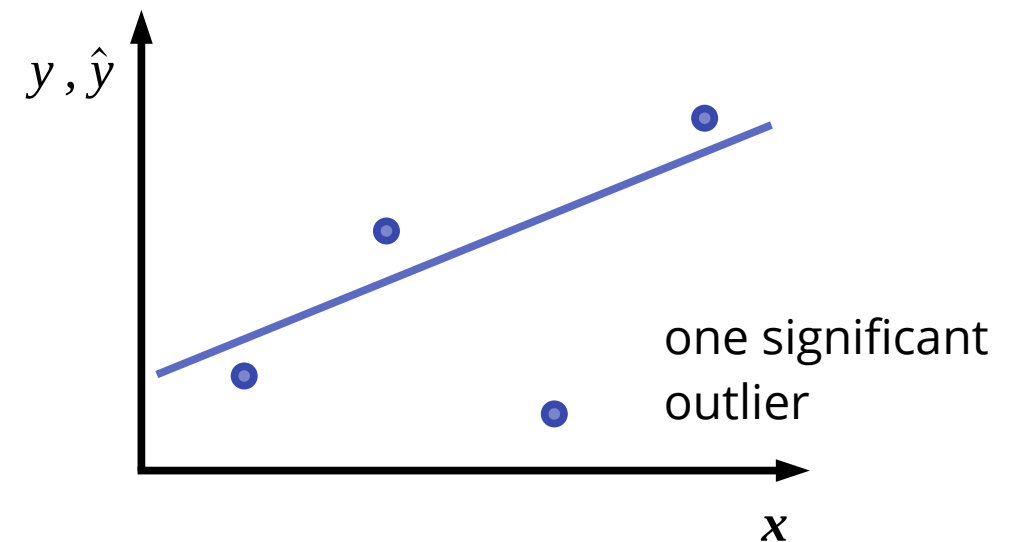
Comparing regression task metrics

We compare both MAE and RMSE for two different (and tiny) datasets:



$(y_i' - \hat{y}_i):$ -1 1 -1 1

→ MAE = 1; RMSE = 1



$(y_i' - \hat{y}_i):$ -1 1 -10 1

→ **MAE = 3.25; RMSE = 5.07**

RMSE is more sensitive to outliers. It depends on your model and problem if this is beneficial, or not.

Binary classification metrics

In a **binary classification** problem, we can simply compare our predictions with the ground-truth, which allows us readily to define a number of different classification metrics:

Accuracy = $(TP + TN) / (TN + TP + FP + FN)$

What is the overall fraction of correct (positive and negative) predictions?

Precision = $TP / (TP + FP)$ (quantifies “correctness”)

What fraction of our positive predictions is truly positive?

Recall = $TP / (TP + FN)$ (quantifies “completeness”)

What fraction of actual positives has been identified?

Prediction	positive	negative
	True positive	False positive
negative	False negative	True negative
Ground-Truth		

Binary classification metrics

Example: LULC classification – Forest class

Accuracy = $(TP + TN) / (TN + TP + FP + FN)$

What is the overall fraction of correct (positive and negative) predictions?

Precision = $TP / (TP + FP)$ (quantifies “correctness”)

What fraction of our positive predictions is truly positive?

Recall = $TP / (TP + FN)$ (quantifies “completeness”)

What fraction of actual positives has been identified?

accuracy = 0.95:

we correctly identified 95% of all pixels as either correctly forest or correctly non-forest

precision = 0.95:

95% of the forest pixels that our model identified are correct

recall = 0.95:

we correctly identified 95% of the forest pixels in the entire image

How to report metrics?

There are different ways to report metrics:

Individual metric: the metric based on a single model and dataset

Best-of- n : the best result out of n different model runs (e.g., trained with different seeds or different datasets → cross-validation)

Averaging results: the average metric over n model runs (e.g., trained with different seeds or different datasets → cross-validation) + standard deviation

α

$\alpha_0 \quad \alpha_1 \quad \alpha_2 \quad \alpha_3$

$$\alpha = \frac{1}{N} \sum \alpha_i$$

The best choice depends on the specific problem and use case.

Confusion matrix

A common way to visualize the performance of a classification model in a multi-class scenario and to find systematic problems, is to use a confusion matrix:

		Ground-Truth		
		Forest	Water	Built-up
Prediction	Forest	82	2	7
	Water	3	152	0
	Built-up	5	0	

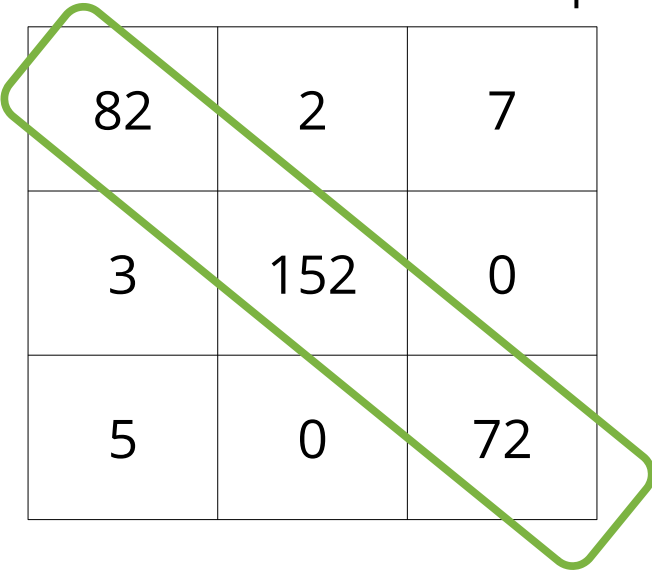
3 Forest samples wrongly predicted to be Water samples.

- Each cell contains the number of samples of a given class (ground-truth) that were predicted by the model to be of another given class.

Confusion matrix

A common way to visualize the performance of a classification model in a multi-class scenario and to find systematic problems, is to use a confusion matrix:

		Ground-Truth		
		Forest	Water	Built-up
Prediction	Forest	82	2	7
	Water	3	152	0
	Built-up	5	0	72



- Each cell contains the number of samples of a given class (ground-truth) that were predicted by the model to be of another given class.
- Diagonal elements are correctly identified, off-diagonal elements are misclassified

Confusion matrix

A common way to visualize the performance of a classification model in a multi-class scenario and to find systematic problems, is to use a confusion matrix:

		Ground-Truth		
		Forest	Water	Built-up
Prediction	Forest	82 TP	2 FP	7 FP
	Water	3 FN	152 TN	0 TN
	Built-up	5 FN	0 TN	72 TN

- Each cell contains the number of samples of a given class (ground-truth) that were predicted by the model to be of another given class.
- Diagonal elements are correctly identified, off-diagonal elements are misclassified

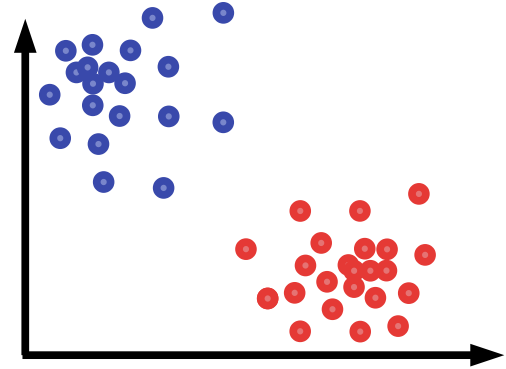
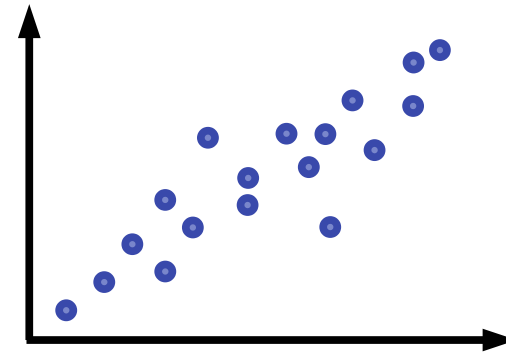
Confusion matrix: Some more metrics

Let's consider the task of land-use/land-cover classification, resulting in the following confusion matrix. With respect to each class, there is a number of additional informative metrics that we can define:

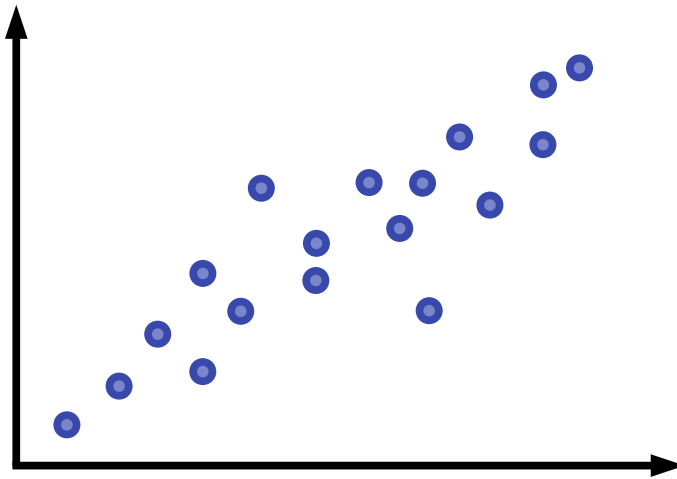
		Ground-Truth		
		Forest	Water	Built-up
Prediction	Forest	82 TP	2 FP	7 FP
	Built-up Water	3 FN	152	0
		5 FN	0	72 TN

- **Error of Omission:** $FP/(TP+FP)$
% of actual class samples that was incorrectly identified
Forest: $(3+5)/(82+3+5) = 8,9\%$
- **Error of Commission:** $FN/(TP+FN)$
% of predictions that were incorrect for a given class
Forest: $(2+7)/(82+2+7) = 9,9\%$
- **Producer's Accuracy:** $TP/(TP+FP) = \text{Precision}$
% of actual class samples that was correctly identified
Forest: $82/(82+3+5) = 91,1\%$
- **User's Accuracy:** $TP/(TP+FN) = \text{Recall}$
% of predictions that were correct for a given class
Forest: $82/(82+2+7) = 90,1\%$

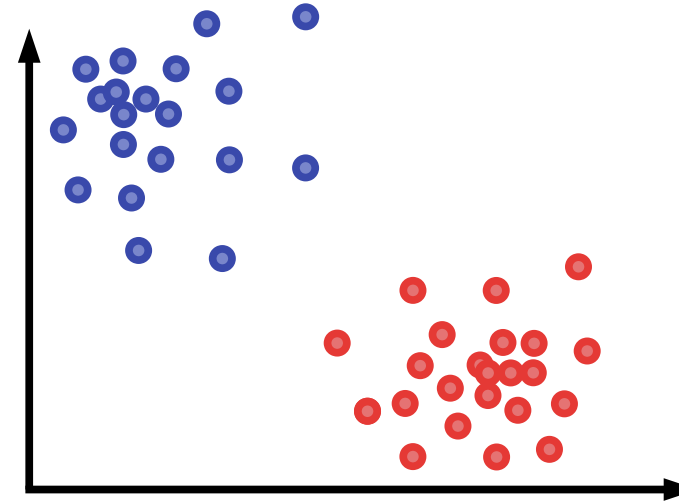
Linear models



Linear models assume **linearity** in the underlying data. They are rather simple but convey many of the concepts utilized in other, more complex models.



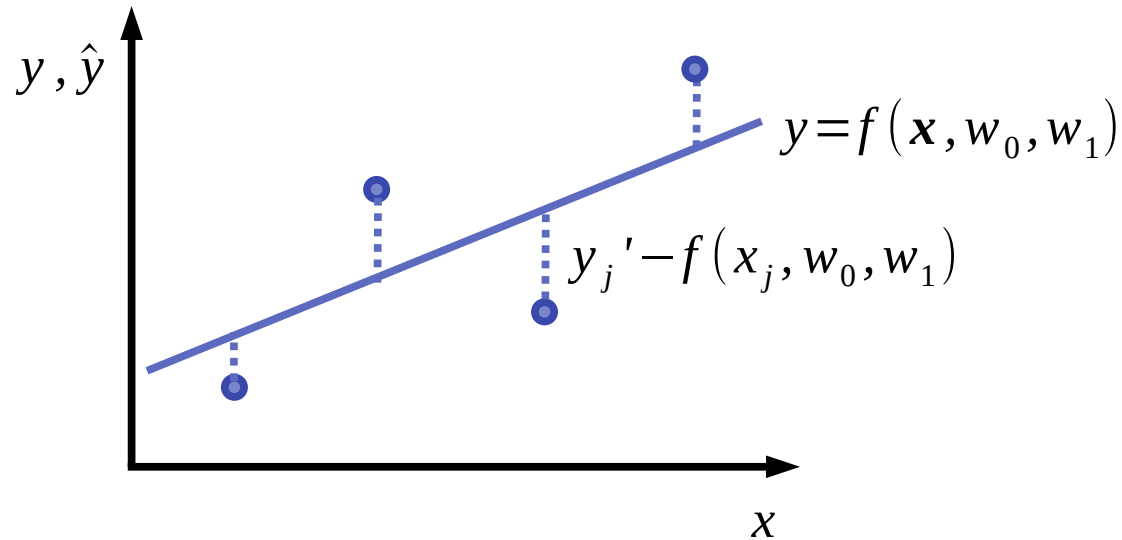
Linear regression



Linear classification

Linear regression (univariate)

Find weights w_0 and w_1 so that the linear function $f(x) = w_1 x + w_0$ with input x and output y best fits the data containing ground-truth values y' .



How can we learn w_0 and w_1 from data?

Linear regression (univariate) – Least squares fitting

Idea: minimize squared errors of prediction with respect to ground truth for each data point:

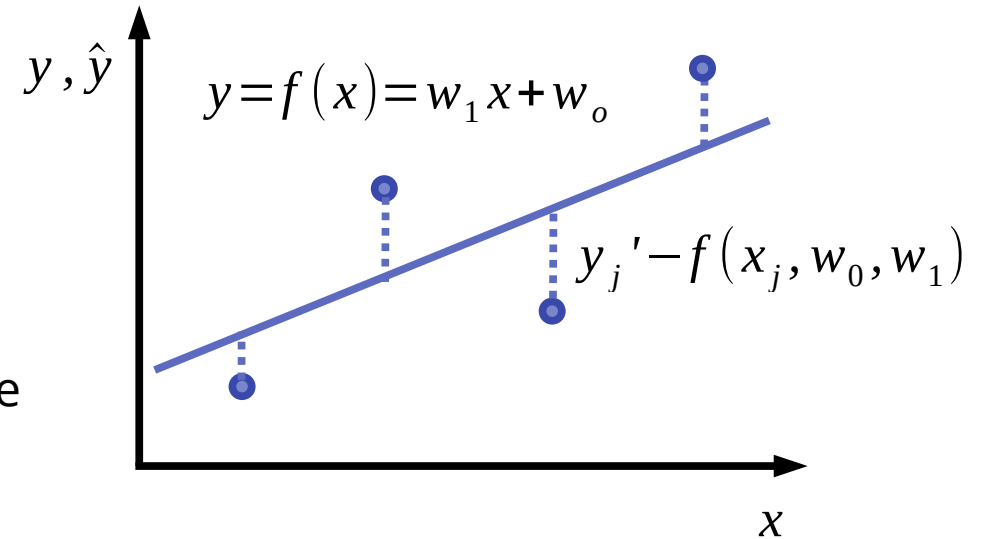
for data point j : $[y_j' - f(x_j, w_0, w_1)]^2$

We define a **Loss** (or Objective) function that is the sum of the squared errors over all data points:

$$L = \sum_j^N (y_j' - f(x_j, w_0, w_1))^2 = \sum_j^N (y_j' - (w_1 x_j + w_0))^2$$

We can find the best-fit model parameters, by **minimizing** the Loss function with respect to those two model parameters:

$$\frac{\partial L}{\partial w_0} = 0 \quad \frac{\partial L}{\partial w_1} = 0 \quad \rightarrow \text{closed-form expressions for best-fit } w_0 \text{ and } w_1.$$



Least-squares + linear model function: the resulting minimum of the Loss function is **global**, i.e., the “learns” immediately the best-possible solution!

Linear classifier (two-dimensional case)

Linear functions can also be used as a classifier if the data are linearly separable.

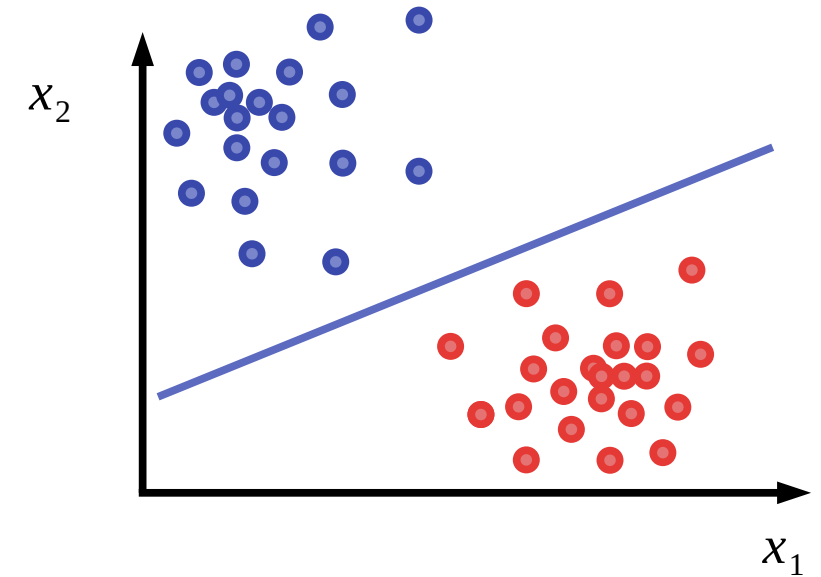
Define decision boundary $f(\mathbf{x}, \mathbf{w}) = \mathbf{w} \mathbf{x} = w_0 + w_1 x_1 + w_2 x_2$
such that:

Class 1: $f(\mathbf{x}, \mathbf{w}) \geq 0$

Class 0: $f(\mathbf{x}, \mathbf{w}) < 0$

We can define class assignments through a threshold function:

$$\bar{f}(\mathbf{x}, \mathbf{w}) = \begin{cases} 1 & \text{if } f(\mathbf{x}, \mathbf{w}) \geq 0 \\ 0 & \text{if } f(\mathbf{x}, \mathbf{w}) < 0 \end{cases}$$



How can we learn $\mathbf{w}$ from data?

Linear classifier (two-dimensional case) – Perceptron learning rule

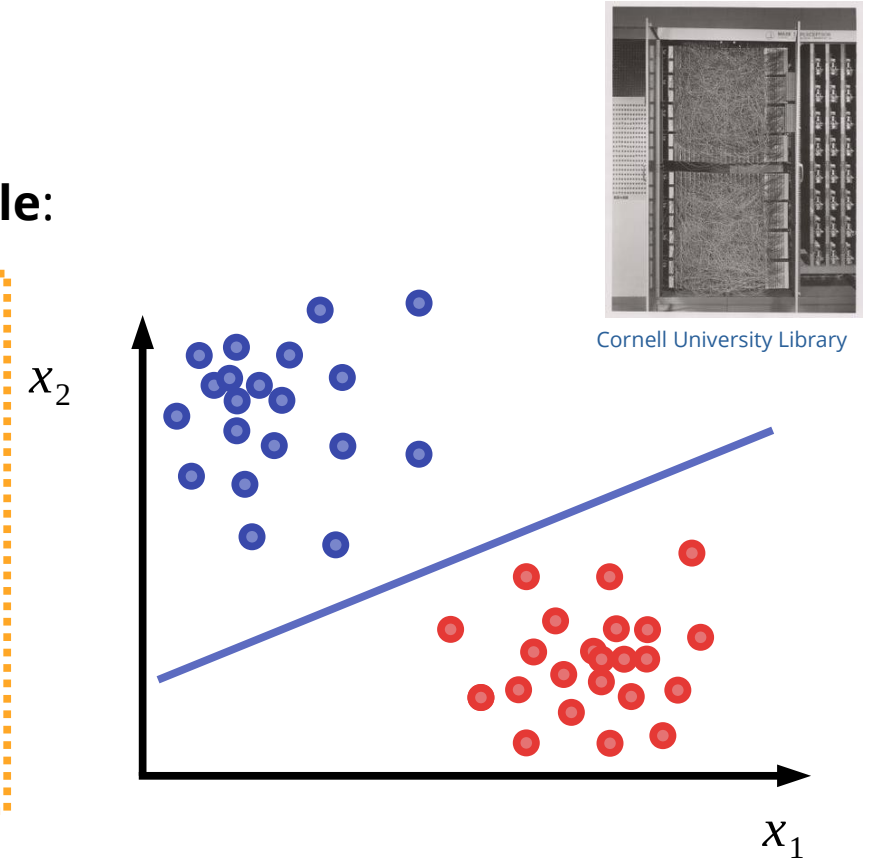
We define the following algorithm as the **Perceptron learning rule**:

We consider each data point, consisting of $\mathbf{x}$ and ground-truth label y' and check whether the prediction from $\bar{f}(\mathbf{x}, \mathbf{w})$ is correct, or not. If...

$\bar{f}(\mathbf{x}, \mathbf{w}) = y$, then do nothing.

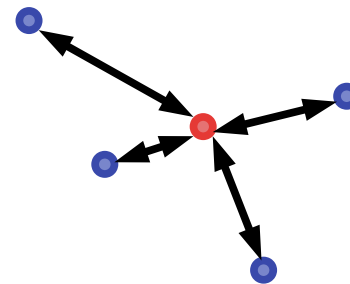
$\bar{f}(\mathbf{x}, \mathbf{w}) = 0$ but $y' = 1$, then increase w_i if $x_i \geq 0$, or vice versa.

$\bar{f}(\mathbf{x}, \mathbf{w}) = 1$ but $y' = 0$, then decrease w_i if $x_i \geq 0$, or vice versa.



Weights are adjusted by a step size that is called the **learning rate**. By iteratively running this algorithm over your training data multiple times, the weights can be learned so that the model performs properly. A solution is “learned” iteratively here.

Nearest-Neighbor models



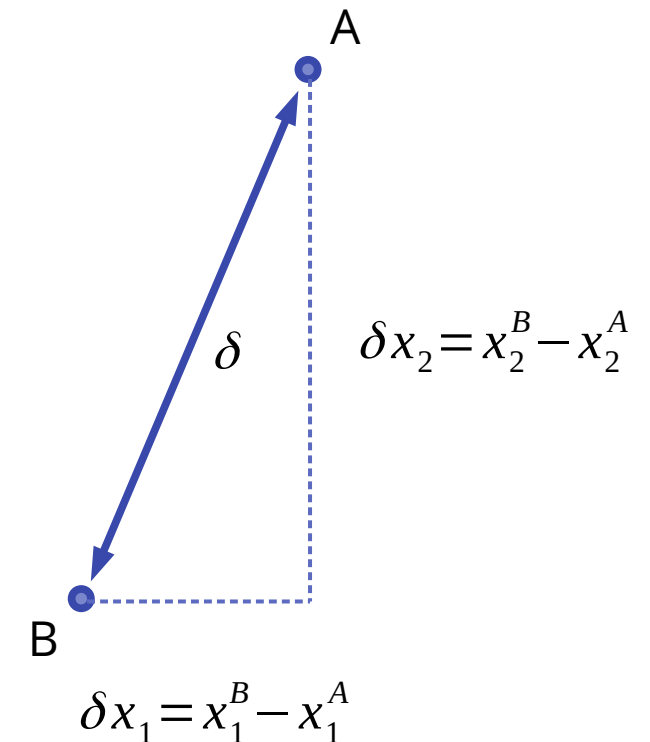
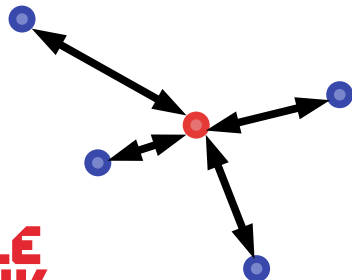
Nearest neighbor models

Nearest neighbor models are **non-parametric** and simply rely on **distances** between data points.

Distances can be defined as metrics. A common distance metric is the **Euclidean distance** between two data points $A=(x_1^A, x_2^A)$ and $B=(x_1^B, x_2^B)$:

$$\delta = \sqrt{\delta x_1^2 + \delta x_2^2} = \sqrt{(x_1^B - x_1^A)^2 + (x_2^B - x_2^A)^2}$$

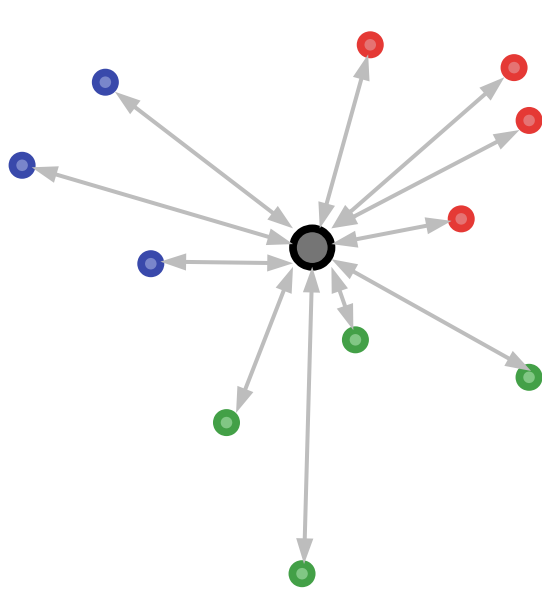
Nearest neighbor methods utilize distances between datapoints for **classification** and **regression** tasks.



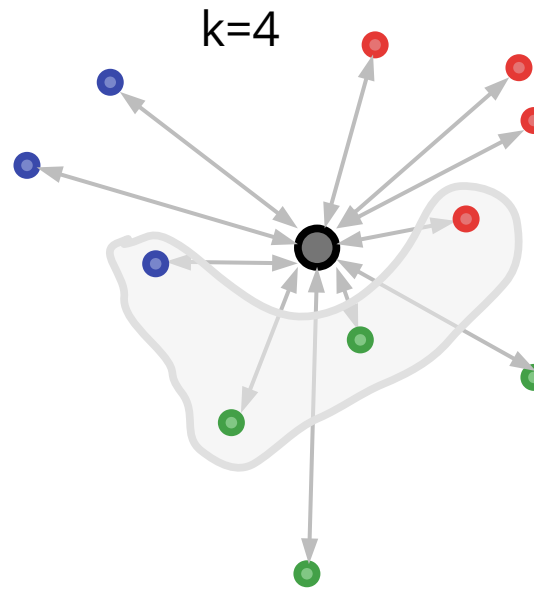
***k*-nearest neighbor classification**

k-nearest neighbor (knn) classifiers predict class affiliation of an unseen data point based on **majority voting** of its ***k* nearest neighbors** in a seen data set with ground-truth labels.

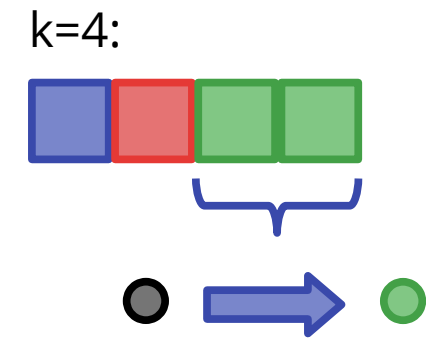
knn models are not trained in the general sense. Instead, the distance of each unseen data point from all seen data points is calculated.



compute distances

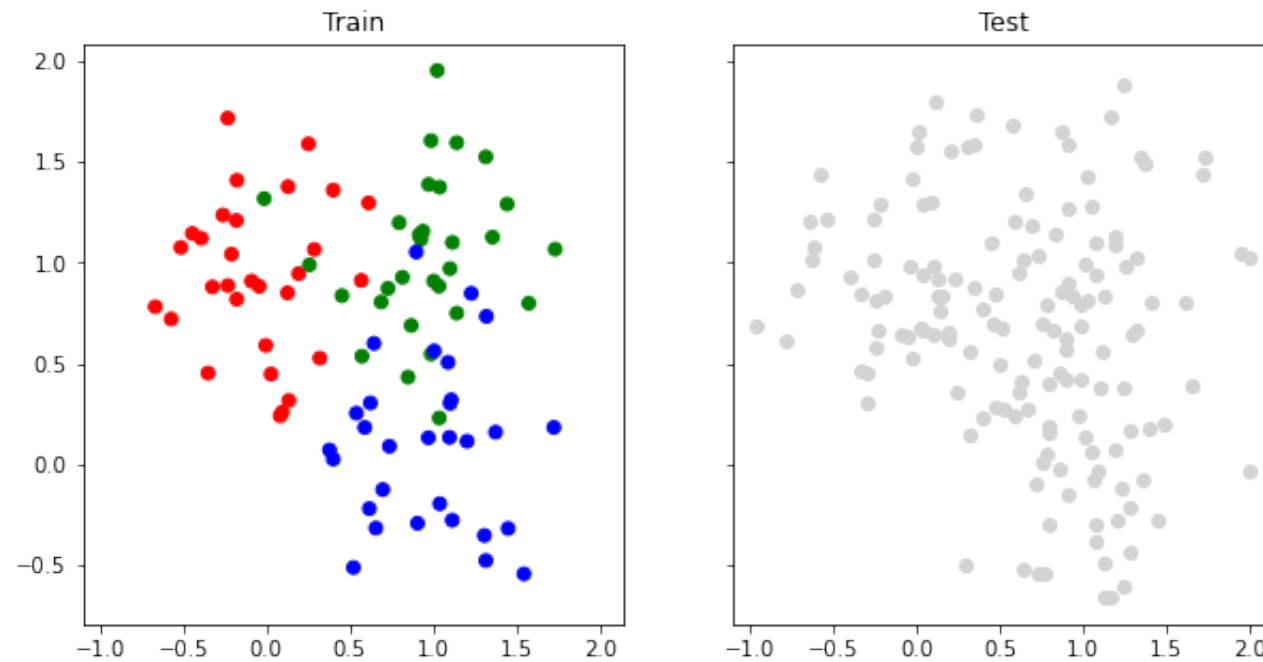


identify *k* nearest neighbors



class assignment

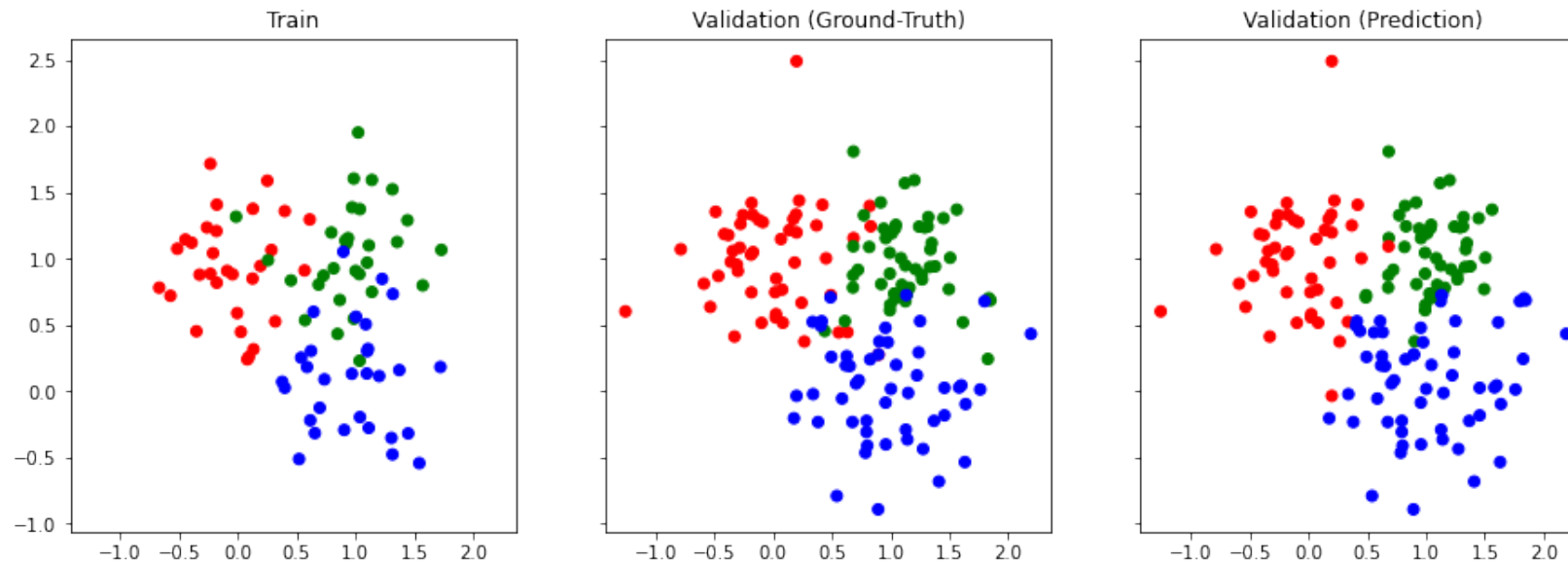
k-nearest neighbor classification - Example



3 overlapping clusters

How well can knn classify
our test data set?

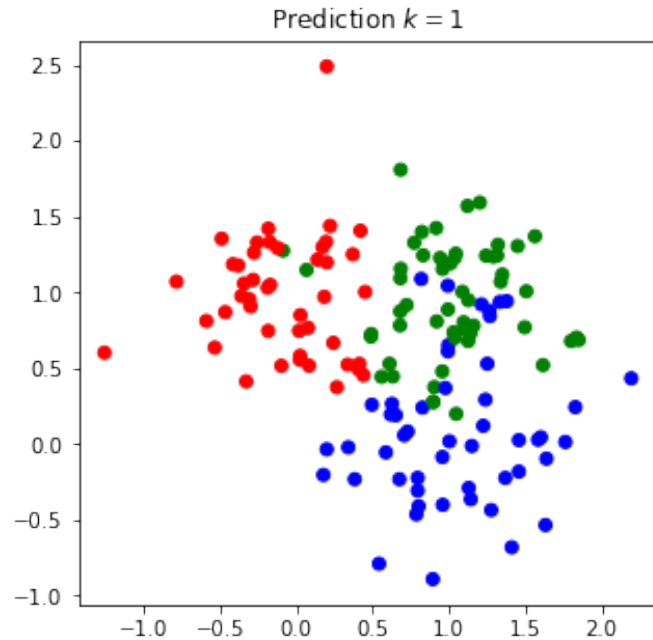
k-nearest neighbor classification - Example



k=5: accuracy=0.867

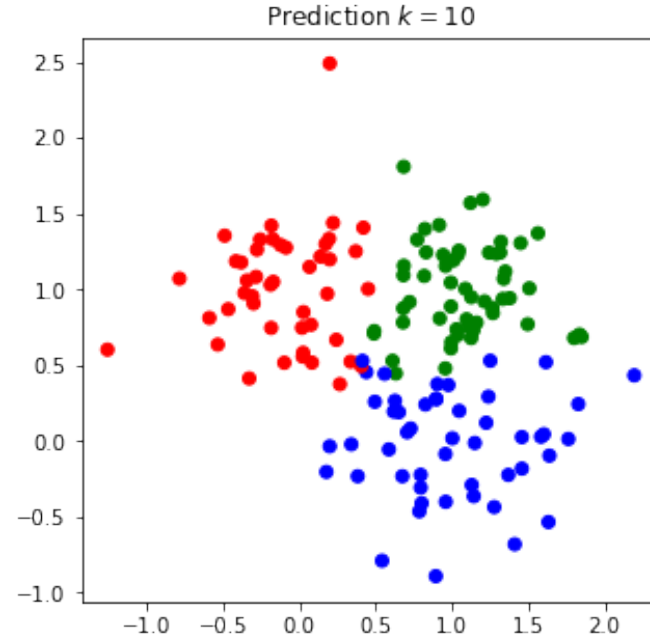
Hyperparameter k has an impact on how well the model generalizes to unseen data: perform a **hyperparameter search**!

k-nearest neighbor classification - Example



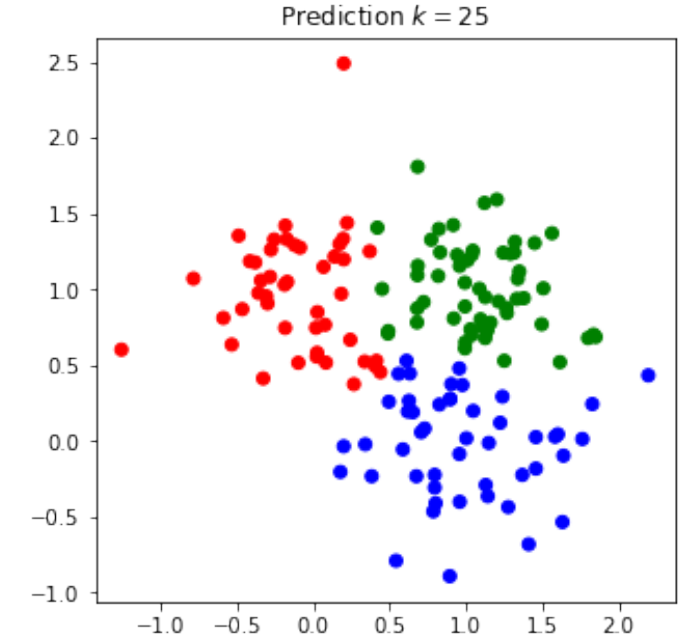
$k=1$
 $\text{accuracy}_{\text{val}}=0.800$

Overfitting!



$k=10$
 $\text{accuracy}_{\text{val}}=0.893$
 $\text{accuracy}_{\text{test}}=0.880$

Best performance

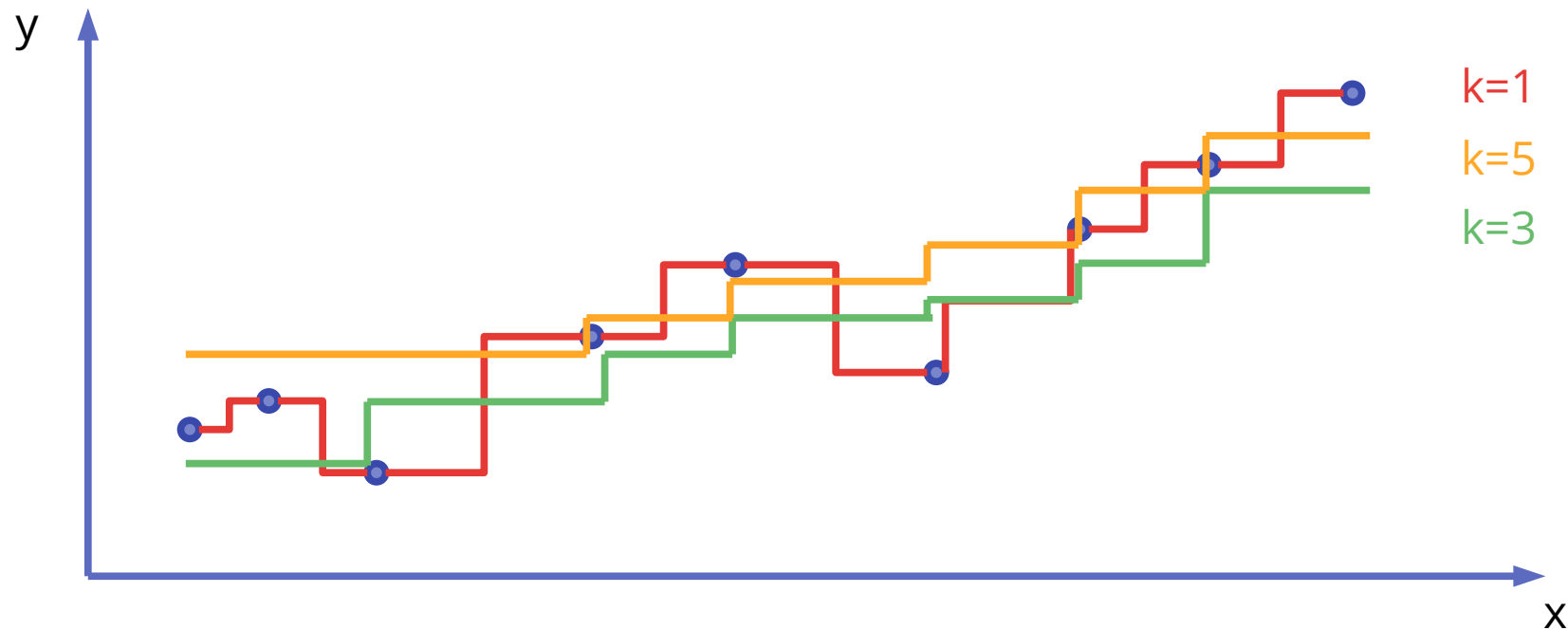


$k=25$
 $\text{accuracy}_{\text{val}}=0.873$

Underfitting!

k-nearest neighbor regression

Nearest neighbor methods can also be implemented as regressors that are able to **interpolate** between and **smoothen** available data.



We introduced the basic elements of supervised machine learning.

We discussed the supervised learning pipeline and the means to evaluate a trained model.

We introduced two simple machine learning methods, both of which can be used for regression and classification tasks.

Let's get our hands dirty and have a look at some example applications:

- [Linear Regression on the California Housing Dataset](#)
- [Pixel-wise Classification on Sentinel-2 Satellite Imagery](#)

That's all folks!