AI/ML in the omics

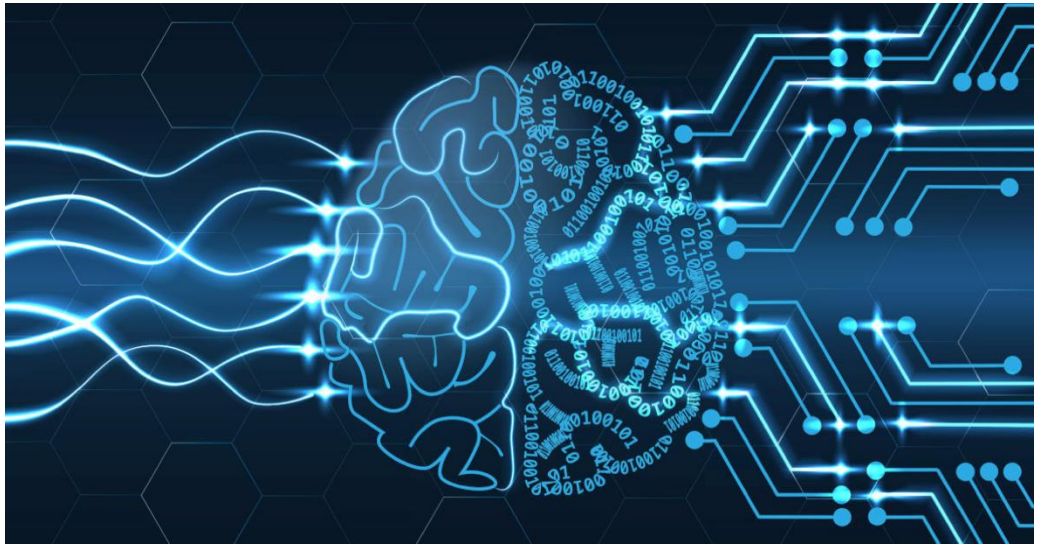
Nikolaus Fortelny

Wolfgang Esser-Skala

January 2026

Why learn about AI / ML?

- TODO - examples



Outline & Structure

- **Lecture:**
  Introduction to basic ML and outlook to more complex AI algorithms.
- **Practical session (XX % o grade):**
  Implementation of basic ML algorithms under guidance. Evaluation: upload your Jupyter notebooks.
- **Analyze a dataset of your choice (XX % o grade):**
  Implement ML algorithm for unseen data yourself. Evaluation: upload your Jupyter notebook and present / discuss your results.
- **Journal club (XX % o grade):**
  Discuss basic ML concepts in one journal article. Evaluation: Present an article.
- **Exam (XX % o grade):**
  Basic concepts of AI/ML. Evaluation: Exam.

Schedule

- Day 1 (today): ML basics / sklearn (lecture + practice)
- Day 2: ML calculations / what is learning? pytorch (lecture + practice)
- Day 3: Multi-class predictions (lecture + practice)
- Day 4: Hyperparameters (lecture + practice)
- Day 5: Random forest and interpretability (lecture + practice)

break of one week → work on final project

- Day 6: Journal club presentations + work on final project
- Day 7: Journal club presentations + work on final project
- Day 8: Final project presentations + discussions
- Day 9: Final project presentations + discussions
- Day 10: Exam

(ideal) prerequesites

- Bioinformatics practicals:
  Python programming

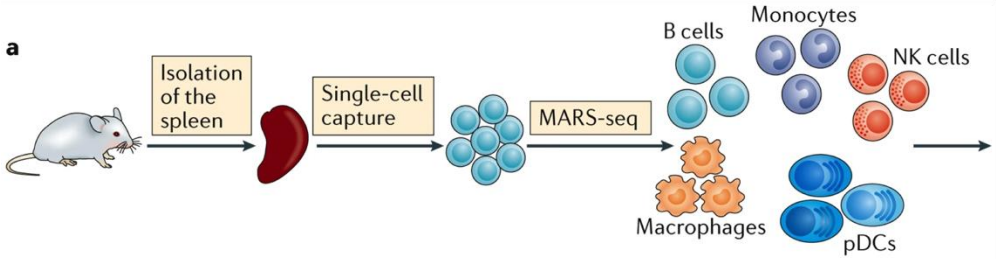
- Module: Biomedical Data Analysis (MSc MedBio)
  R programming; statistics & data science in omics

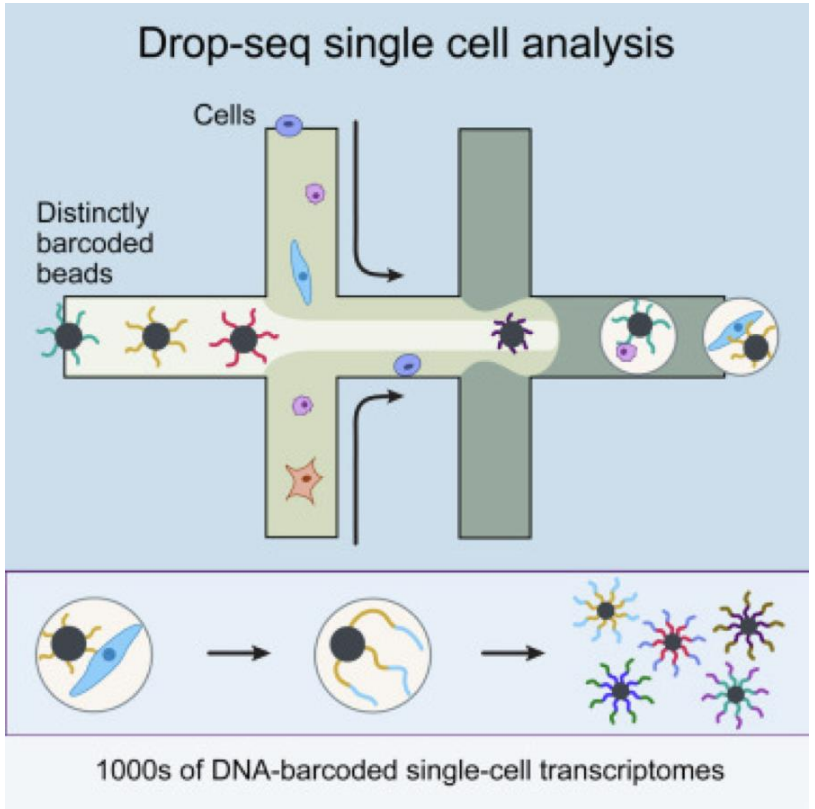
- Lecture: Big Data Management (MSc MolBio)
  ML & data science in omics; single-cell data

Single-cell RNA-seq as “omics” data

- **Wetlab (experimental)**
  - Tissue
  - Single-cell suspension
  - Sequencing library

- **Drylab (computational)**
  - Raw reads
  - Aligned reads
  - Data matrix
  - Clusters of cells
  - Assign cell types
  - Differential genes



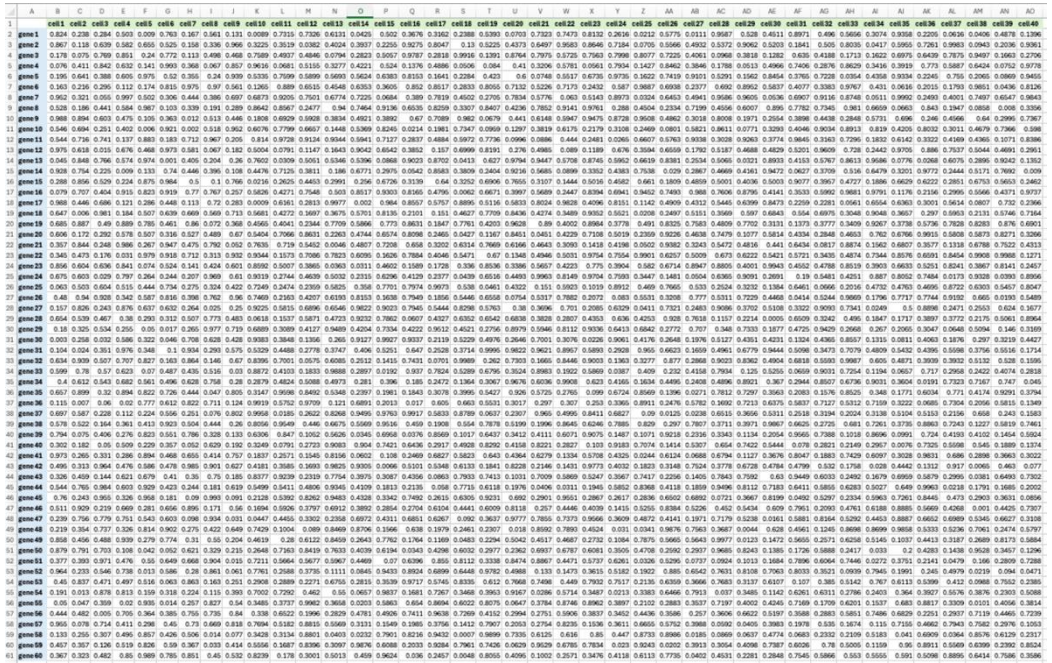


https://doi.org/10.1186/s13073-017-0467-4
https://doi.org/10.1038/nri.2017.76

What do the resulting data look like?

Data

Genes

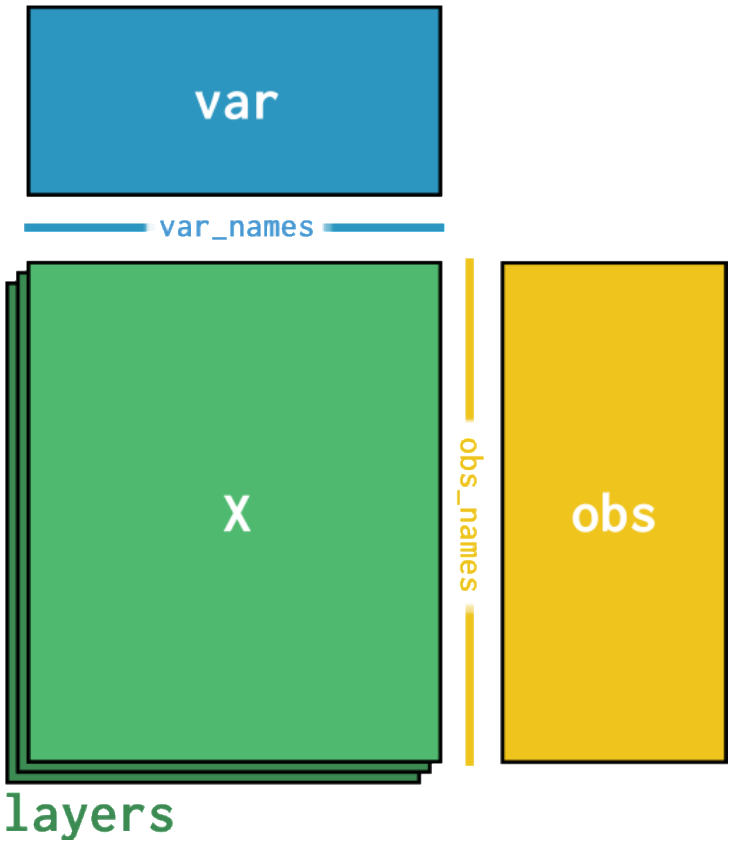


Cells

Metadata

Cell barcode	Sample
ATATGAC	Sample 1
TTAGAGT	Sample 1
TACCATT	Sample 2
CATTAGA	Sample 2
CCCATTA	Sample 3
...	...

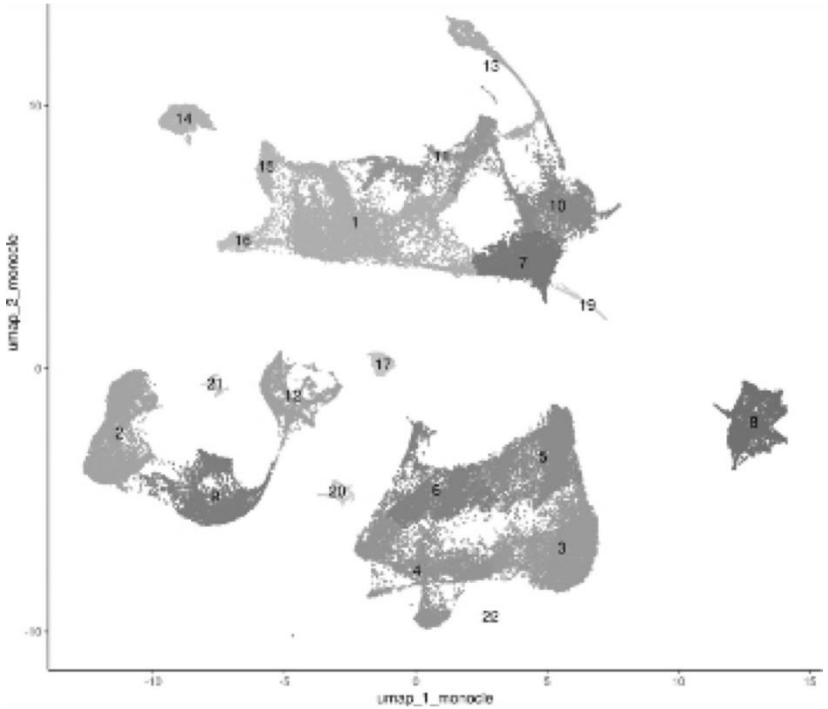
Data structure



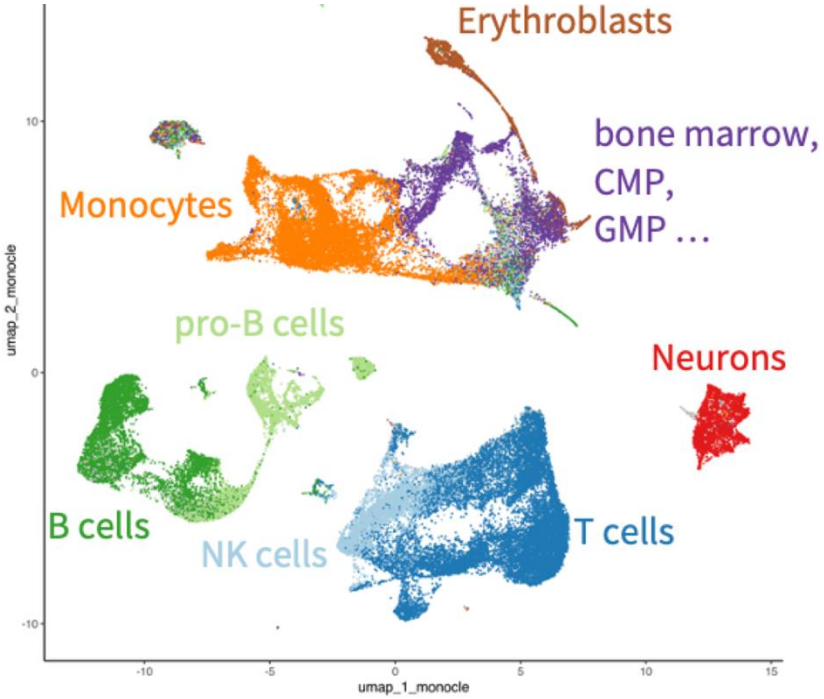
- **X: Data matrix**
  - gene expression (values)
- **Obs: “observations” (meta data)**
  - describes cells
  - sample, cell type, organ, treatment… for each cell
- **Var: “variables” (features):**
  - describes genes
  - one row per variable
  - gene length, gene name, IDs from different databases,… for each gene

AnnData: https://doi.org/10.21105/joss.04371

Our tasks: predict cell types



ML algorithm

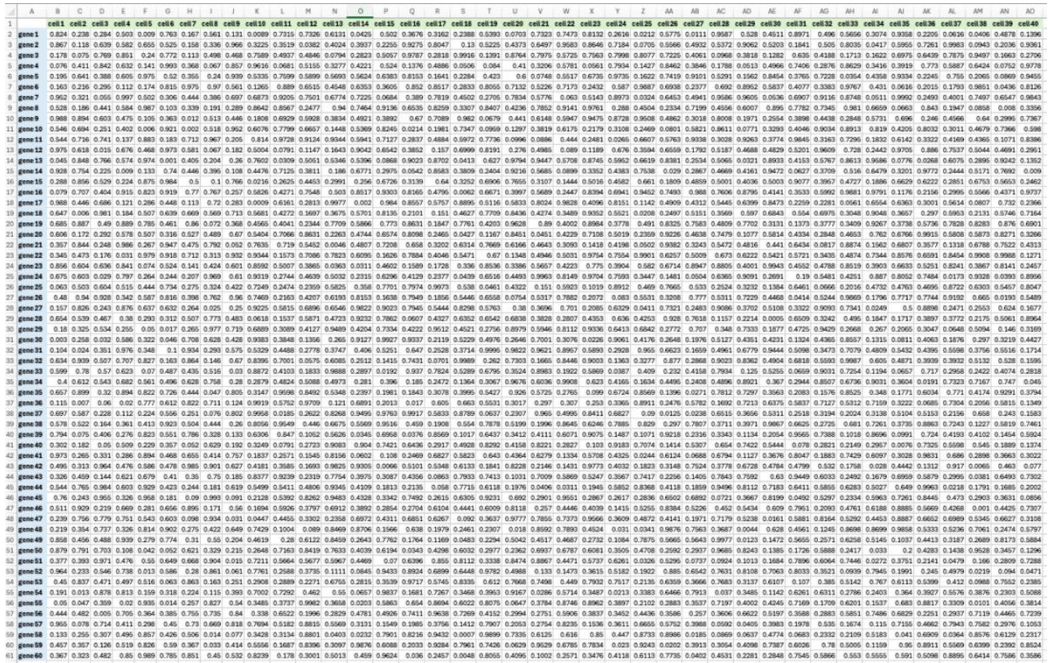


This is what you will learn in this course!

Result after the prediction

Data

Genes



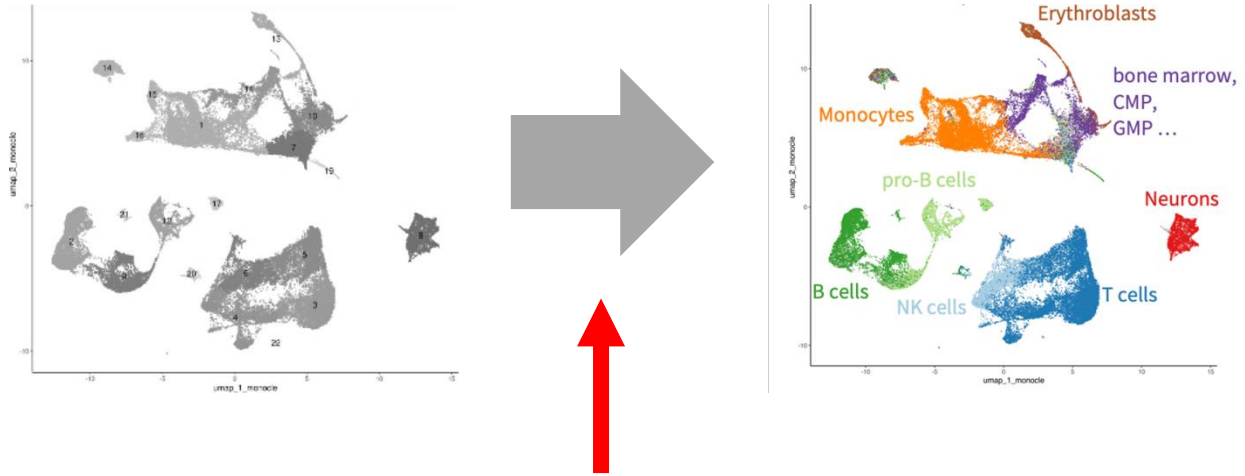
Cells

Metadata

Cell barcode	Sample	Cell type
ATATGAC	Sample 1	B cell
TTAGAGT	Sample 1	Neuron
TACCATT	Sample 2	B cell
CATTAGA	Sample 2	Monocyte
CCCATTA	Sample 3	Neuron
...	...	...

How do we achieve this?

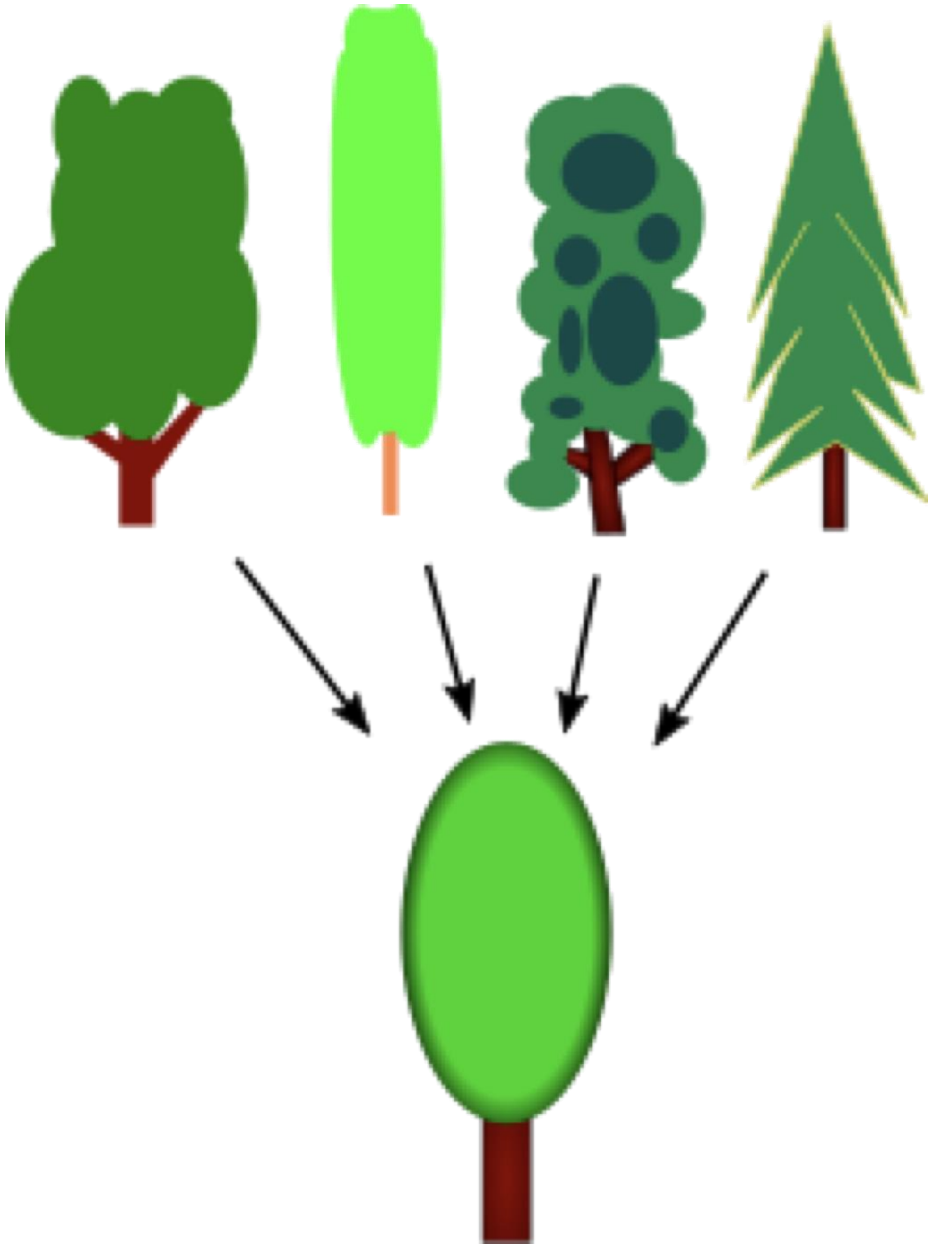
1. We have data where know the answer (another dataset).
2. We train an algorithm on this data.
   → train
3. We apply this model to data where we do not know the answer.
   → predict



We train an algorithm based on (additional) data where we know the correct cell type!

Train / test splits

- We know the answer in our dataset!
- We want to learn a function that will also predict on **unseen** data
- Will our trained model **generalize**?



A generalised tree

Structure of ML problems

1. We have data where know the answer (cell type).
2. We train an algorithm on part of this data

   → **train**
3. We test on the remaining data (where we know the answer but the algorithm hasn’t seen it)!

   → **test**
4. We apply this model to data where we do not know the answer.

   → **predict**

Performance evaluation 1: continguency tables and metrics

- Needs true labels and predicted labels
- Creates a continguency table
- Various metrics:
  - Sensitivity / specificity
  - Precision / recall
  - …

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

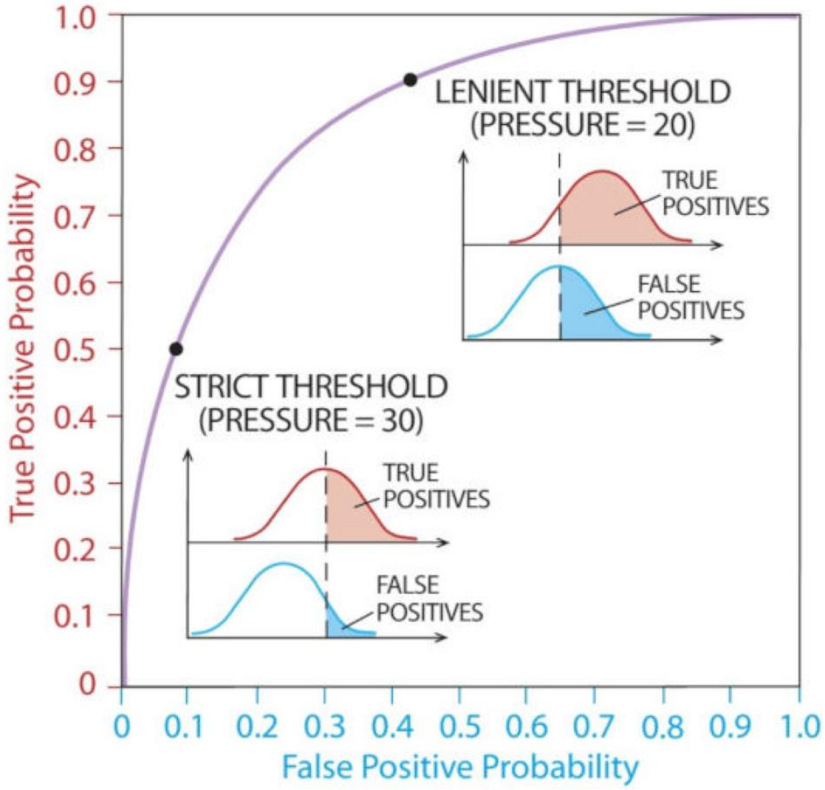
We will use accuracy (easiest)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

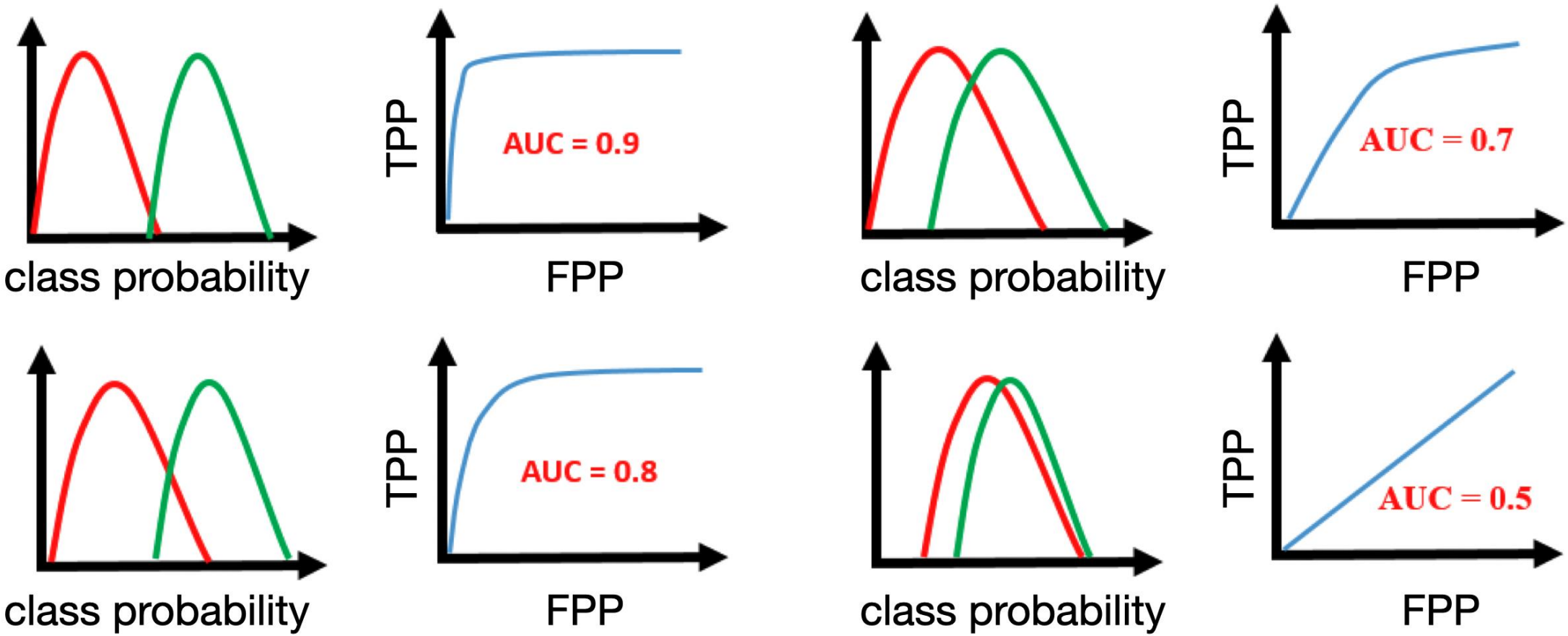
https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers

Performance evaluation 2: ROC AUC curves

- ML predictions are usually class probabilities [0;1]
- A cutoff (e.g. > 0.5) can be used to predict class 1 and class 0
- ROC evaluates **all cutoffs**
- At every step, we calculate
  - Sensitivity
    (true positive probability)
  - Specificity
    (false positive probability)



Performance evaluation 2: ROC AUC curves



Practical session 1

1. Install and load python packages
2. Download data (only bash step)
3. Load data (anndata)
4. Normalize (logCPM) and filter the data (lowly expressed genes)
5. Train ML model with sklearn
6. Evaluate performance
7. Train/test split

→ Explore the objects (type, dimensions, values)!

Python concepts: basics

- Python works with indentation (and not {})
- Functions are pre-defined (or imported from packages)
- Basic data structures (lists/set and dictionary)
- Objects have methods (~ functions) and attributes (~ variables)
  - obj.method1()
  - obj.method2()
  - obj.attribute1
  - obj.attribute2

Python concepts: pandas DataFrame

- Uses a dict to initialize the DataFrame
- my_df.population will access the respective column (attribute)
- my_df["population"] also works
- Columns are attributes of the DataFrame object

- two-dimensional data structure
- columns: variables
- rows: observations

	Jedermann	*leap year*	*population*
2000	Ulrich Tukur	True	143120
2005	Peter Simonischek	False	148546
2010	Nicholas Ofczarek	False	149065
2015	Cornelius Obonya	False	149728

```
DataFrame({"population": [143120, 148546,
                          149065, 149728],
           "Jedermann": ["Ulrich Tukur", "Peter Simonischek",
                         "Nicholas Ofczarek", "Cornelius Obonya"],
           "leap year": [True, False,
                         False, False]},
           index=[2000, 2005, 2010, 2015])
```

Practical session 1: Install packages

```
import numpy as np
import pandas as pd
import sklearn.metrics as skm
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import anndata
import os
from scipy import stats
```

→ How do you call the functions?

Practical session 1:
Download and load the data

wget -O data/my_dataset_small.h5ad
https://datasets.cellxgene.cziscience.com/9d09a542-672b-4cc0-b4d4-05d318f5705f.h5ad

data = anndata.read_h5ad(...)

Note:

- read_h5ad is a function from anndata
- data.X, data.obs, and data.var are attributes of the object

Practical session 1: Normalize the data

You can modify data.X as follows:

data.X = ...

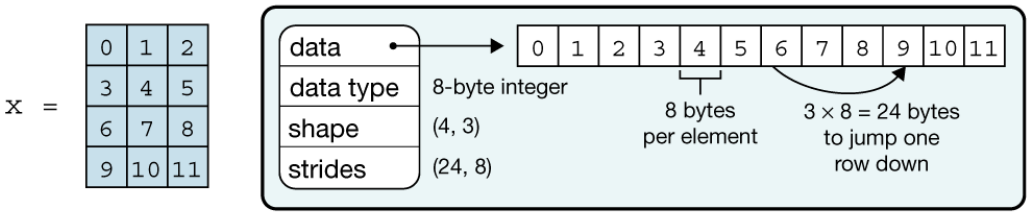
For example:

data.X /= data.X.sum(axis=1)

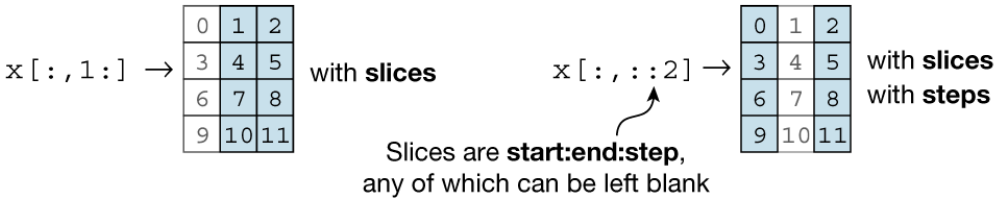
calculates the sum of reads per cell and devide the data by this sum

Practical session 1: Numpy

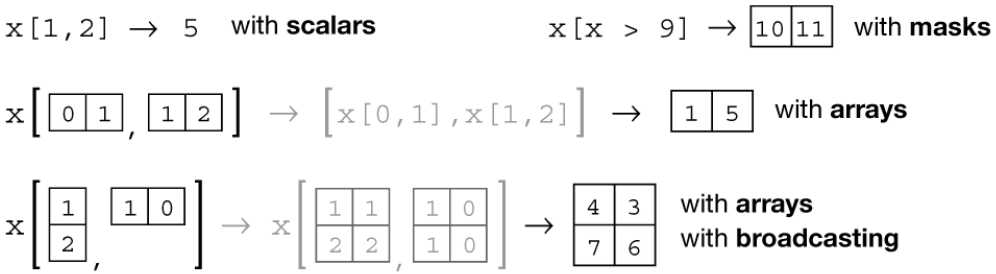
a Data structure



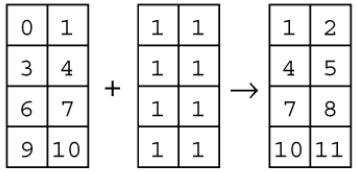
b Indexing (view)



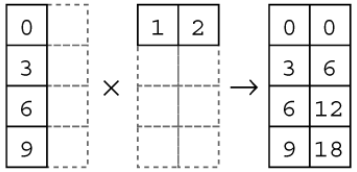
c Indexing (copy)



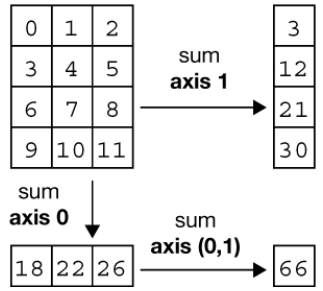
d Vectorization



e Broadcasting



f Reduction



g Example

```
In [1]: import numpy as np

In [2]: x = np.arange(12)

In [3]: x = x.reshape(4, 3)

In [4]: x
Out[4]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])

In [6]: x = x - np.mean(x, axis=0)

In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
       [-1.5, -1.5, -1.5],
       [ 1.5,  1.5,  1.5],
       [ 4.5,  4.5,  4.5]])
```

Practical session 1: Train ML model (1)

Define:

1. Input data X: normalized data.X
2. Output data Y: list of cell_type labels from data.obs
3. Model model: LogisticRegression(...)

LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='deprecated', *,
C=1.0, l1_ratio=0.0, dual=False, tol=0.0001, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs',
max_iter=100, verbose=0, warm_start=False, n_jobs=None) #          [source]
```

Practical session 1: Train ML model (2)

Fit the model to data using the method:

model.fit(...)

`fit(X, y, sample_weight=None)` [source]

Fit the model according to the given training data.

Parameters:

X : *{array-like, sparse matrix} of shape (n_samples, n_features)*

Training vector, where `n_samples` is the number of samples and `n_features` is the number of features.

y : *array-like of shape (n_samples,)*

Target vector relative to X.

sample_weight : *array-like of shape (n_samples,) default=None*

Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

> ***Added in version 0.17:*** *sample_weight* support to LogisticRegression.

Returns:

self

Fitted estimator.

Practical session 1: Evaluate the model

Predict labels based on the trained model:

predict(*X*) [source]

Predict class labels for samples in X.

Parameters:

X : *{array-like, sparse matrix} of shape (n_samples, n_features)*

The data matrix for which we want to get the predictions.

Returns:

y_pred : *ndarray of shape (n_samples,)*

Vector containing the class labels for each sample.

→ We will then compare the predicted labels with the true labels

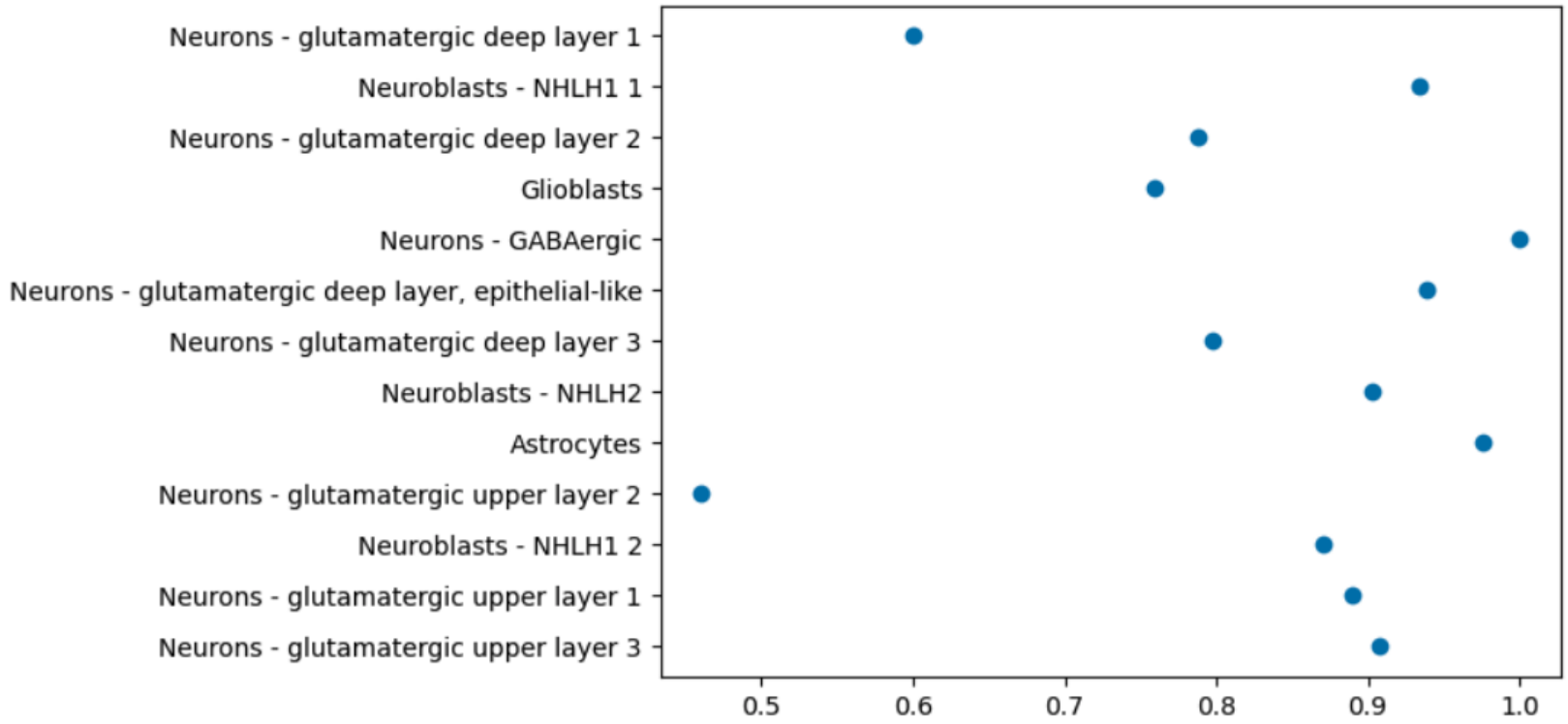
Plotting

- We will make simple plots
- matplotlib package
- You can use any list-like element as x-axis and y-axis
- Shown here for a pandas DataFrame

	cell_type	accuracy	number
0	Neurons - glutamatergic deep layer 1	0.604167	48
0	Neurons - GABAergic	0.933333	30
0	Neuroblasts - NHLH2	0.743590	39
0	Neuroblasts - NHLH1 1	0.851852	54
0	Neurons - glutamatergic upper layer 2	0.630769	65
0	Neurons - glutamatergic upper layer 3	0.825000	120
0	Neurons - glutamatergic deep layer 2	0.712500	80
0	Astrocytes	0.938272	81
0	Glioblasts	0.782609	46
0	Neurons - glutamatergic deep layer, epithelial...	0.977778	90
0	Neurons - glutamatergic upper layer 1	0.871560	109
0	Neurons - glutamatergic deep layer 3	0.770270	74
0	Neuroblasts - NHLH1 2	0.754098	61

In [19]:
```
# Make a plot
plt.scatter(eval.accuracy, eval.cell_type)
```

Out[19]: <matplotlib.collections.PathCollection at 0x7f43206066c0>



Practical session 1: Train / test split

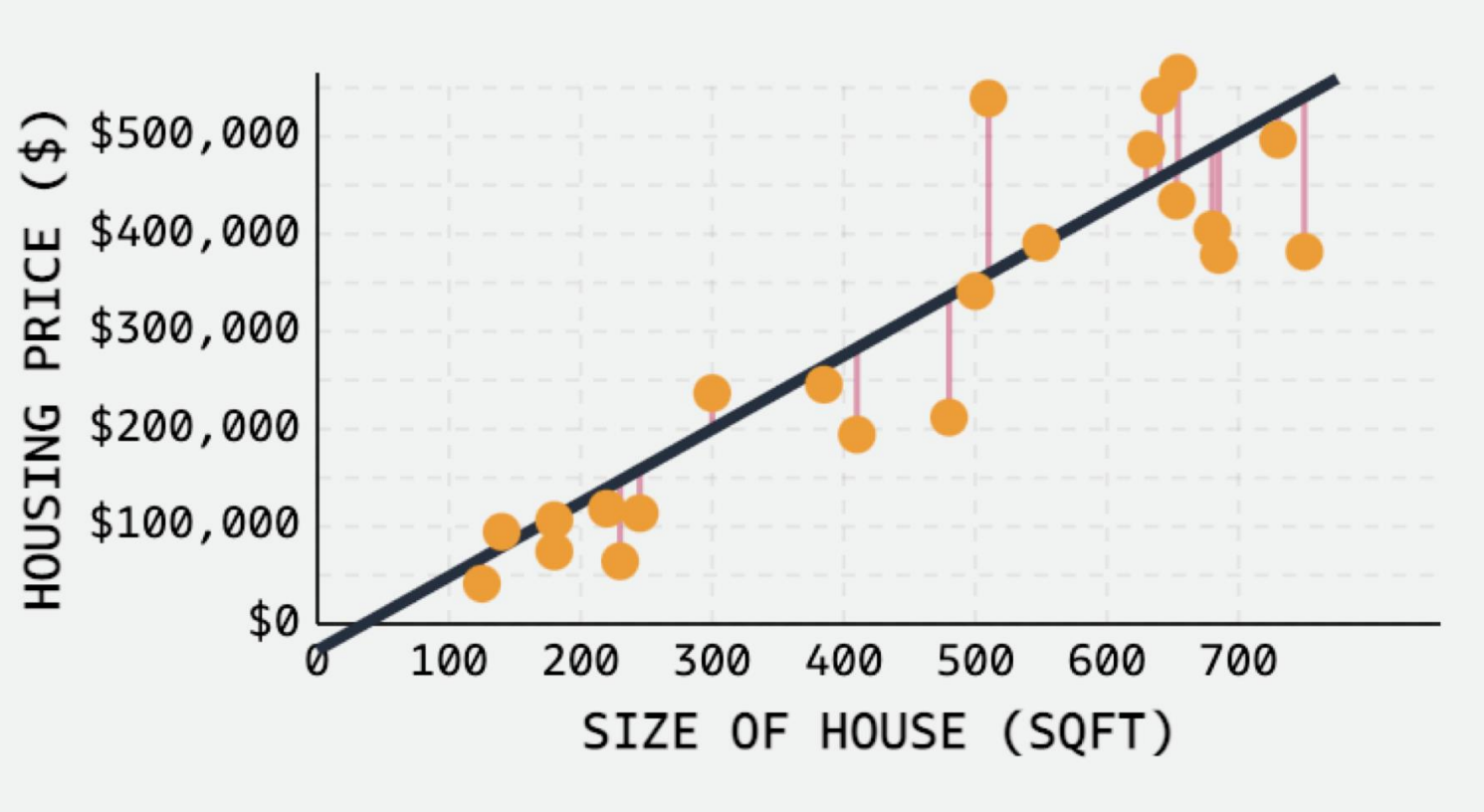
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
```

Notes:

- Use X_train and Y_train for training
- Use X_test and Y_test only after training is completed

Session 2: ML calculations

What is a model?



A formula to map input to output!

$y = a + b * x$

$y = w_0 + w_1 * x$

y ... housing price

x ... size of the house

a / w_0 ... intercept

b / w_1 ... slope

E.g. predict cell size from the expression of one gene

ML vocabulary

- **Vocabulary: Statistics - Machine learning**
  - Response, predictant (y) **output** ← we know this
  - Covariates, predictor, features (x) **input** ← we know this
  - Parameters, coefficients (w) **weights** ← we want this!

- **Two main types**
  - **Classification:** Predicting classes (cell types, …)
  - **Regression:** Predicting a number (time, red blood cell count, …)

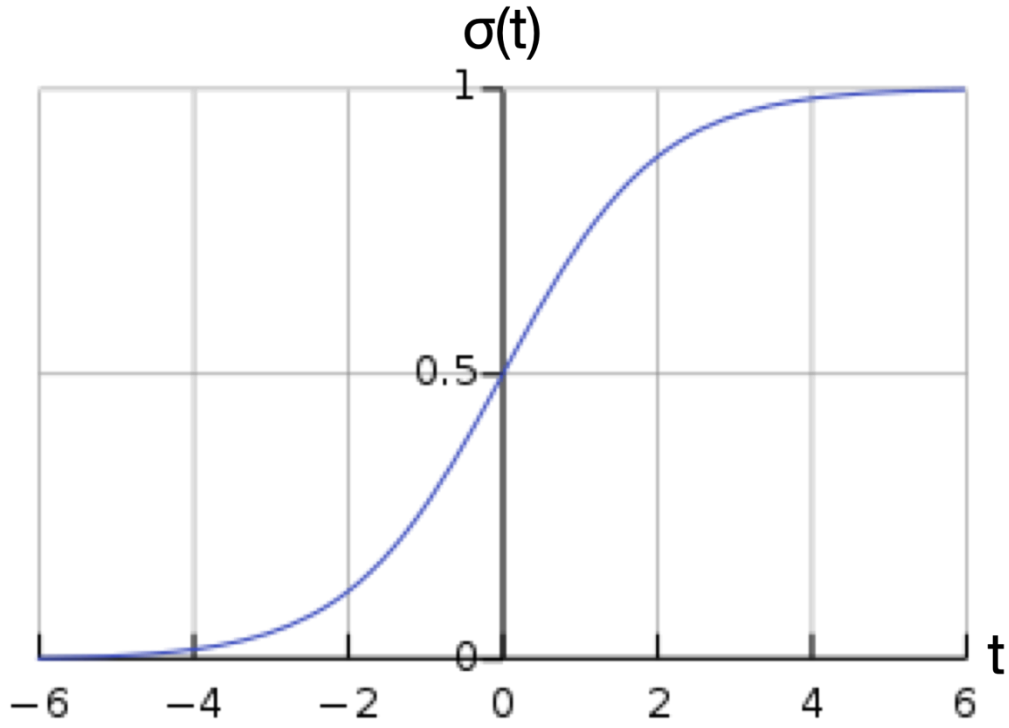
Classification instead of regression

We have a classification task:

- Cell type 1 → class 0
- Cell type 2 → class 1

Classification requires a sigmoid function:

- $y = \sigma(w_0 + w_1 * x_1)$
- y is bound between 0 and 1
  → probability of class 1



$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

E.g. predict cell type from the expression of one gene (x_1)

What does this weight mean?

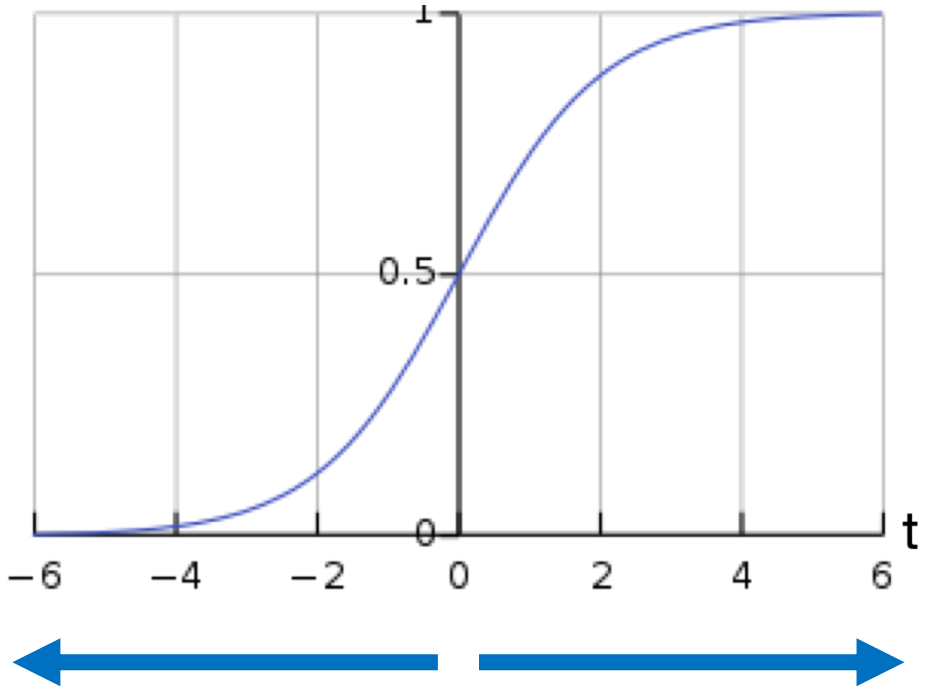
$w_1 = 0$ → not relevant

$w_1 < 0$ → class 0

$w_1 > 0$ → class 1

Large value → larger effect

$y = \sigma(w_0 + w_1 * x_1)$



Negative values push towards class 0

Positive values push towards class 1

How does this calculation like in our example?

Input (x): 20,000 expression values
Output (y): cell type 1 or cell type 2
Weights (β/w): which genes are important?

<u>Different ways of writing the same:</u>

$$y = \sigma(w_0 * x_0 + w_1 * x_1 + w_2 * x_2 + \dots + w_{20,000} * x_{20,000})$$

y … [1;0] cell type 1 or cell type 2
x_i … expression of gene I
x0 … is always 1 (intercept)
σ()… sigmoid function

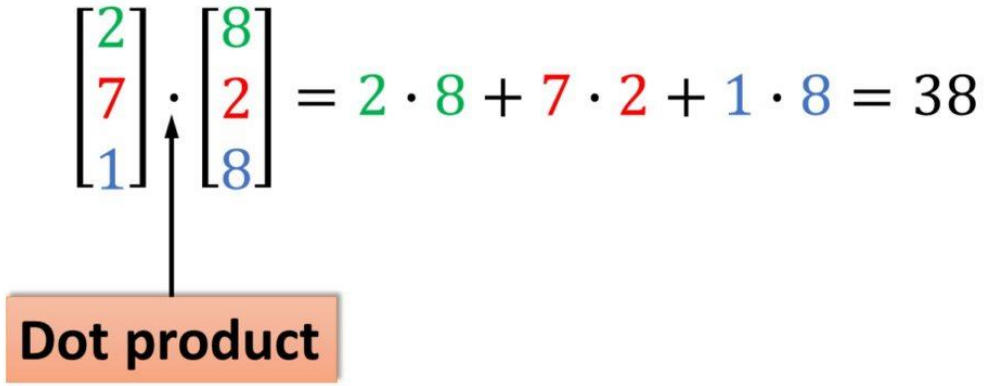
Calculation is done in a **dot product**

The Dot Product

Definition

$\mathbf{a} = \langle a_1, a_2, a_3 \rangle \quad \mathbf{b} = \langle b_1, b_2, b_3 \rangle$

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$



$y = \sigma(w_0 * x_0 + w_1 * x_1 + w_2 * x_2 + \ldots + w_{20,000} * x_{20,000})$

$Y = \sigma(W . X)$

$$Y = \sigma(W . X)$$

This is the logistic regression model we used in practical session 1!

What do the weights mean?

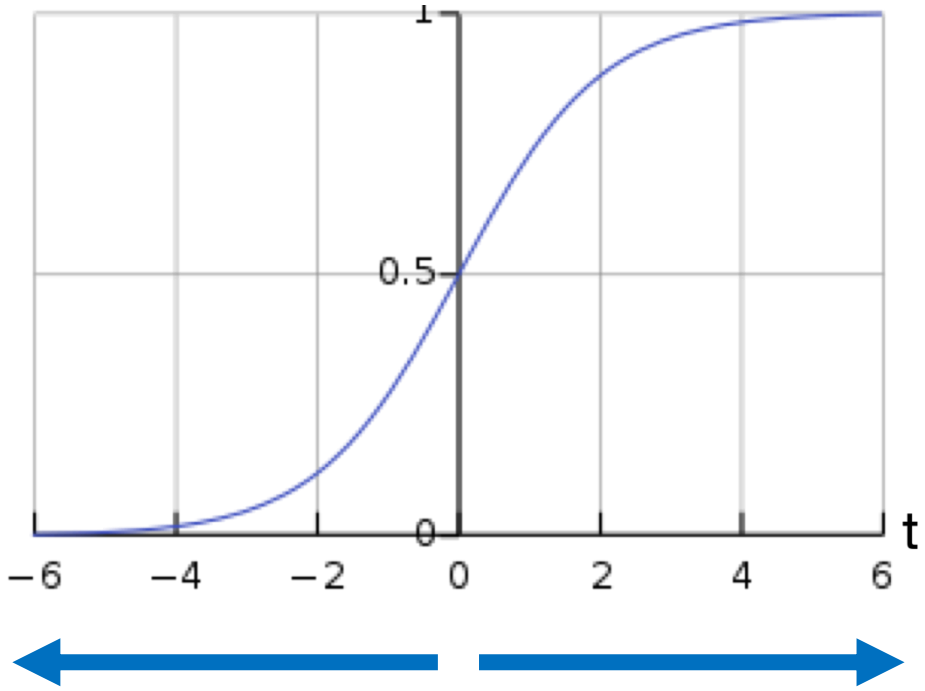
$w_i = 0$ → not relevant

$w_i < 0$ → class 0

$w_i > 0$ → class 1

Large value → larger effect

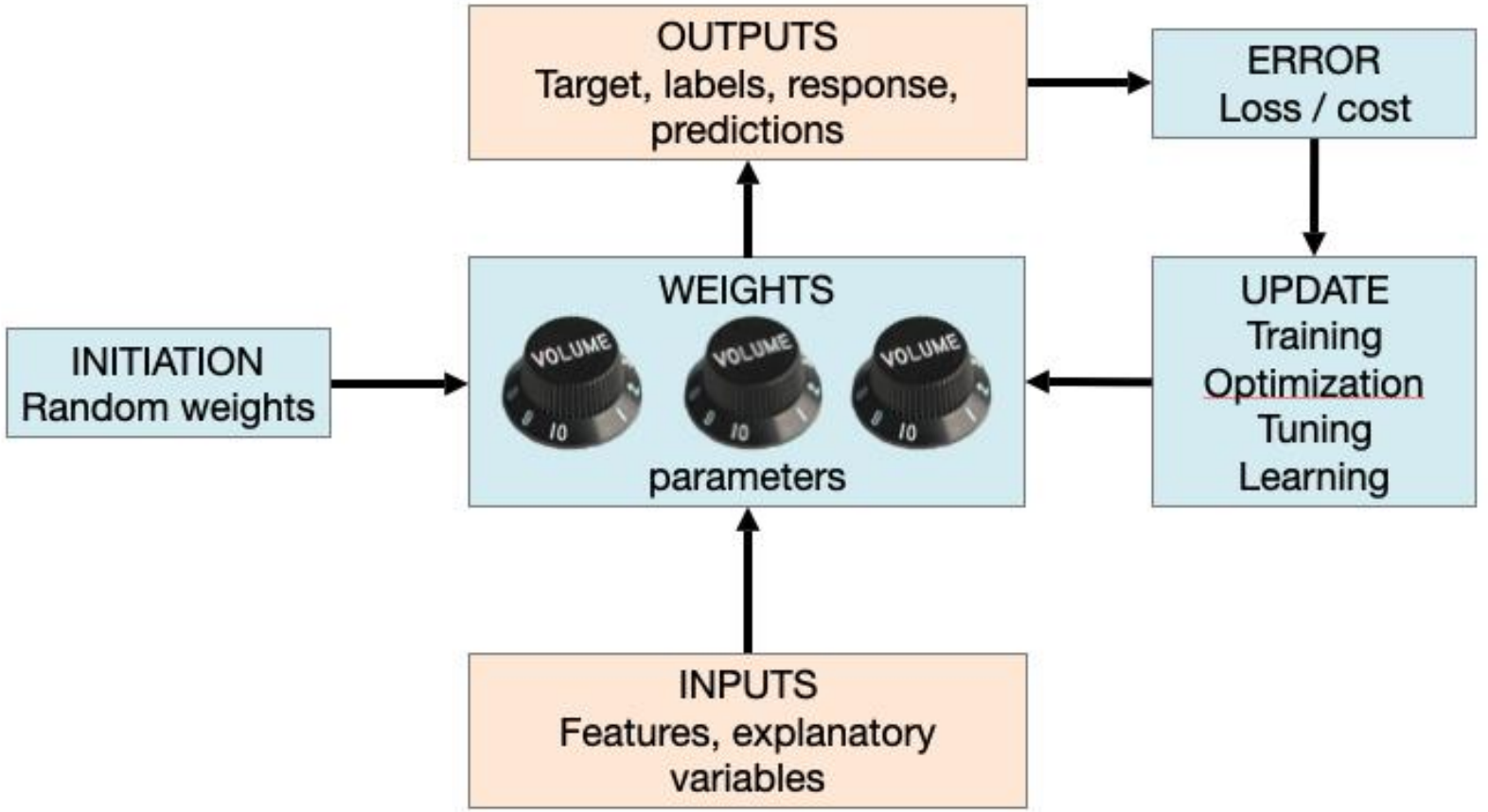
$$y = \sigma(w_0 * x_0 + w_1 * x_1 + w_2 * x_2 + \ldots + w_{20,000} * x_{20,000})$$



Negative values push towards class 0

Positive values push towards class 1

Now how do we learn?



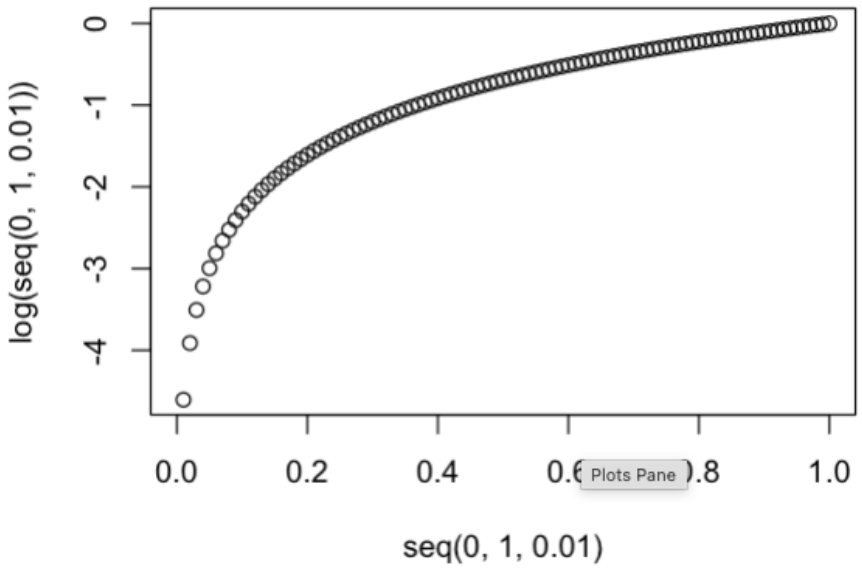
Details on the update tomorrow!

Loss function

$$\mathrm{BCE}(y, \hat{p}) = -\left[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})\right]$$

Large loss if
p_hat close to 0

Large loss if
p_hat close to 1



y … true label [0;1]

p_hat … predicted label [0-1]

Key steps

1. Input (X): cells (observations) x genes (features)
2. Linear layer (X . W): input features → logits
3. Sigmoid transformation σ(): logits → class probabilities
4. Loss calculation: class probabilities + true class labels → error

Practical session 2: pytorch implementation

```
model = SimpleModel(input_dim=???)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters())

outputs = model(X)
loss = criterion(???, ???)

model.train()          # tell pytorch that we will train
optimizer.zero_grad()  # remove previous gradients
loss.backward()        # Do one backwards pass to calculate gradients
optimizer.step()       # Optimize weights
```

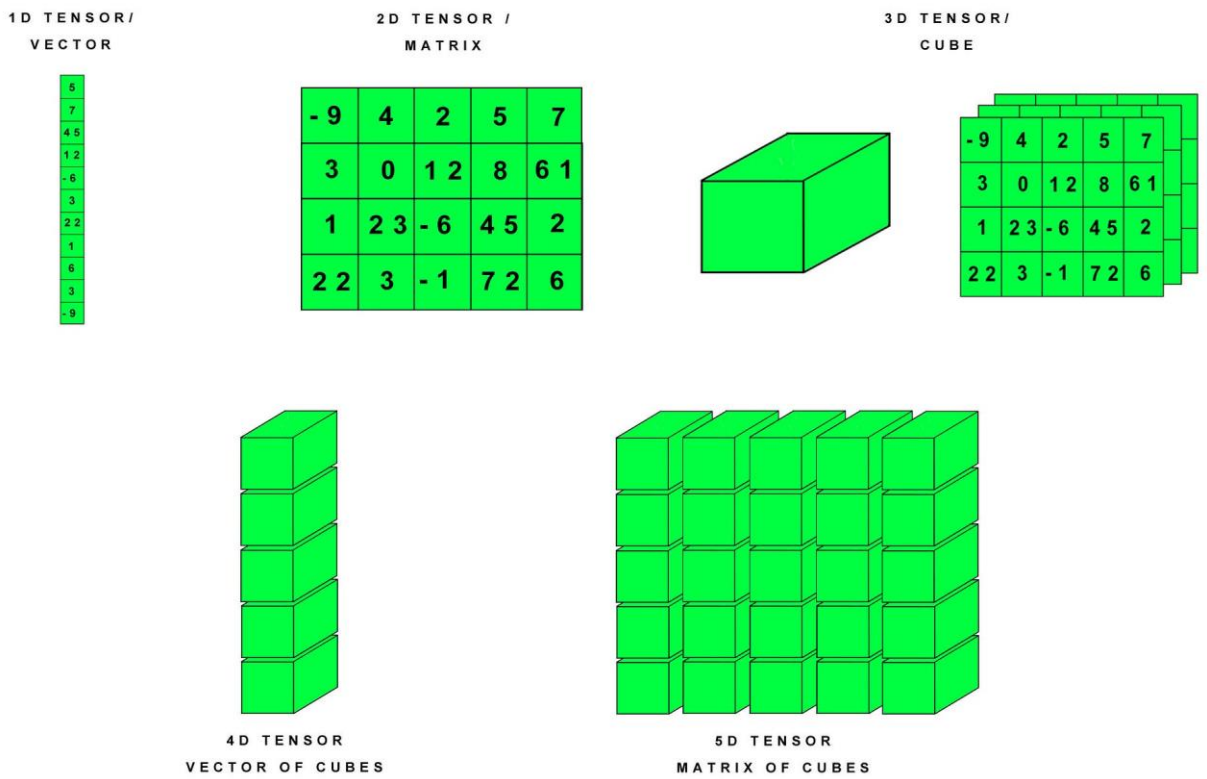
```
class SimpleModel(nn.Module):
    def __init__(self, input_dim):
        super(SimpleModel, self).__init__()
        self.linear1 = nn.Linear(???, 1)

    def forward(self, x):
        x = self.linear1(x)
        return x
```

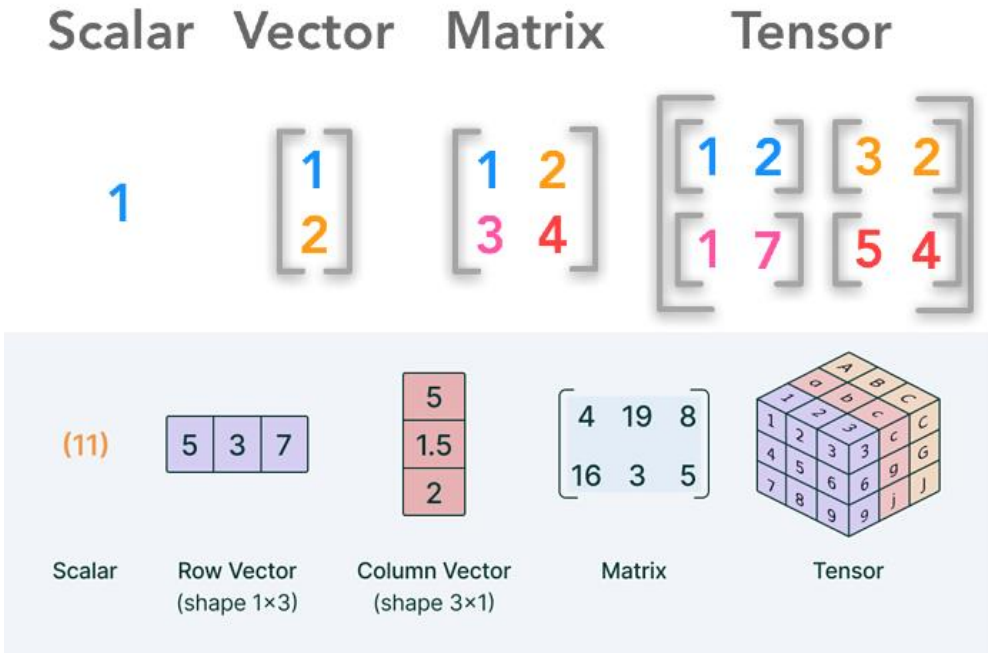
Details tomorrow!

You only define the model (forward pass). Pytorch then optimizes weights (backward pass).

Pytorch and tensors

Tensors are a generic approach to store numbers

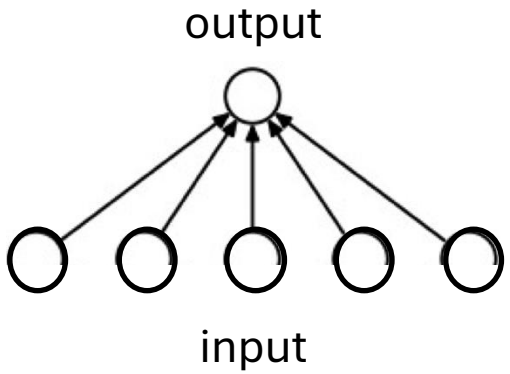




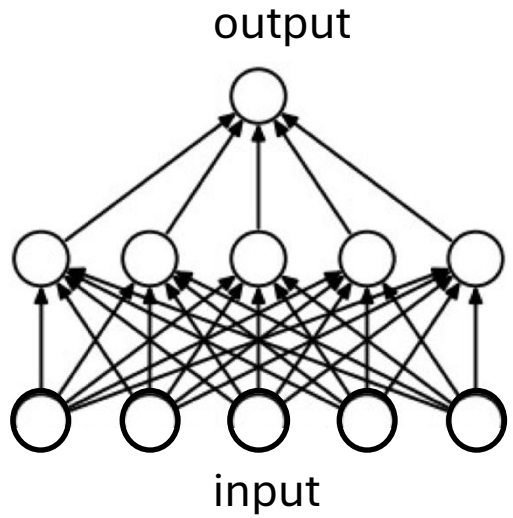
(Deep) neural networks

- In neural networks, we “stack” multiple regression models onto each other
- The output of the first layer are hidden node activations

Logistic regression



Neural network



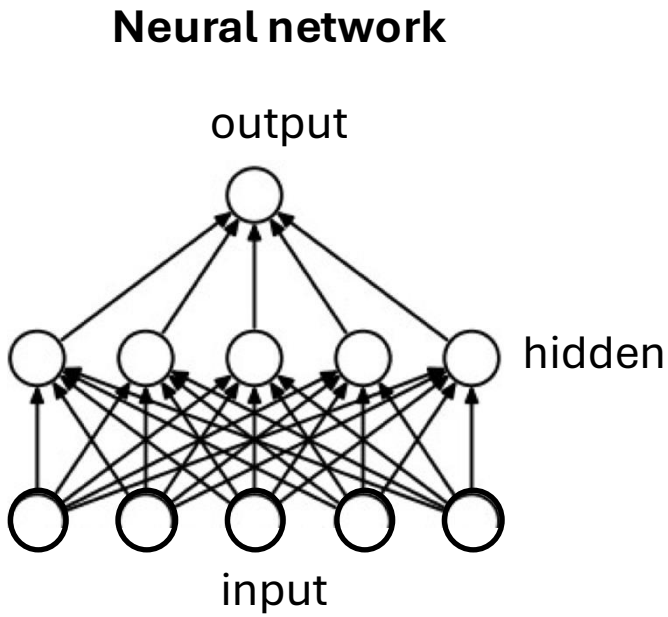
(Deep) neural networks

1. Input (X)

2. Layer 1 (hidden layer)
   - Linear layer ($X \cdot W_1$): 20000 → 50 dimensions
   - Sigmoid layer $\sigma_1()$: hidden node activation (A)

3. Layer 2 (output layer)
   - Linear layer ($A \cdot W_2$): 50 → 1 dimensions (logits)
   - Sigmoid layer $\sigma_2()$: class probabilities

4. Loss calculation



Implementation in pytorch just as for logistic regression

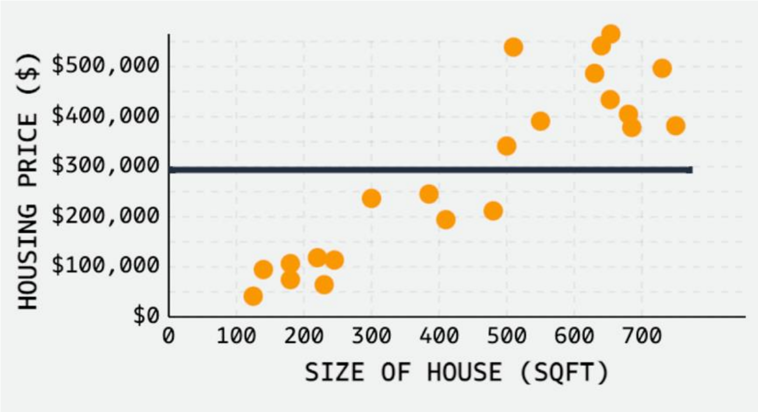
Session 3: weight optimization

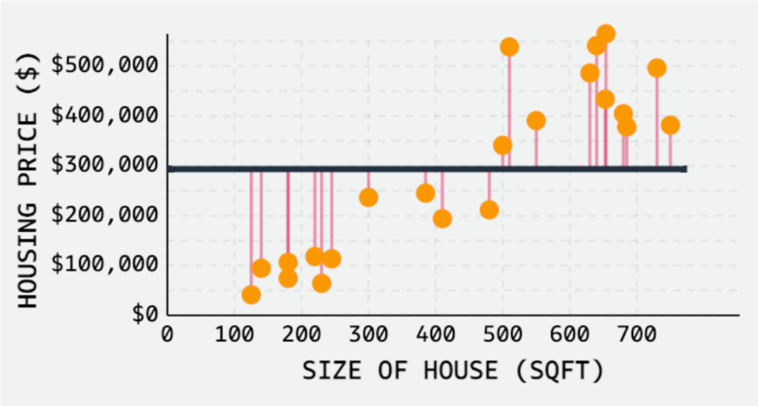
How is a model trained?

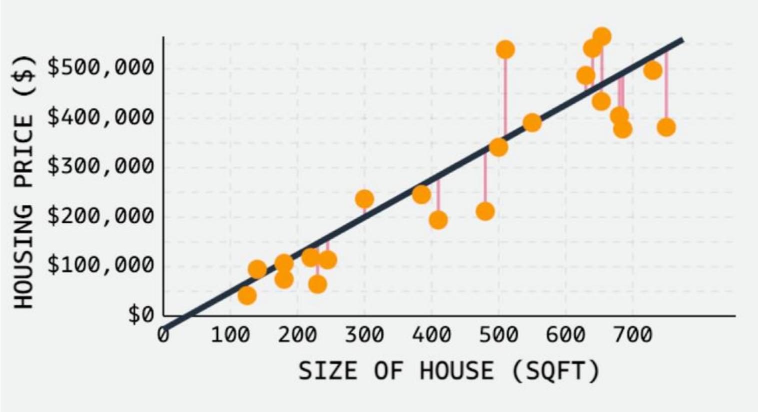
How are weights obtained?

In this example:

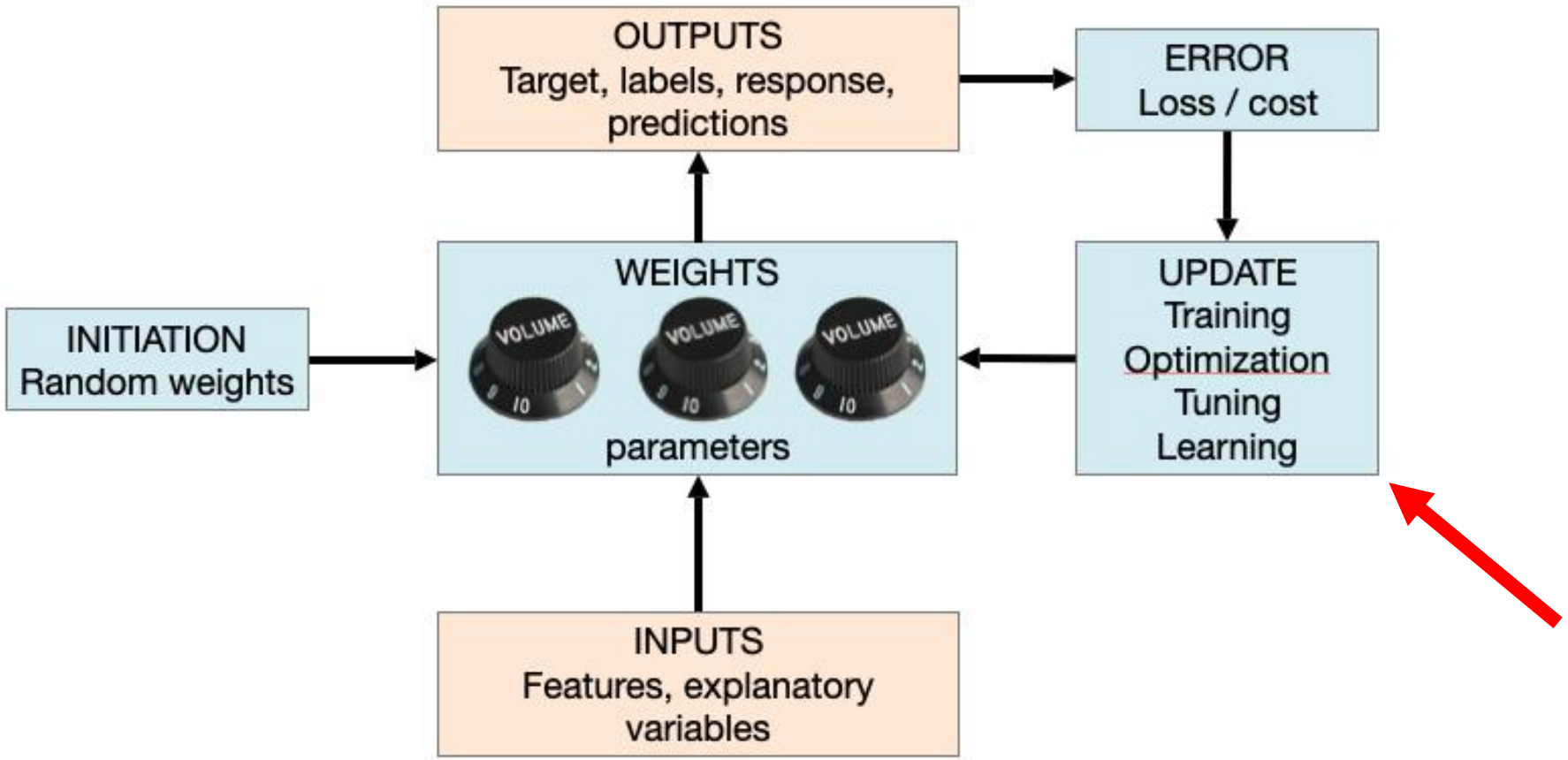
1. Start with slope = 0
2. Measure error
3. Identify slope that minimizes error







Summary

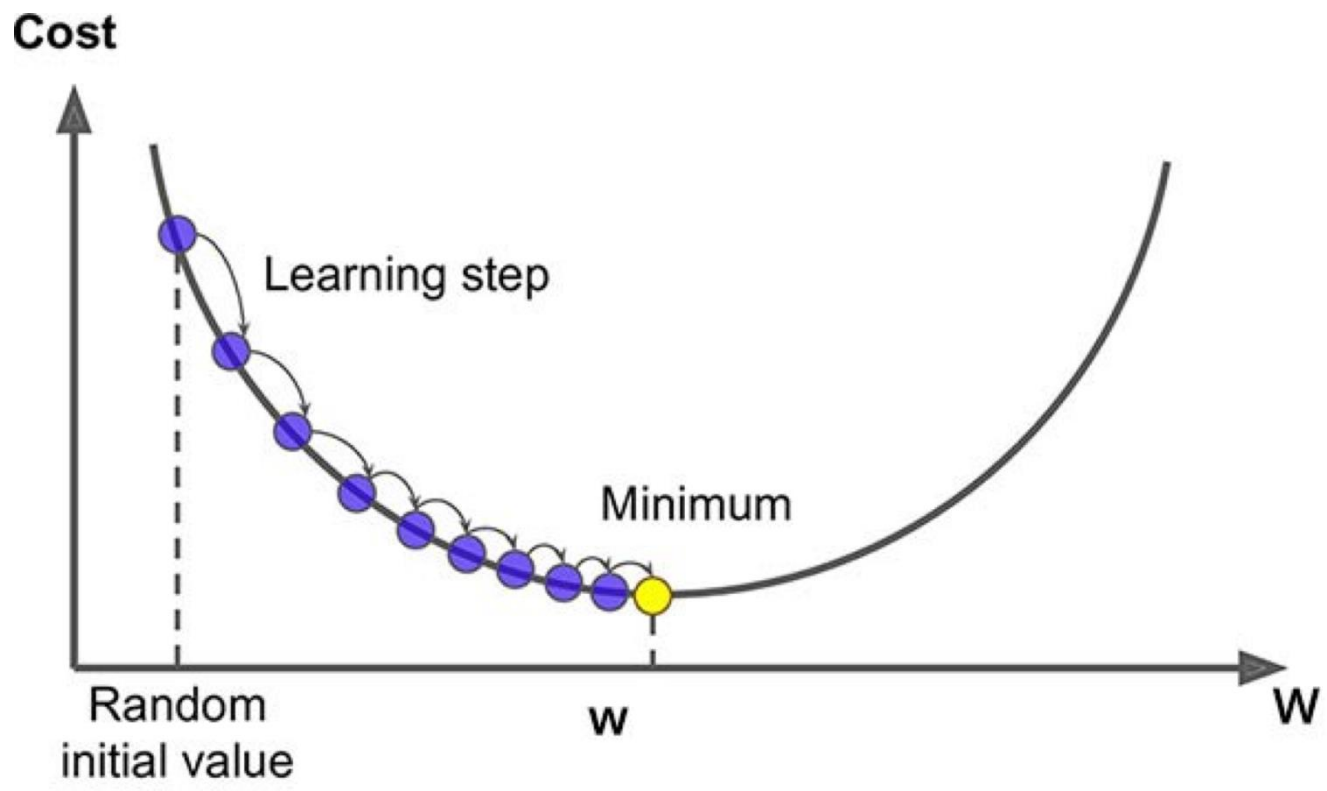

OUTPUTS
Target, labels, response, predictions
ERROR
Loss / cost
WEIGHTS
VOLUME
VOLUME
VOLUME
parameters
INITIATION
Random weights
UPDATE
Training
Optimization
Tuning
Learning
INPUTS
Features, explanatory variables

How do we find the right weights in ML?

- Use labeled dataset **we know the solution!**

Generic process:

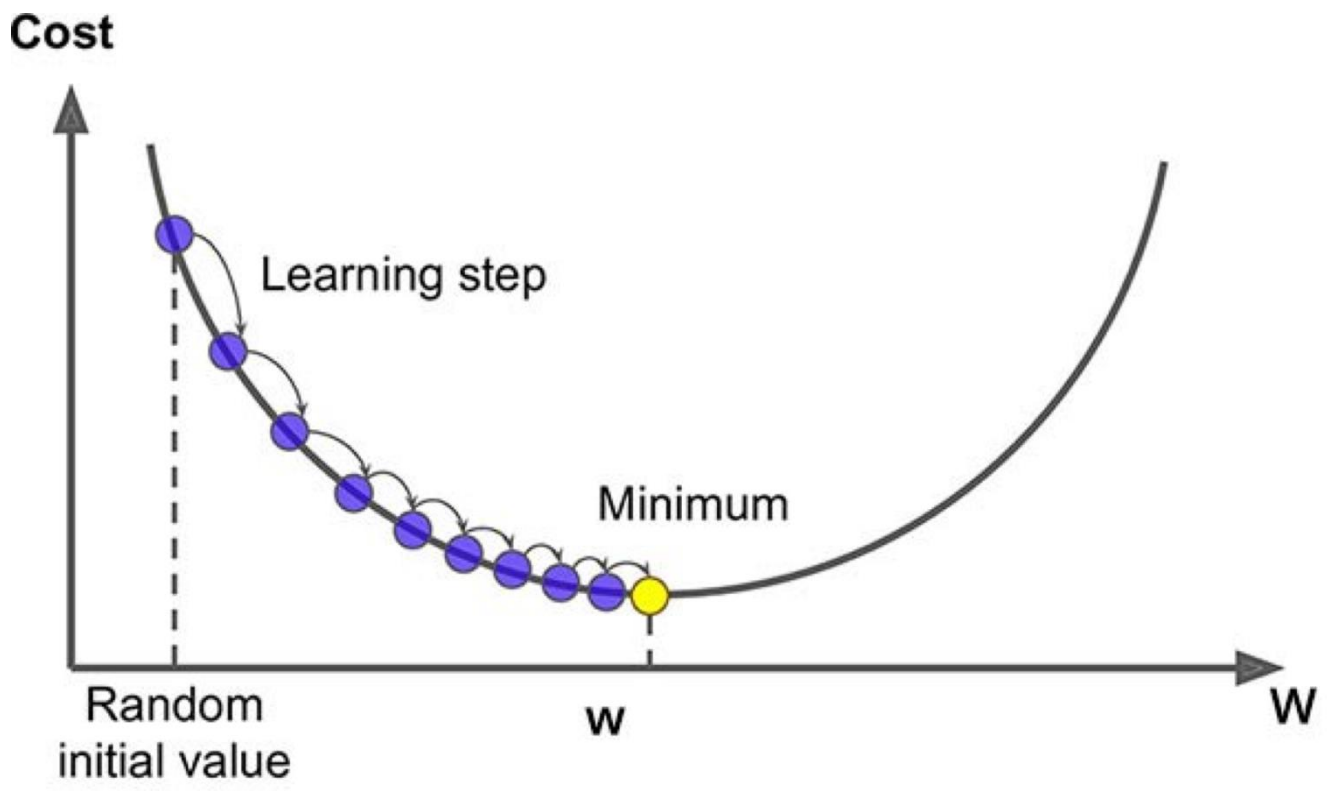
- Start with **random weights**
- Measure cost /loss (error)
- Find weights that minimize cost



How do we find the right weights in ML?

- Differential calculus?
- 1st derivative is the slope

THIS IS LEARNING!



Key steps: backpropagation

1. Input (X)
2. Linear layer (X . W)
3. Sigmoid transformation σ()
4. Loss calculation

1. Inputs are fixed.
2. How much would logits change when each weight changes?
3. How much would class probabilities change when logits change?
4. How much would the loss change when the class probabilities change?

Requirement: calculations are differentiable

Summary

- Weight updates use differential calculus (“gradients”)
  → how does a change in a weight affect the loss?

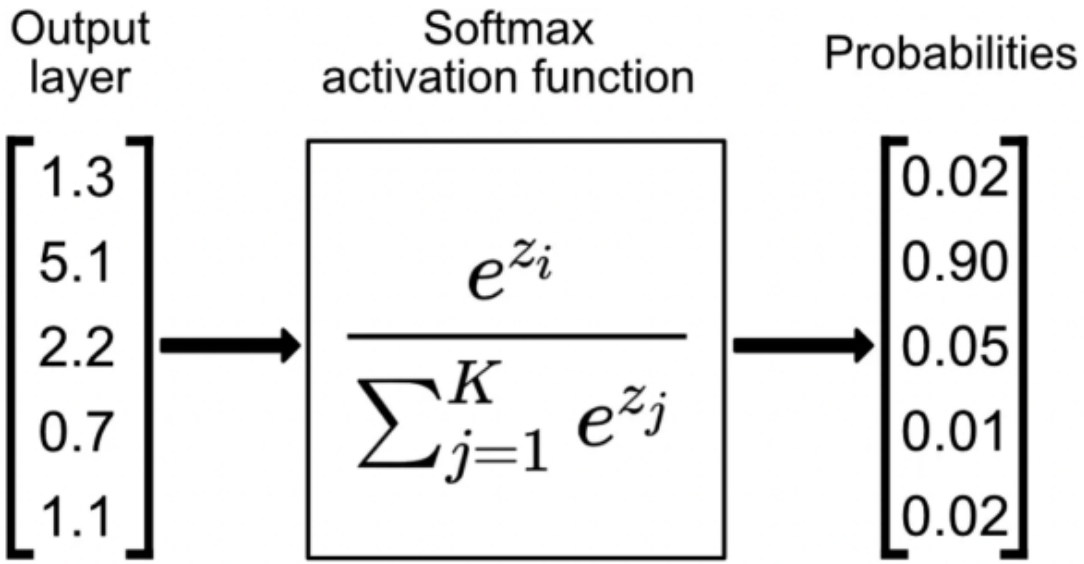
- Training contains the following steps
  1. Forward pass calculates class probabilities
  2. Backward pass calculates gradients
  3. Weights are updated based on gradients

- All of this happens during **training**
  (not while testing and prediction)

Practical session 3: multi-class predictions

Softmax for multi-class probabilities

- Converts real numbers into class probabilities
- Probabilities that sum to 1

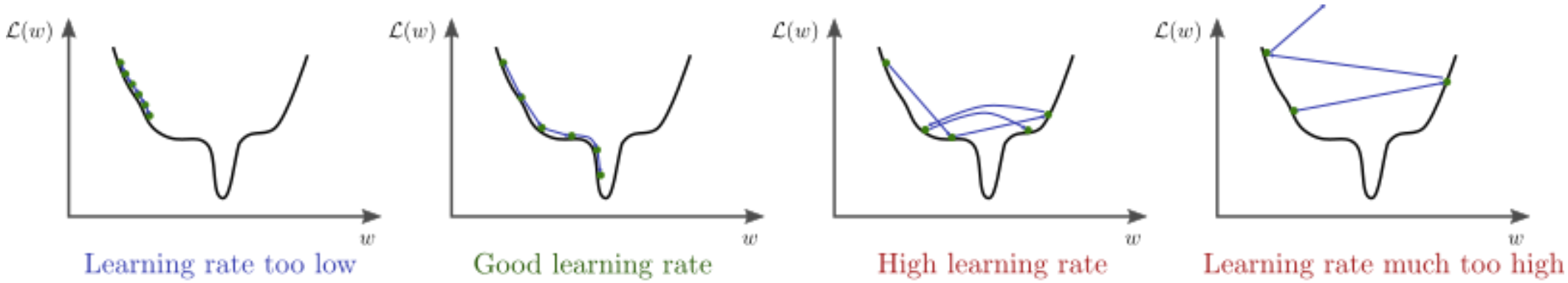


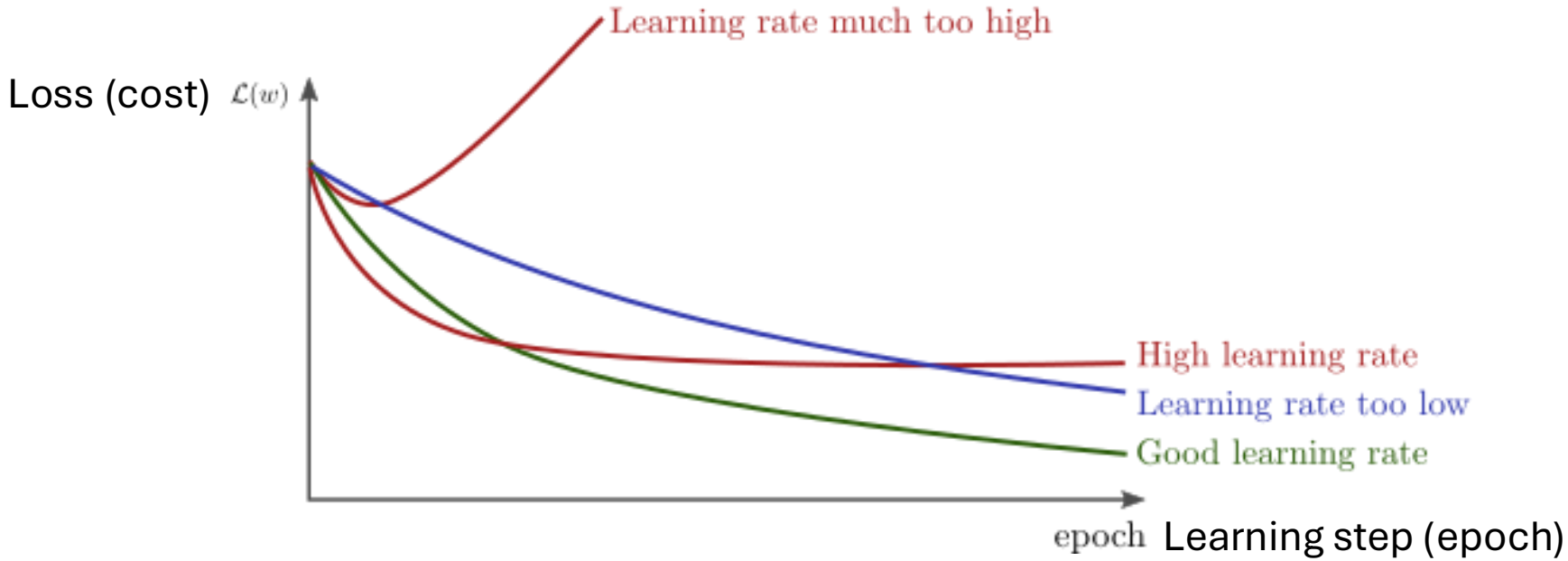
True labels

	Cell type 1	Cell type 2	Cell type 3	Cell type 4	Cell type 5
Cell 1	1	0	0	0	0
Cell 2	1	0	0	0	0
Cell 3	0	1	0	0	0
…	…	…	…	…	…

Session 4: Hyperparameters

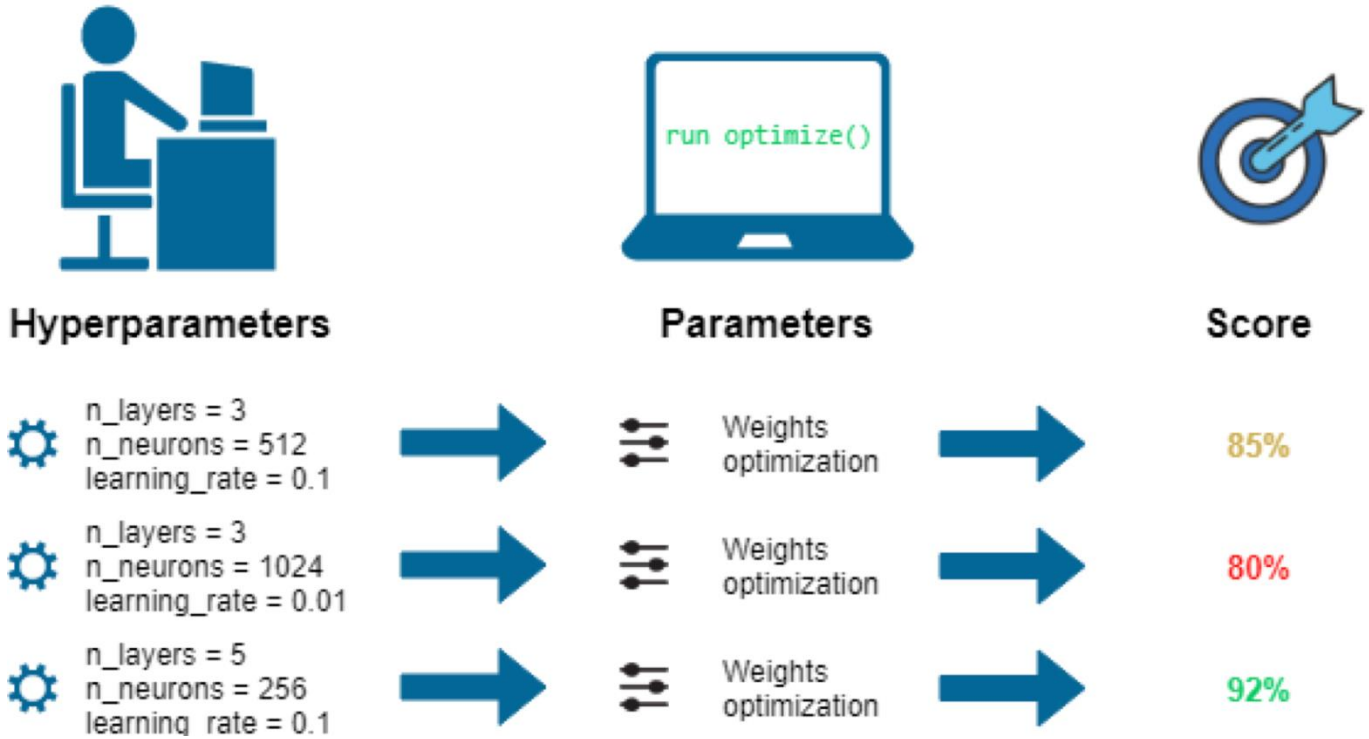
How big are our learning steps?





Hyperparameter optimization

- in addition to weights (parameters)
- other choices are “hyperparameters”
- also need optimization

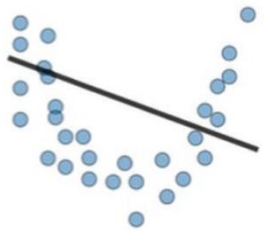
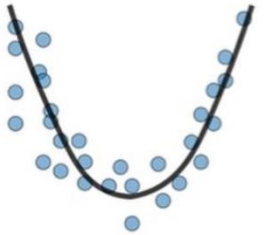
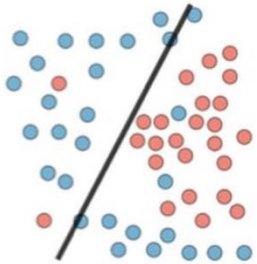
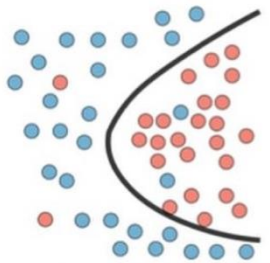
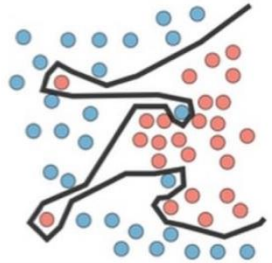
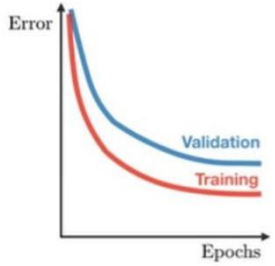
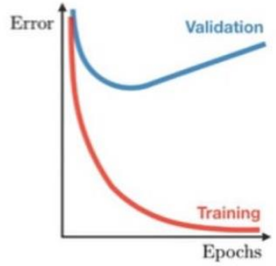


Under- / Overfitting

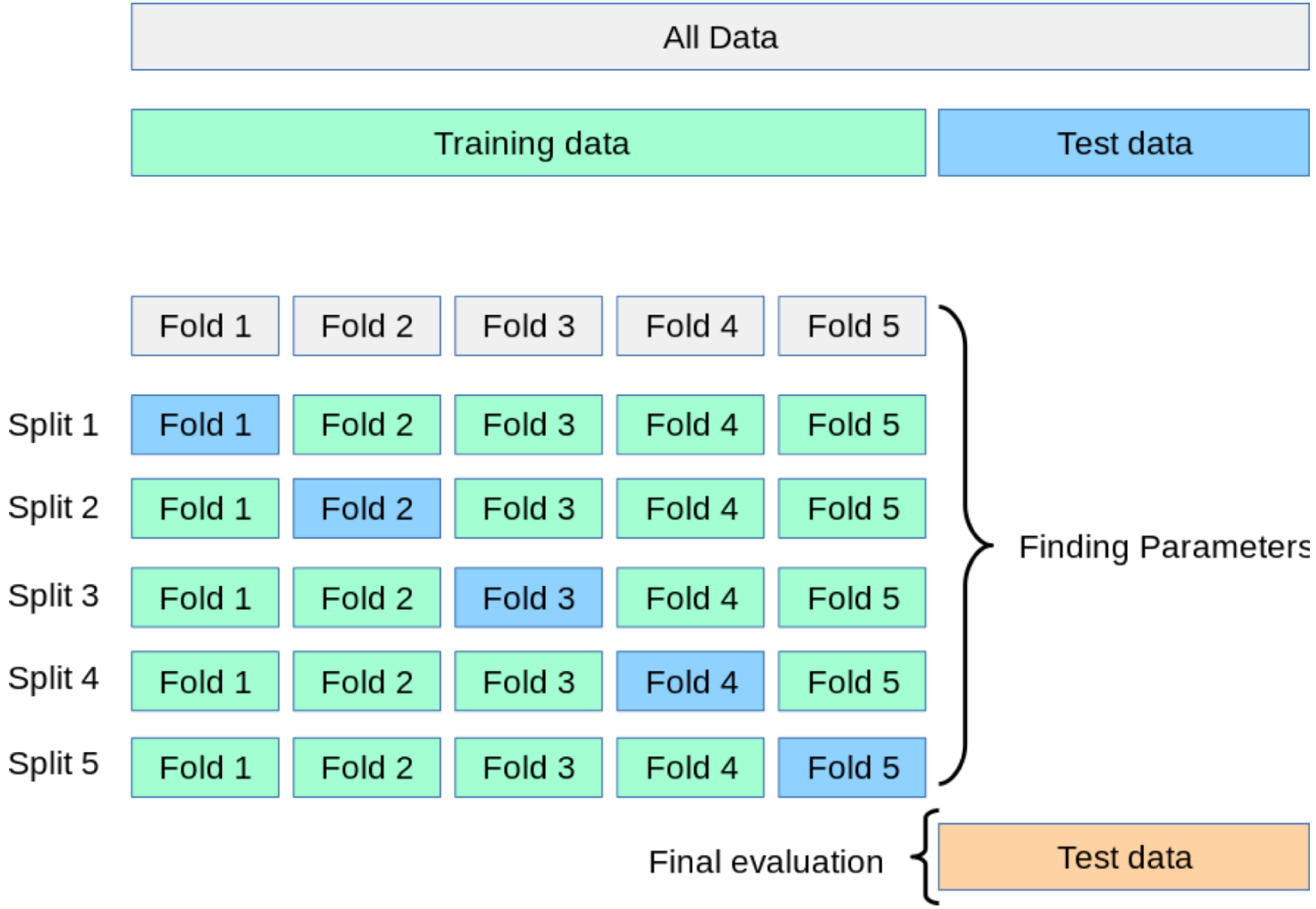
- **Underfitting**
  → model is not good enough

- **Overfitting**
  → model is too good
  (on training data)

□ **Bias/variance tradeoff** — The simpler the model, the higher the bias, and the more complex the model, the higher the variance.

	Underfitting	Just right	Overfitting
Symptoms	• High training error • Training error close to test error • High bias	• Training error slightly lower than test error	• Very low training error • Training error much lower than test error • High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	• Complexify model • Add more features • Train longer		• Perform regularization • Get more data

Cross-validation for hyperparameter optimization

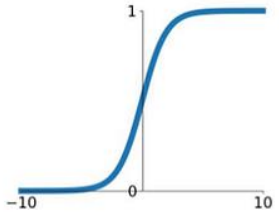


Activation functions: another hyperparameter

- Non-linear transformations introduce non-linearity
- Predictions are not simple additions of features x weights

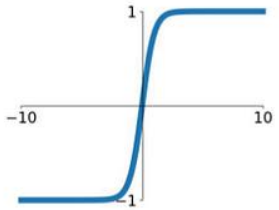
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



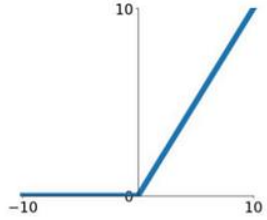
tanh

$$\tanh(x)$$



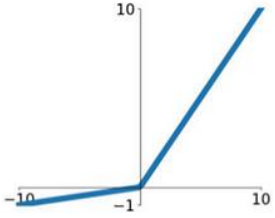
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

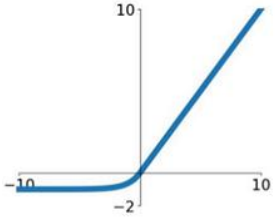


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



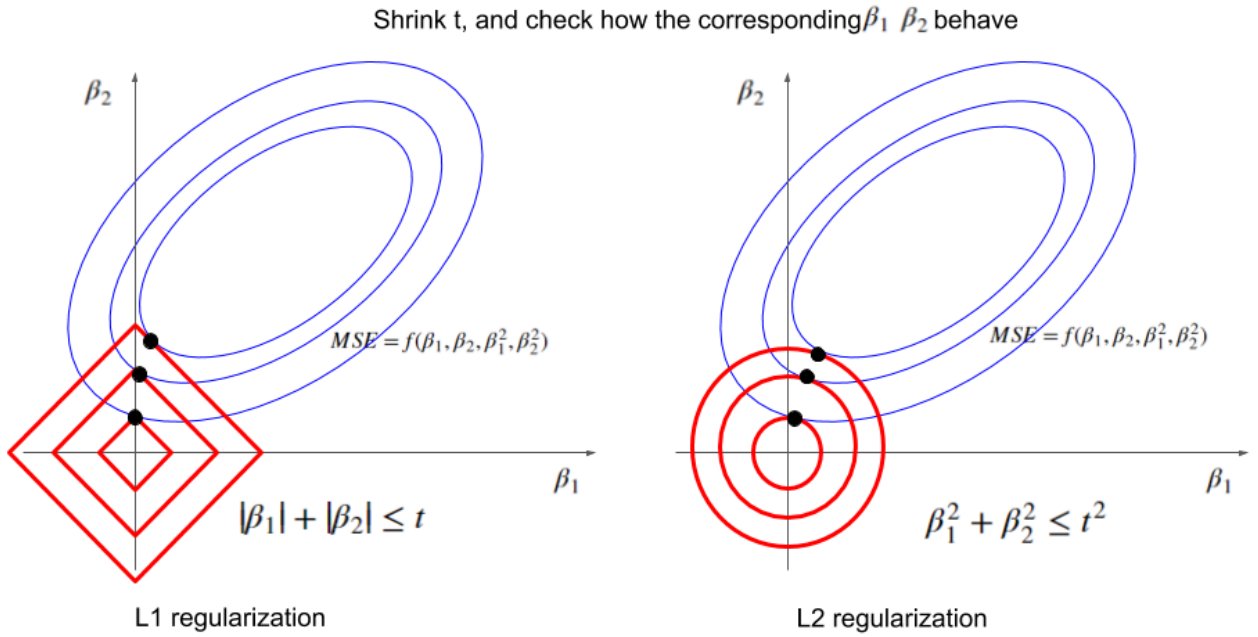
Weight regularization

Commonly used:

- l1 loss
- l2 loss ("ridge")
- Both ("elastic net")

Why?

- avoids overfitting
- select predictive features



Adds weights to loss:

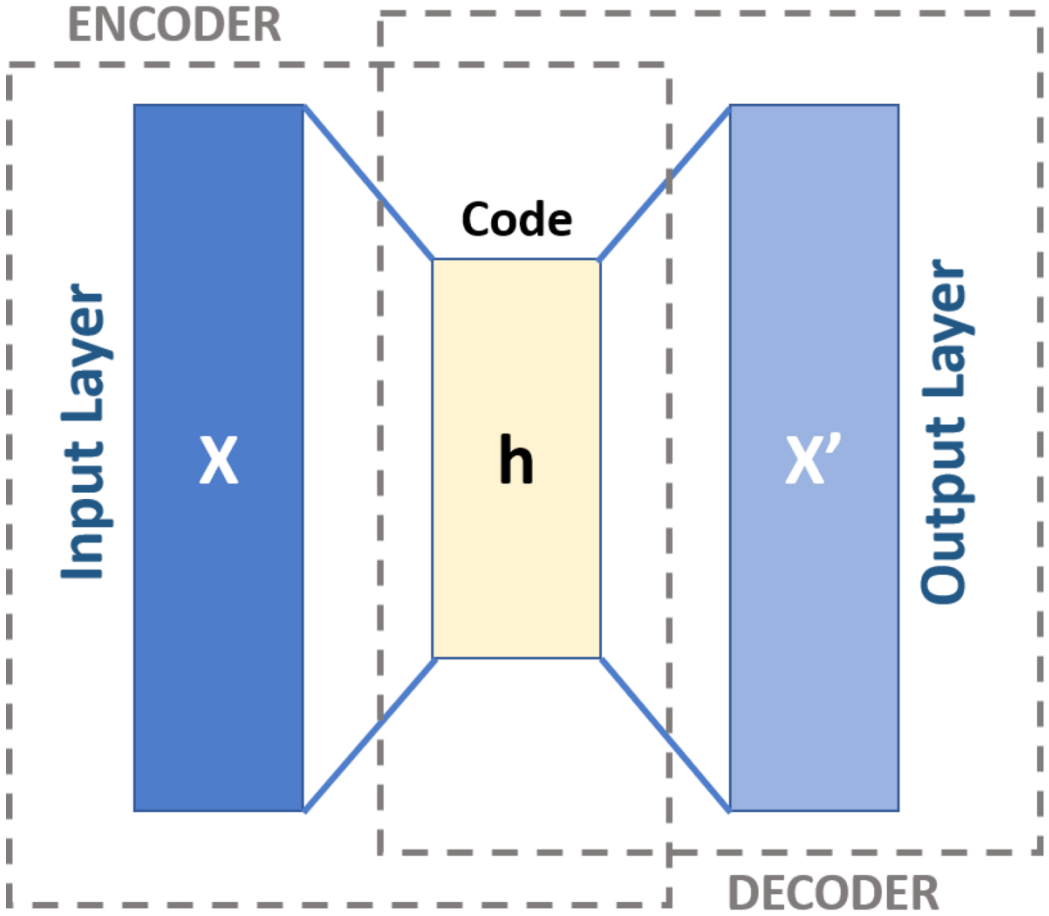
loss = loss + lambda x sum(f(weights))

The regularization term affects weight optimization!

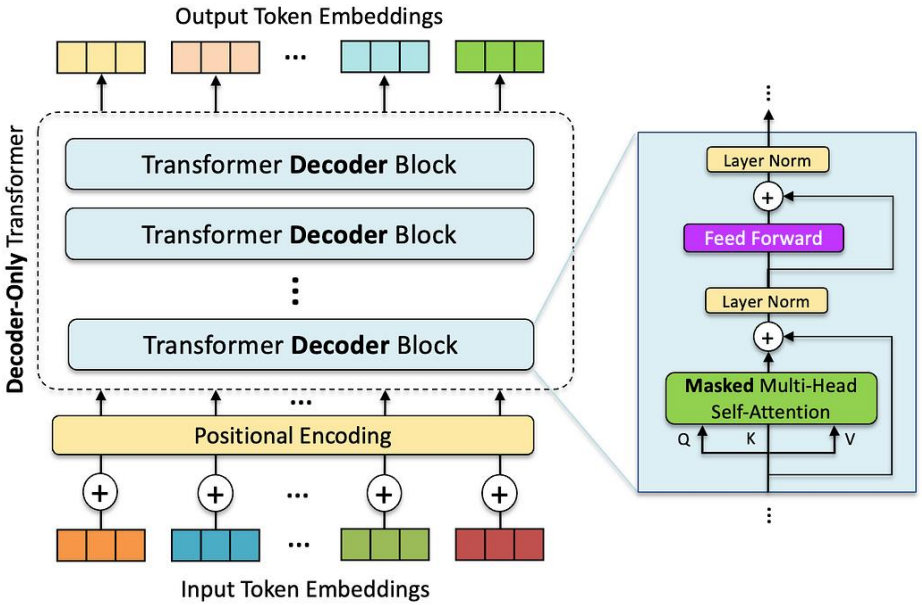
Session 5: Other models and interpretability

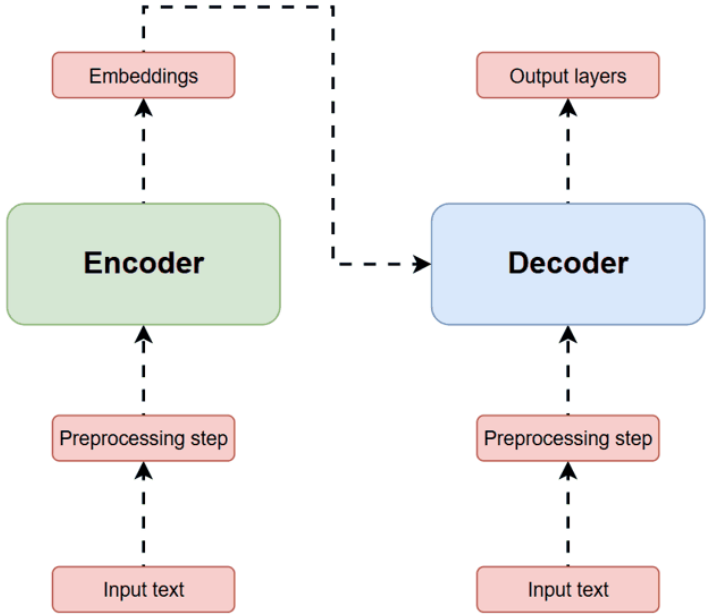
Autoencoders

- Model is trained on input to re-produce input as output
- Compresses information in a latent dimension
- Similarity to dimensionality reduction



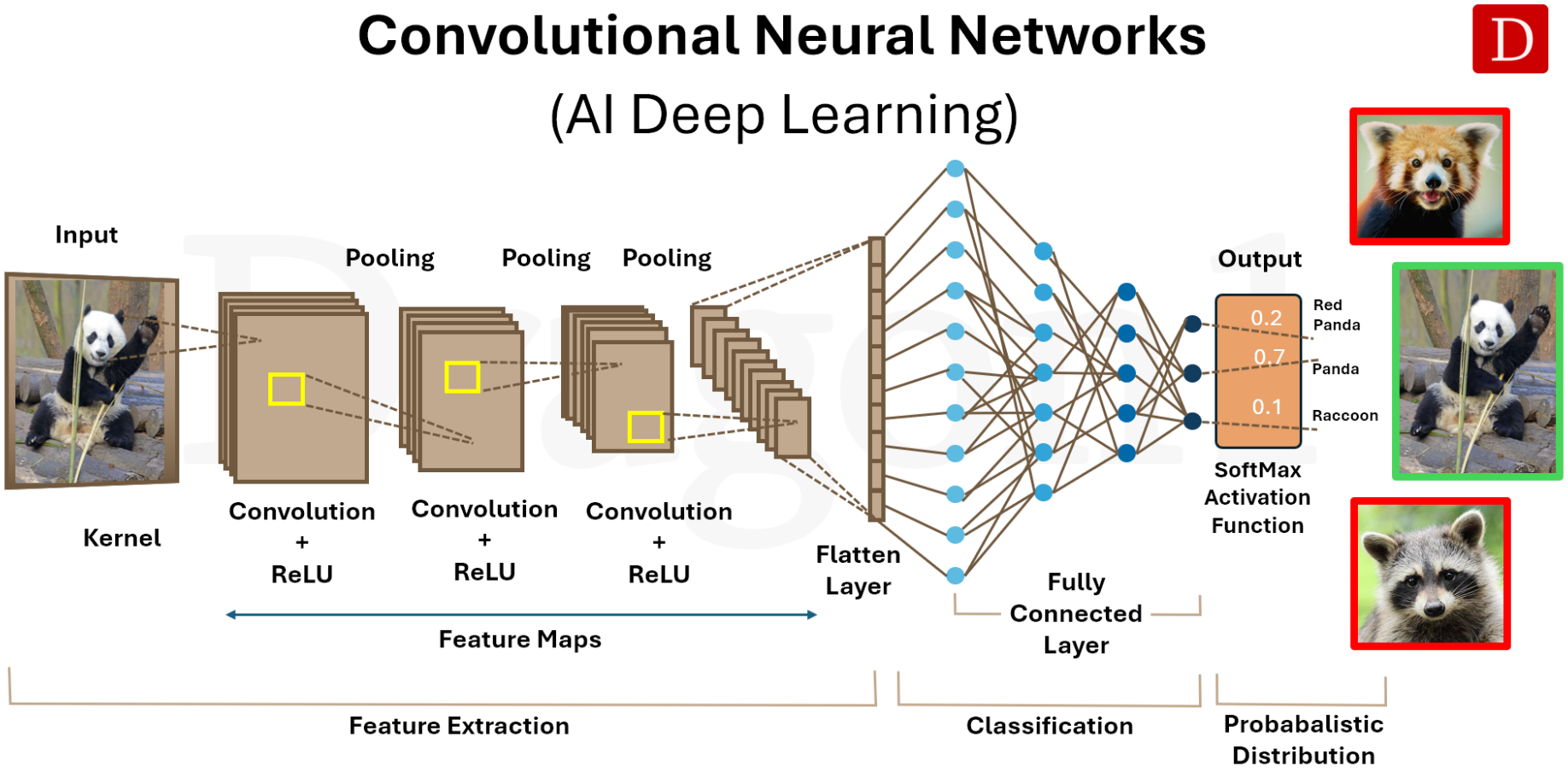
Transformers





https://ig.ft.com/generative-ai/

Convolutional neural networks

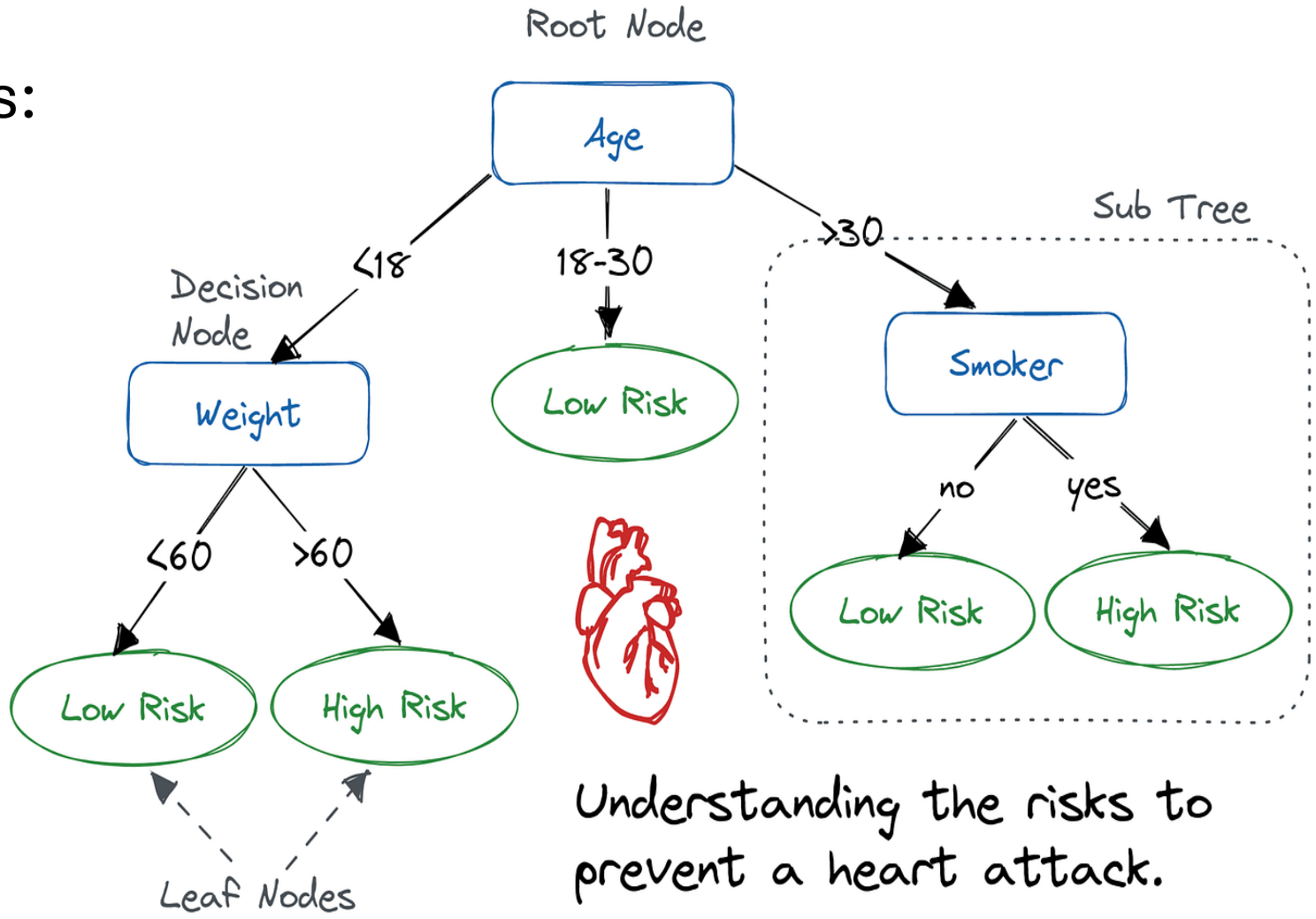


Non-neural networks

Decision trees

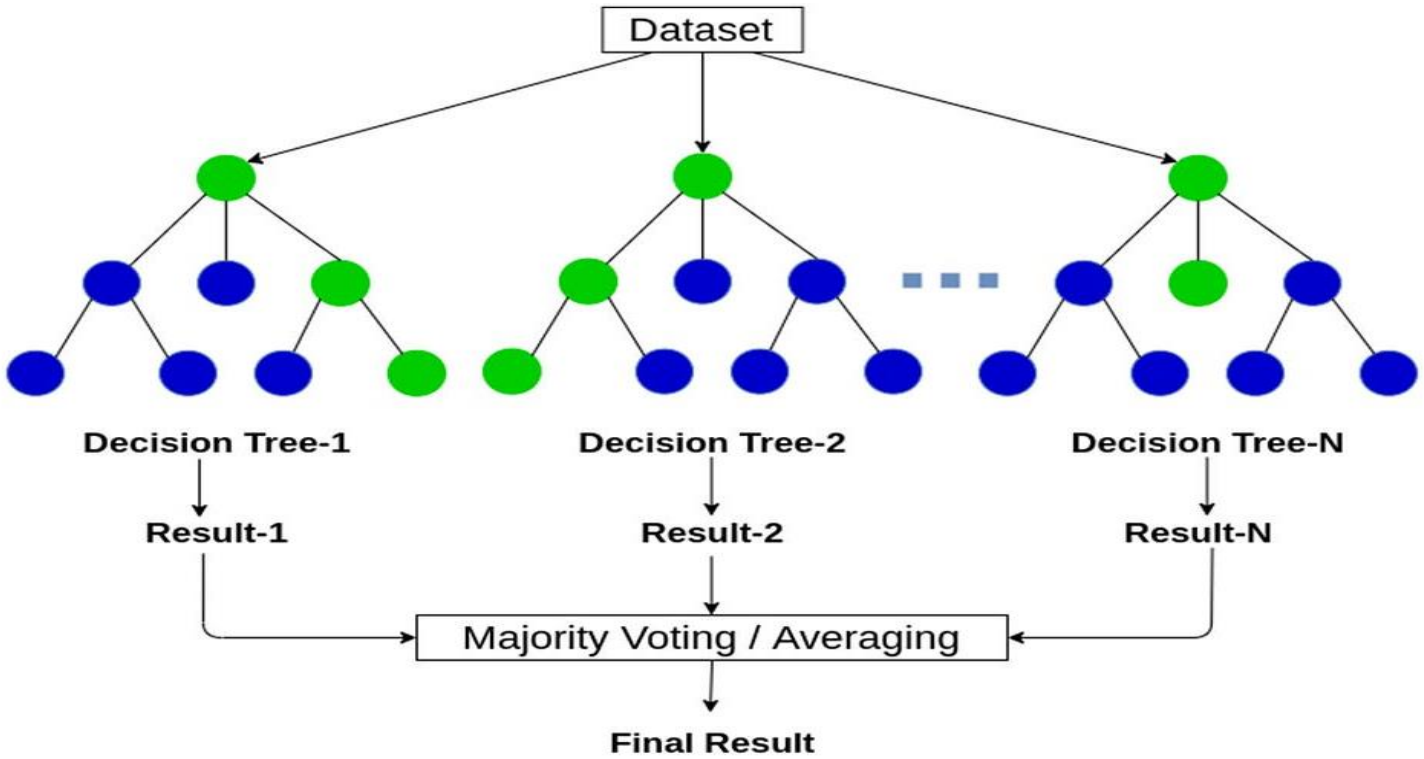
The parameters / weights:

- Which variable to test?
- Which values?



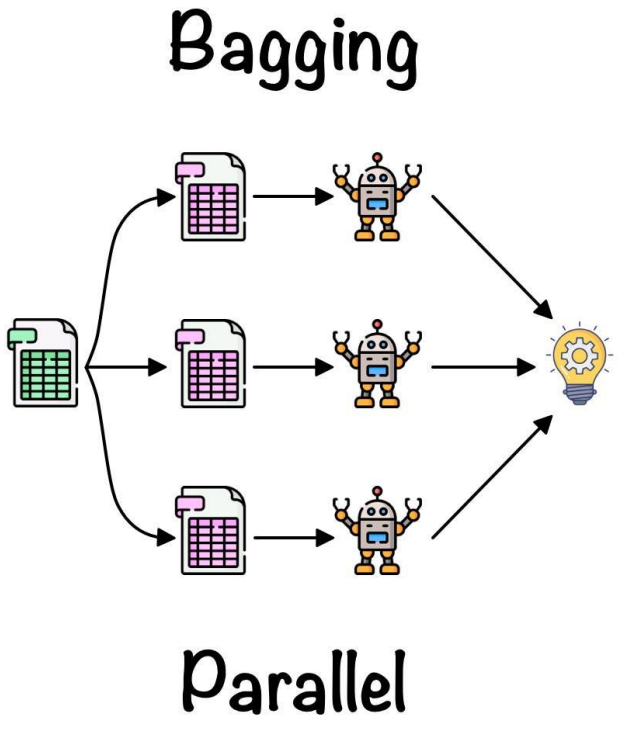
Random forest: ensemble learners

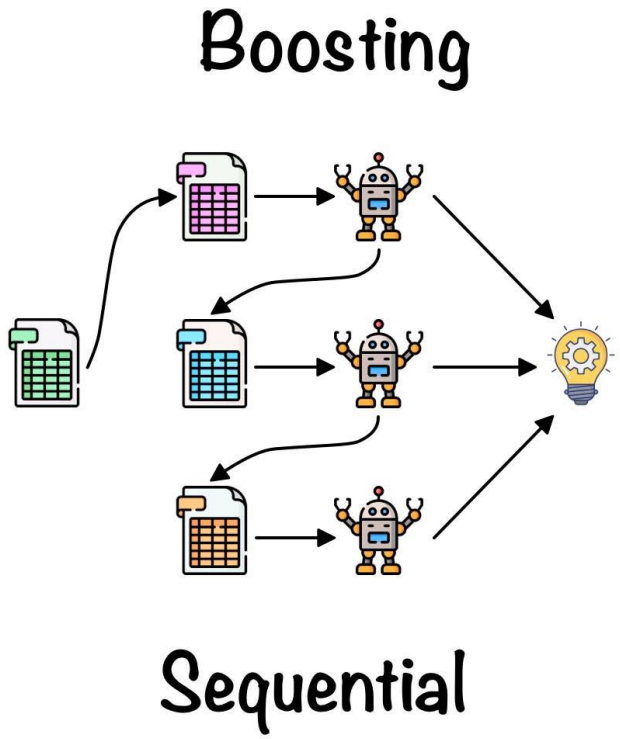
Many trees make one forest!



Bagging and boosting

- Bagging (many strong learners)
- Boosting (many weak predictors): one model trained on errors of other model





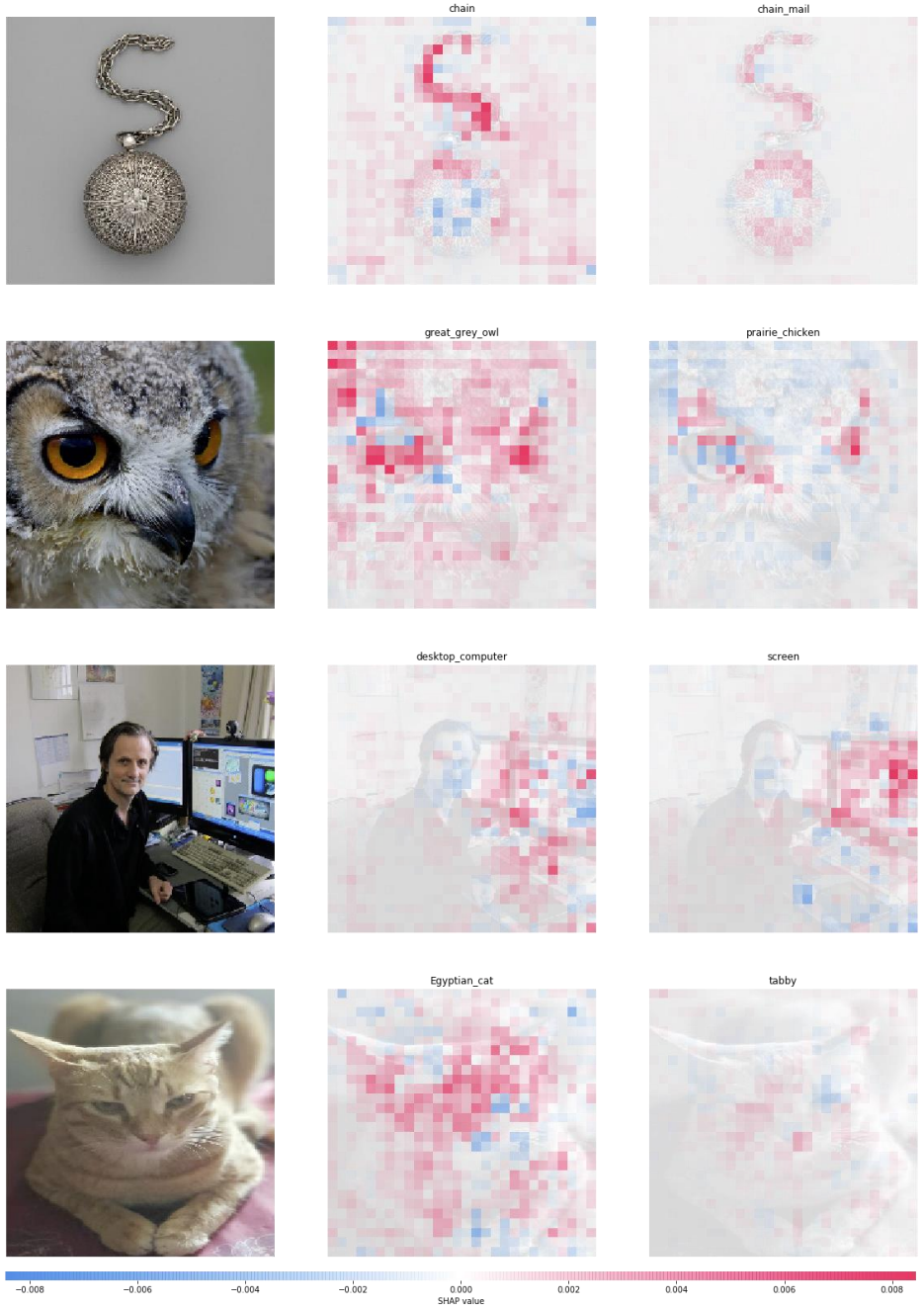
Random forest and the related XGBoost models have good performance “out of the box”

What to choose?

- Depends on the data
- Research: compare different algorithms
- Usually standard choices work well:
  - logistic regression / elastic net
  - random forest / XGBoost
- Neural networks for dedicated applications or complex problems (images, text,…)

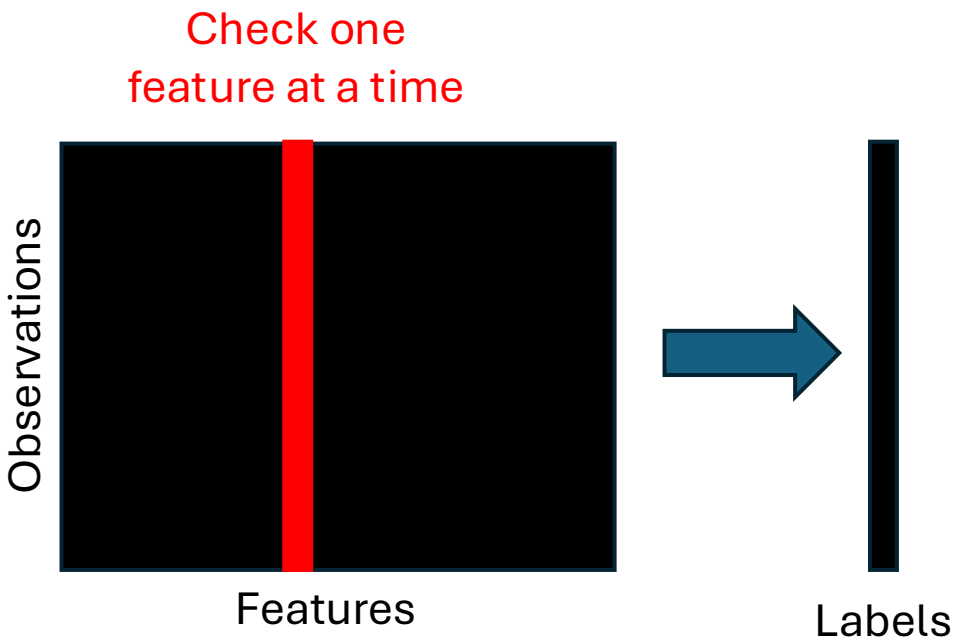
Interpretability

- What has the network learned?
- Common approach: Which features are important?



Interpretability

- Feature permutation: shuffle feature values
- Feature perturbation: set features to 0

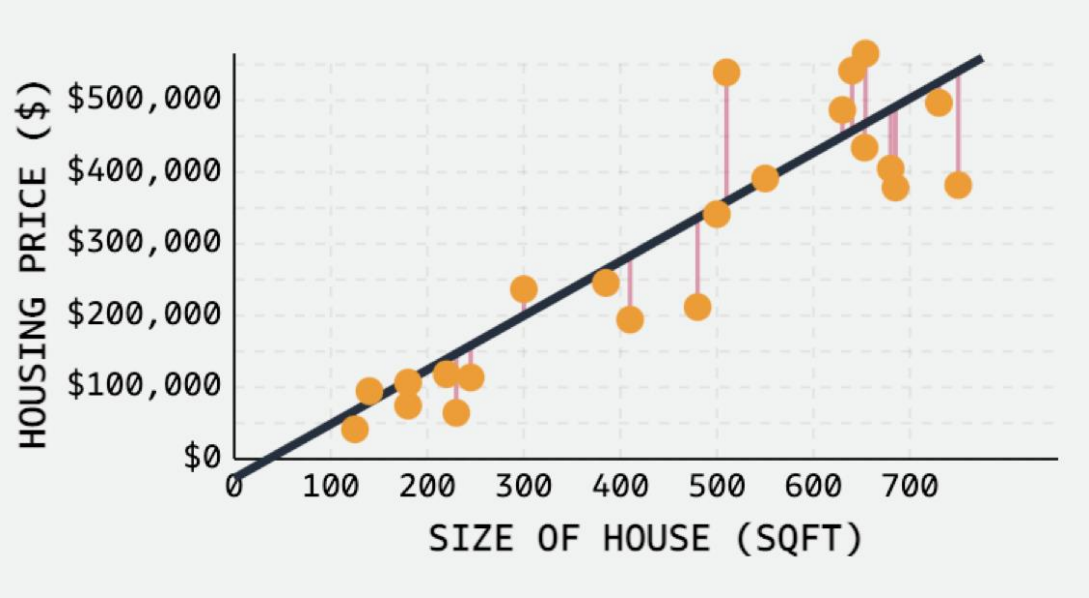


Big research field:

- Feature *interactions*?
- What has the model learned *internally*?
- …

Machine learning vs Statistics

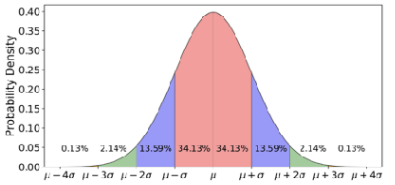
- Statistical inference is focused on the weights / parameters β
  - Builds on assumptions that need rigorous testing (normal distribution,…)
  - If the model fits well, we can interpret the parameters β
  - We can obtain confidence estimates and p-values for parameters
- Machine learning is focused on prediction
  - Assessed with train/test split (otherwise no assumptions to test)
  - Weight interpretation is less reliable
  - Model usually more complex - difficult to interpret the weights



- Statistics and machine learning are related and overlapping, but:
  - Statistical models focus more on **inferring** something about the population
  - Machine learning models focus more on **predicting** new observations

- Both models look at the same formula: y = β0 + β1*x1

- **Statistics:** What does β1 tell us about the population?



X → Y

- **Machine learning:** Which β1 most accurately predicts new y?
  - We can test this!
  - Assumptions are less important
  - More "creative" algorithms

X → ? → Y

- For an interesting read, see: Leo Breiman: Statistical Modeling: The Two Cultures

Final nodes

- Goal is understanding
- For many approaches (cross-validation, interpretation), dedicated functions / packages exist.
- You need to understand what the functions are doing!

Data leakage

For simplicity, we overlooked some approaches that should be done for 100% proper ML.

Be careful when performance is too good!

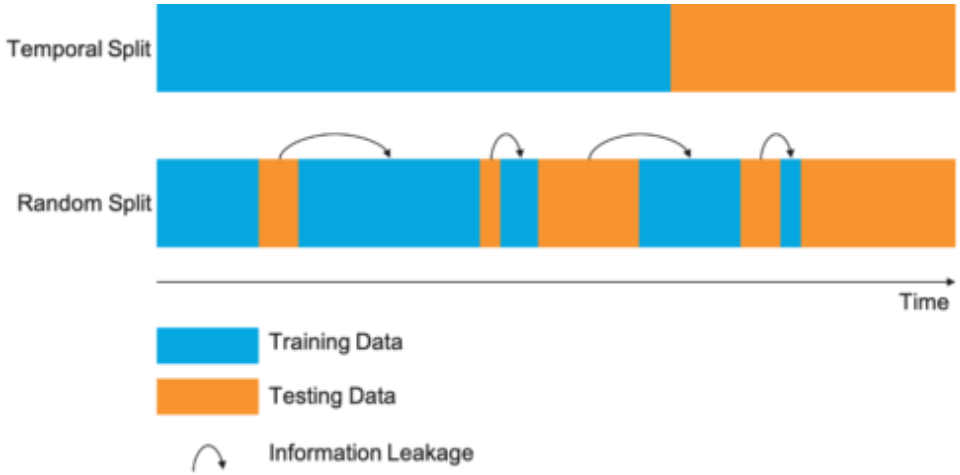
E.g.: no preprocessing of the dataset before training:

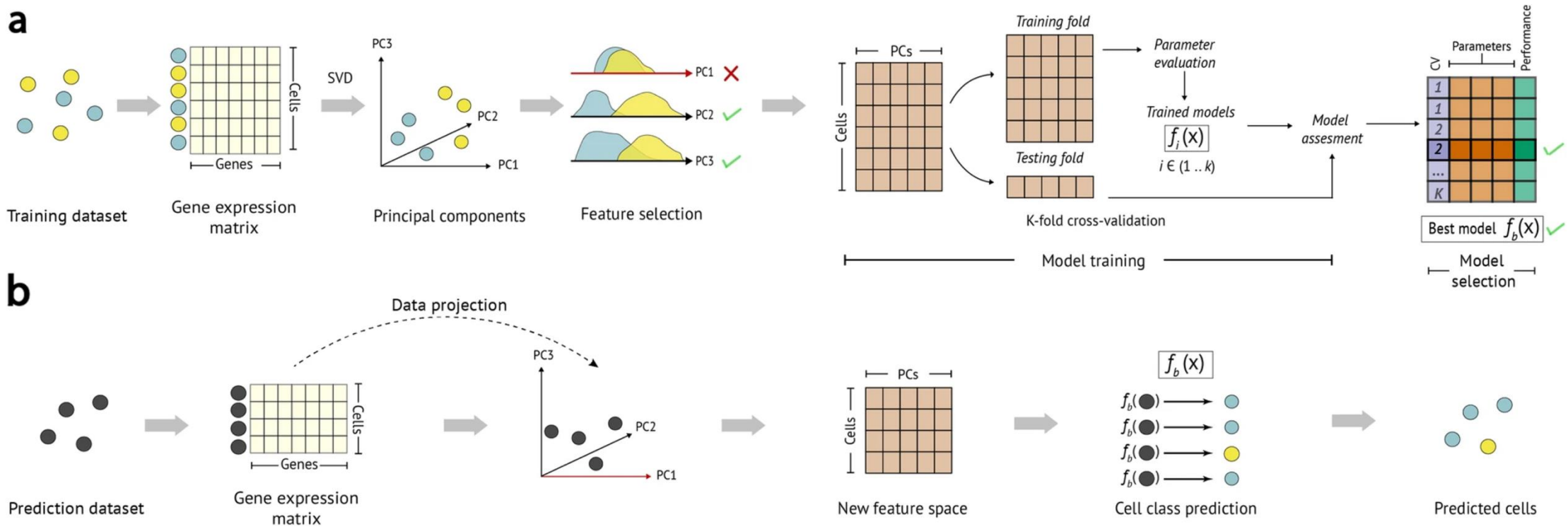
- Preprocessing only in training data
- Filter features only in training data
  (we filtered lowly expressed genes before)

Structure of data not accounted for. For example, multiple samples from the same:

- Patient (clinical samples)
- Chromosome (protein duplicates)
- ...

--> Does not test true generalizability!




a
Training dataset
Cells
Genes
Gene expression matrix
SVD
PC3
PC2
PC1
Principal components
PC1
PC2
PC3
Feature selection
PCs
Cells
Training fold
Testing fold
K-fold cross-validation
Parameter evaluation
Trained models
$f_i(x)$
$i \in (1 .. k)$
Model assesment
Model training
CV
Parameters
Performance
1
1
2
2
...
K
Best model $f_b(x)$
Model selection
b
Prediction dataset
Data projection
Cells
Genes
Gene expression matrix
PC3
PC2
PC1
PCs
Cells
New feature space
$f_b(x)$
Cell class prediction
Predicted cells

Computational Systems Biology Group

Expertise:

- Transcriptome, epigenome, proteome,…
- Omics integration
- Single-cell and spatial analysis
- Networks
- Interpretable machine learning

For more details, visit:

https://plus.ac.at/fortelny