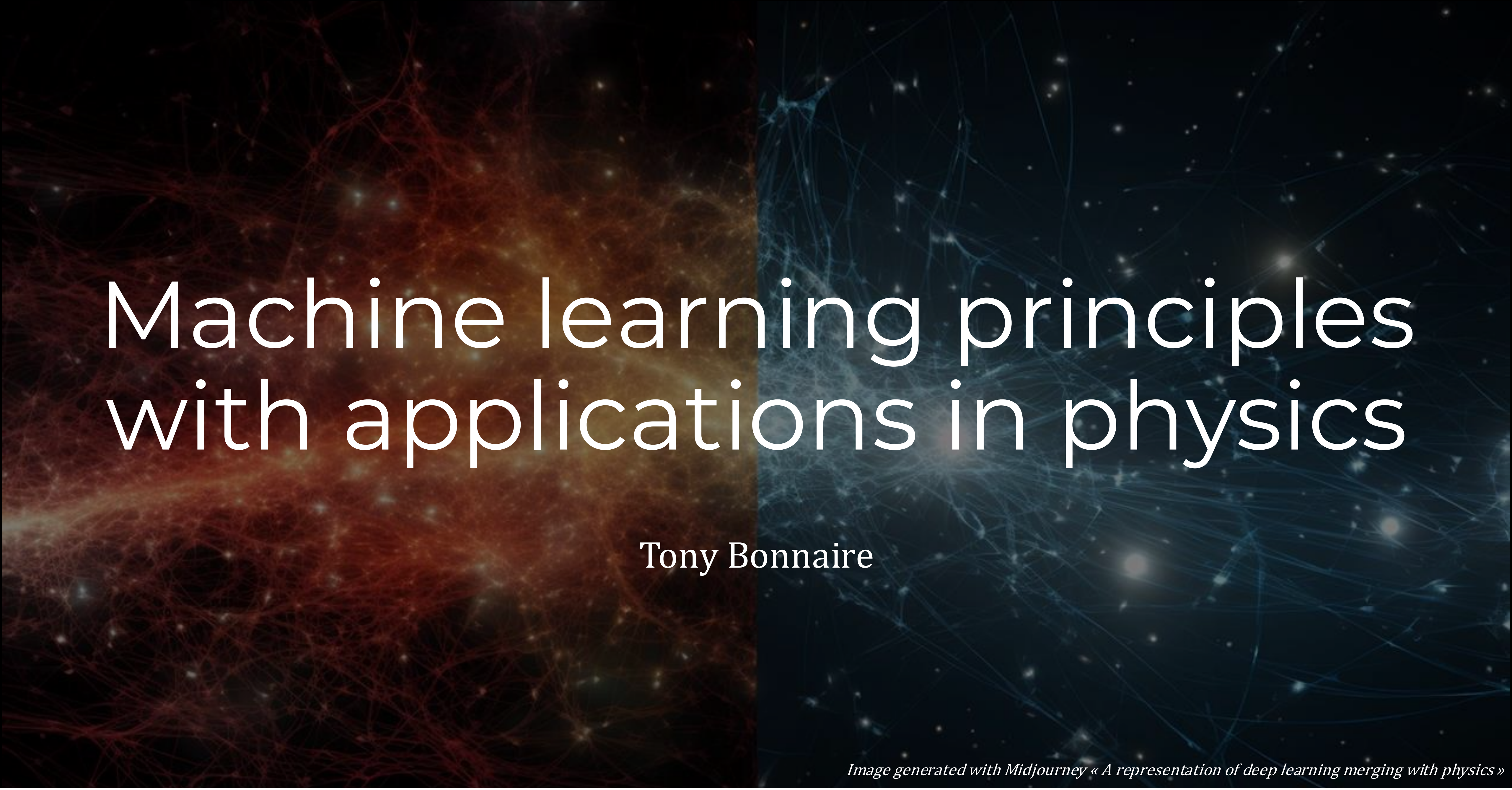


Machine learning principles with applications in physics

Tony Bonnaire

Image generated with Midjourney « A representation of deep learning merging with physics »

September 3rd, École Normale Supérieure, 2024-2025

Organisation of the course

Given by: Myself and **PIERRE Sébastien**.

Format: 3 lectures + projects.

Exam: Oral presentation of your “solution” to the projects.

Aim: Introduce you to the basics of ML allowing to workout applications in physics.

Quick syllabus:

1. **(today 02/09) Crash course on ML: introduction, basic models, optimisation, generalisation**
2. (next week 10/09) End of ML + hands-on + beginning of DL (CNNs, unsupervision with Autoencoders, Diffusion, etc.)
3. (in two weeks 17/09) End of DL + hands-on + presentation of projects

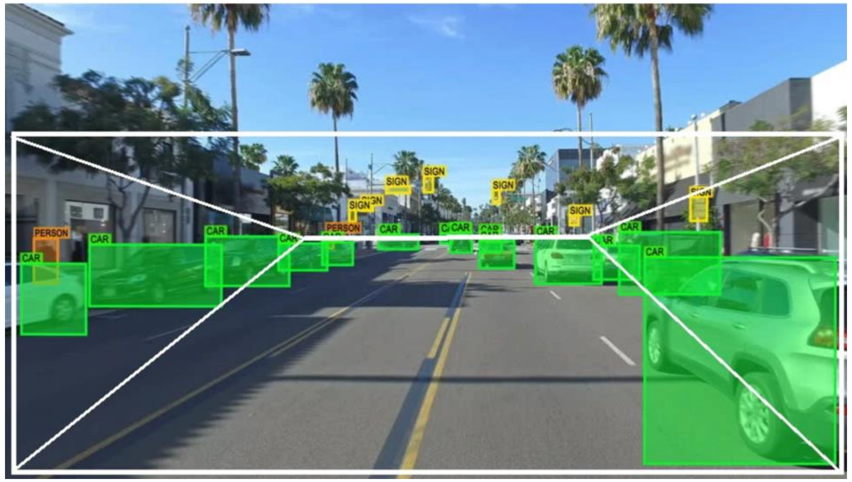
Some references:

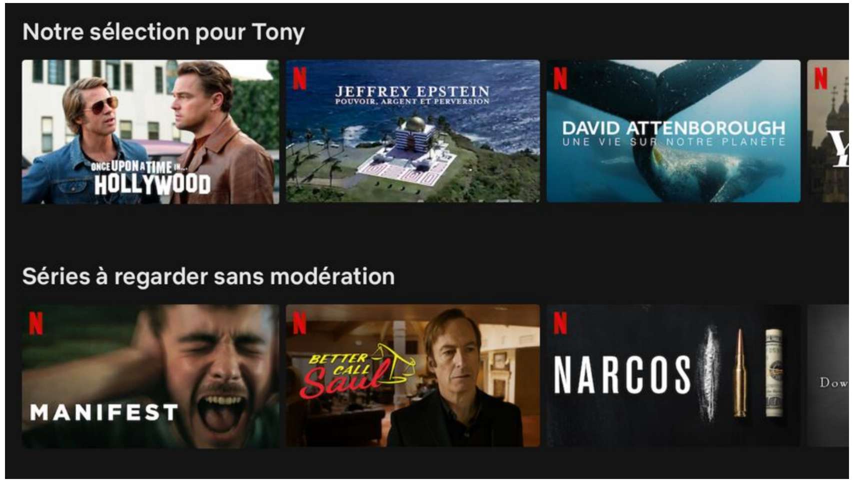
- Deep Learning: Foundations and Concepts, Bishop & Bishop, 2023
- Pattern recognition and Machine Learning, Cristopher Bishop, 2006
- Deep Learning, Goodfellow, Bengio and Courville, 2016
- Learning Theory from First Principles, Francis Bach, 2024
- https://challengedata.ens.fr: a bank of data science challenges to apply all the things we will learn in this course

Intelligent systems

AI goal

Design **systems** capable of performing complex tasks requiring ***intelligence*** (i.e. using reasoning, perception or language) **to take decisions** and **make predictions**.





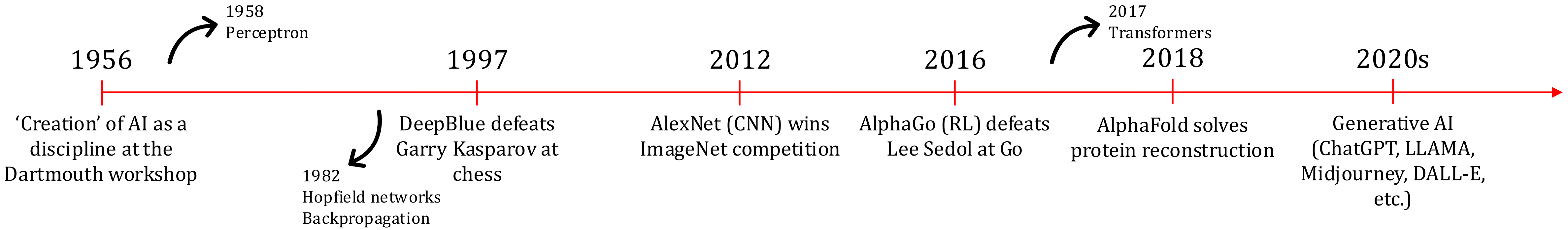
Notre sélection pour Tony

Séries à regarder sans modération

ONCE UPON A TIME IN... HOLLYWOOD
JEFFREY EPSTEIN
POUVOIR, ARGENT ET PERVERSION
DAVID ATTENBOROUGH
UNE VIE SUR NOTRE PLANÈTE
MANIFEST
BETTER CALL Saul
NARCOS

AI revolution

Some (selected) AI breakthroughs


1956
'Creation' of AI as a discipline at the Dartmouth workshop

1958
Perceptron

1982
Hopfield networks
Backpropagation

1997
DeepBlue defeats Garry Kasparov at chess

2012
AlexNet (CNN) wins ImageNet competition

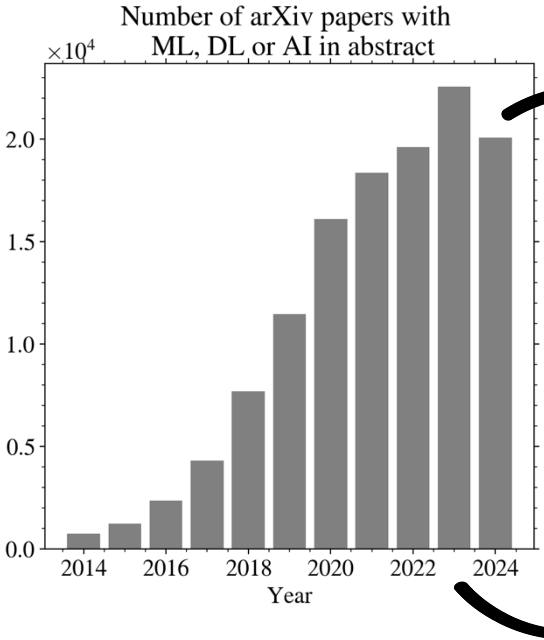
2016
AlphaGo (RL) defeats Lee Sedol at Go

2017
Transformers

2018
AlphaFold solves protein reconstruction

2020s
Generative AI (ChatGPT, LLAMA, Midjourney, DALL-E, etc.)

AI in science


Number of arXiv papers with ML, DL or AI in abstract

2024 not over yet!

60 papers a day in average in 2023 (!)

ML in science

Some scientific applications

Healthcare

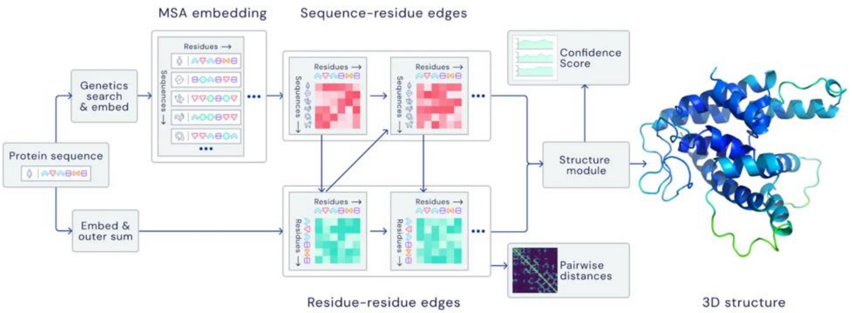
- Drug discovery
- Protein structure reconstruction

Astrophysics and cosmology

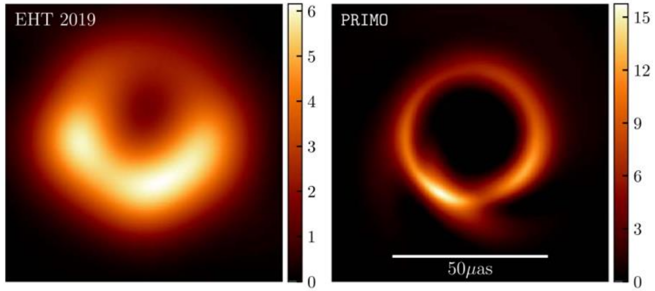
- Galaxy deblending
- Image restoration
- Source separation

Theoretical physics

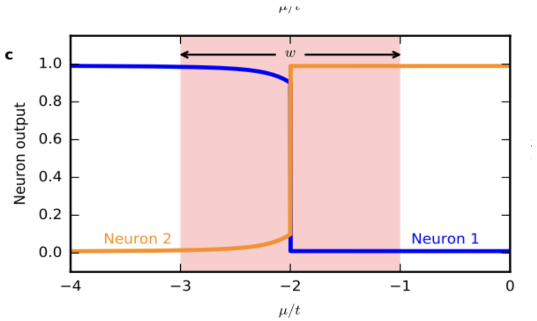
- Study phase transitions
- Discover experiments and equations



Jumper et al., 2021



Medeiros et al., 2023

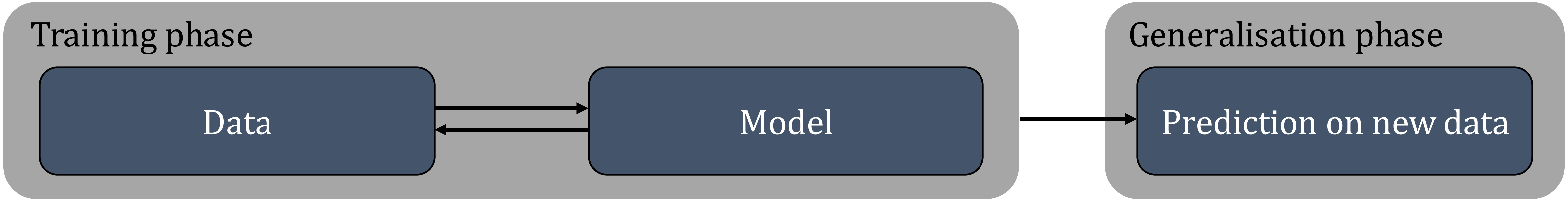


Van Nieuwenburg et al., 2017

... And many more (climate forecast, fraud detection in cybersecurity, binding energies in quantum chemistry)

What is “learning”?

Machine Learning came as a solution to design intelligent systems, replacing handcrafted decision rules by **learnt rules** using **training data** and **optimisation** of **parameterised models**.



Images

Tabular

Sound

Text

....

Linear models

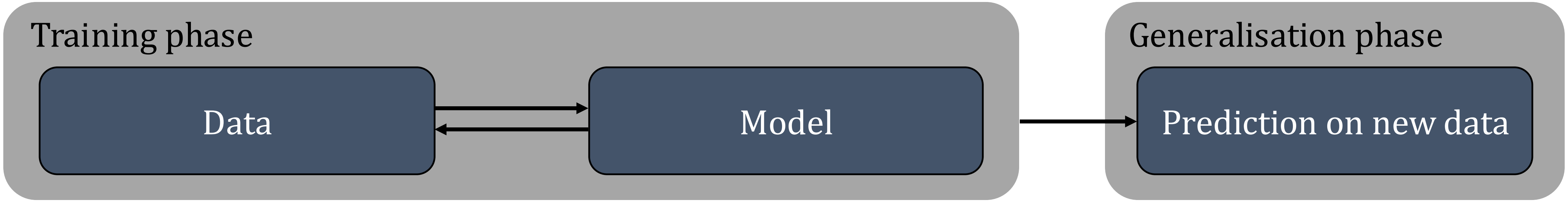
Trees and forests

Neural networks

CNN

...

What is “learning”? An example


Training phase

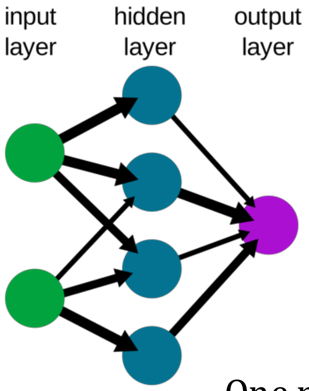
Data

Model

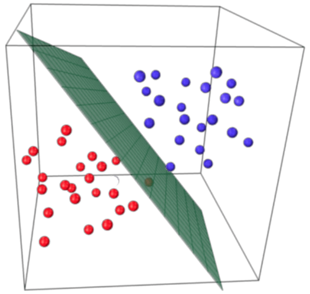
Generalisation phase

Prediction on new data

Images of a ”cat” or “dog”



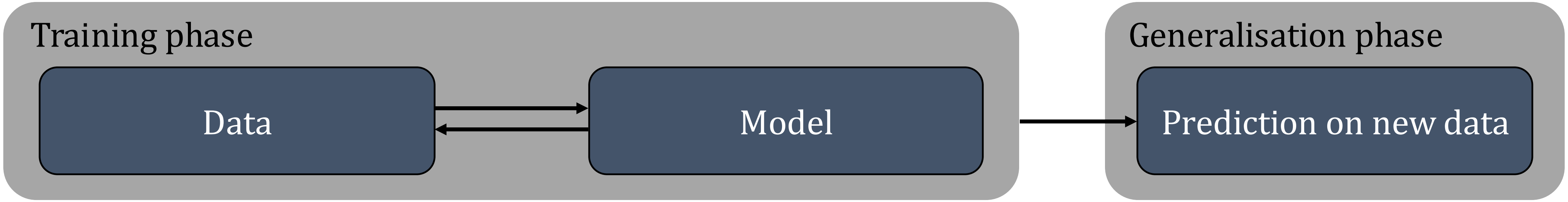
One model



Another model

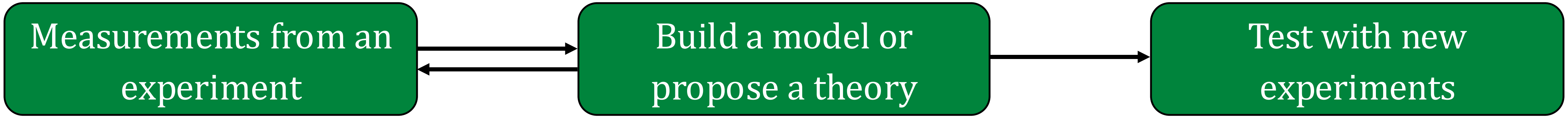
”cat” or “dog” ?

ML and the scientific method



...In fact, all this is close to what you know!

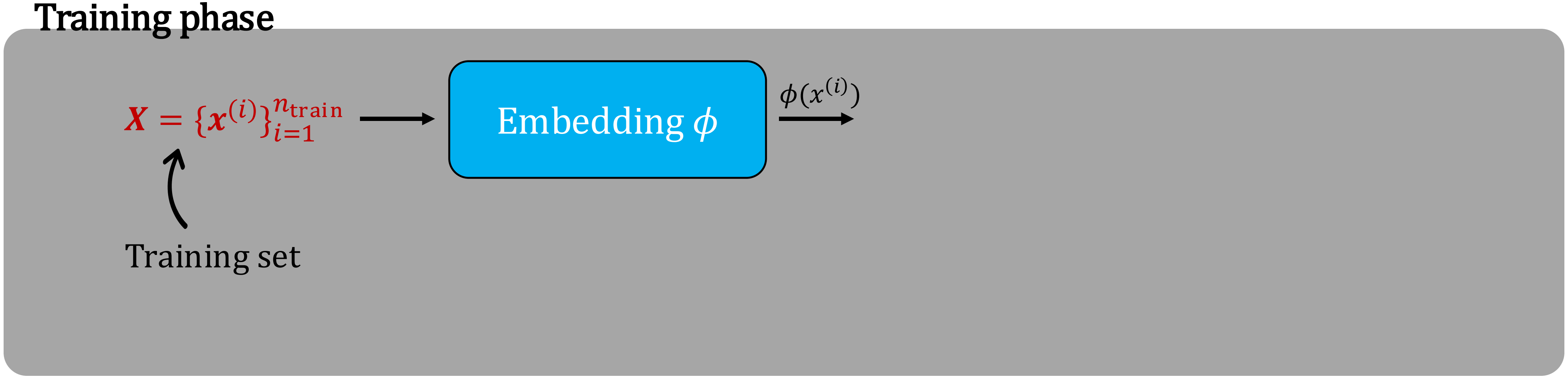
The scientific method


Measurements from an experiment
Build a model or propose a theory
Test with new experiments

Validate/invalidate your model

ML building blocks

Training phase



$\boldsymbol{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{n_{\text{train}}} \longrightarrow$ Embedding ϕ $\xrightarrow{\phi(x^{(i)})}$

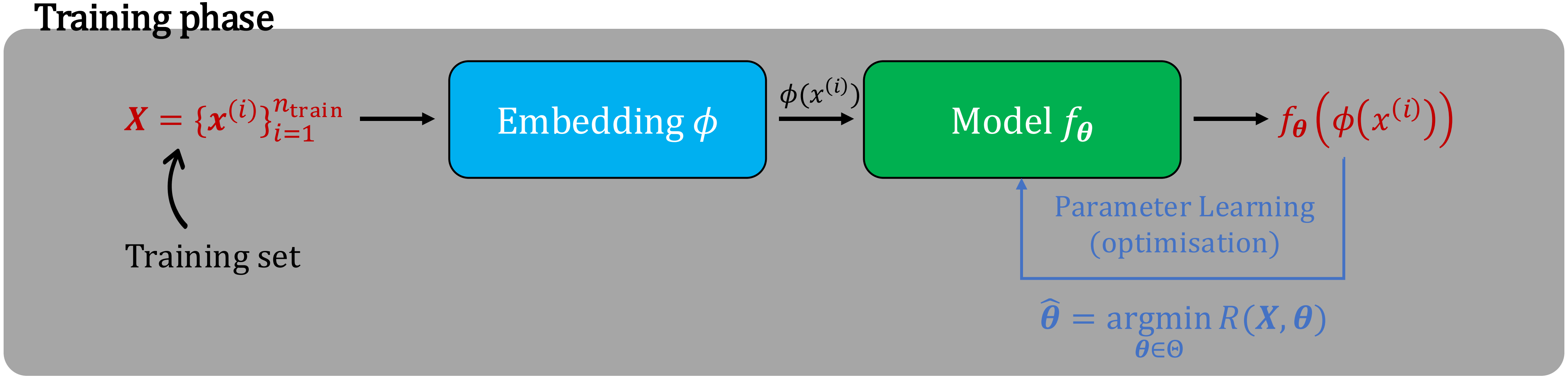
Training set

1. Data X are **unstructured**, sometimes **noisy** and **unprocessed** like pixels of an image or sequence of characters or words.
2. The embedding $\phi(x^{(i)})$ is a **structured**, **numerical** vector representation of the data whose elements are **meaningful features**. It depends on the data and the purpose. It can be **handcrafted or learnt**.

Finding a good embedding is a central part of ML: it eases the problem by preserving the essential structure of the data that matters for the task.

ML building blocks

Training phase



$\boldsymbol{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{n_{\text{train}}}$

Training set

Embedding ϕ

$\phi(x^{(i)})$

Model $f_{\boldsymbol{\theta}}$

$f_{\boldsymbol{\theta}}\left(\phi\left(x^{(i)}\right)\right)$

Parameter Learning (optimisation)

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}}\, R(\boldsymbol{X}, \boldsymbol{\theta})$$

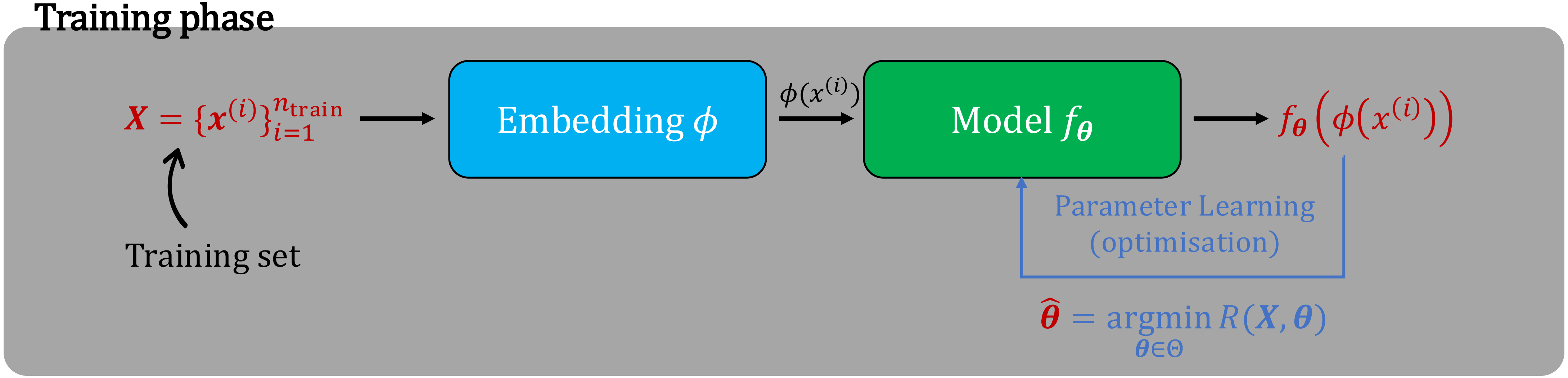
Some notations and terminologies:

- $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ is *one* ***training data*** (there are n_{train} of them),
- $\phi\left(x^{(i)}\right) \in \mathbb{R}^{d'}$ is an embedding of $x^{(i)}$ sometimes called ***feature vector,***
- $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^p$ are the ***parameters*** of the model,
- $R(\boldsymbol{X}, \boldsymbol{\theta})$ is the ***risk*** and measures the error of the model with parameters $\boldsymbol{\theta}$ on data $\boldsymbol{X}$.

At the end of the training procedure, we have a model $f_{\widehat{\theta}}$ committing an error of $R_{\text{train}} = R(\boldsymbol{X}, \widehat{\boldsymbol{\theta}})$ on the training set.

ML building blocks

Training phase



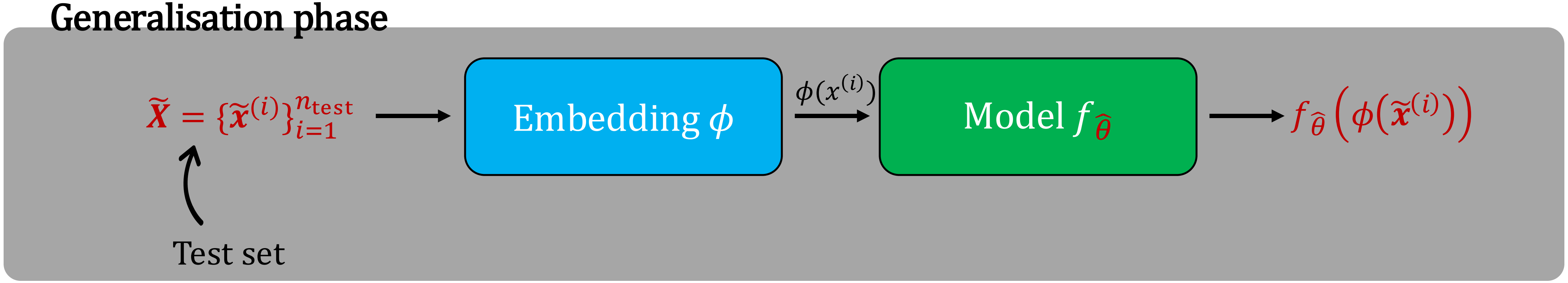
$\boldsymbol{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{n_{\text{train}}}$ → Embedding ϕ $\xrightarrow{\phi(x^{(i)})}$ Model $f_{\boldsymbol{\theta}}$ → $f_{\boldsymbol{\theta}}\left(\phi\left(x^{(i)}\right)\right)$

Training set

Parameter Learning (optimisation)

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}}\, R(\boldsymbol{X}, \boldsymbol{\theta})$$

Generalisation phase



$\widetilde{\boldsymbol{X}} = \{\widetilde{\boldsymbol{x}}^{(i)}\}_{i=1}^{n_{\text{test}}}$ → Embedding ϕ $\xrightarrow{\phi(x^{(i)})}$ Model $f_{\widehat{\theta}}$ → $f_{\widehat{\theta}}\left(\phi\left(\widetilde{\boldsymbol{x}}^{(i)}\right)\right)$

Test set

Using the test set, we can evaluate the test error $R_{\text{test}} = R(\widetilde{\boldsymbol{X}}, \widehat{\boldsymbol{\theta}})$ and compare it to R_{train} to detect **generalisation issues** (**overfitting** or **underfitting**).

Families of ML problems

Supervised learning

$\boldsymbol{x}^{(i)}$ is the i th **data vector** of the training base, and $y^{(i)}$ is the called the **target (or predicted) variable**

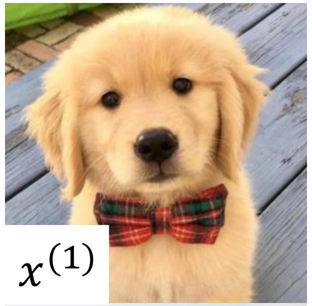
- Training data are actually X and Y coming as pairs

$$\boldsymbol{X} = \left\{\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right)\right\}_{i=1}^{n_{\text{train}}}, \qquad \left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right) \in \mathbb{X} \times \mathbb{Y}$$

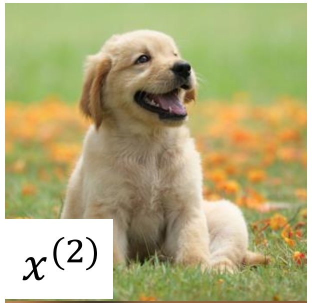
- If $\mathbb{Y}$ is continuous, then the task is called regression, and if $\mathbb{Y}$ is discrete, then it is a classification problem.

<u>Example</u>: Determine if an image encodes a cat or a dog (called a **classification** task)

$X =$

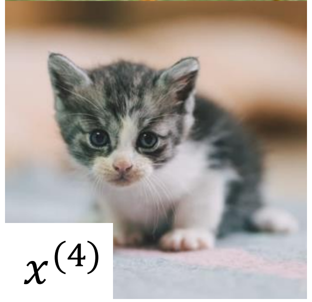


$x^{(1)}$



$x^{(2)}$

$x^{(3)}$



$x^{(4)}$

$Y = \{1, 1, 0, 0\}$

$$f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) = \hat{\boldsymbol{y}}^{(i)} \longrightarrow f_{\hat{\theta}}$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, R(\boldsymbol{X}, \boldsymbol{\theta})$$

Training data

Model

Optimisation

Learning through empirical risk minimisation

Supervised learning

- Training data are actually X and Y coming as pairs

$$\boldsymbol{X} = \left\{\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right)\right\}_{i=1}^{n_{\text{train}}}, \qquad \left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right) \in \mathbb{X} \times \mathbb{Y}$$

- If $\mathbb{Y}$ is continuous, then the task is called regression, and if $\mathbb{Y}$ is discrete, then it is a classification problem.

- Ideally, we would like to minimise the **expected risk**, i.e. the **expected value of a loss function** $\ell(y, \hat{y})$

$$R(\boldsymbol{X}, \boldsymbol{\theta}) = \mathbb{E}_{X,y}[\ell(y, \hat{y})]$$

Loss function: measures how bad your model is on a single example

However, we do not know $p(X, y)$ so in practice we rely on the **empirical risk** instead

$$\hat{R}(\boldsymbol{X}, \boldsymbol{\theta}) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell\left(y^{(i)}, \hat{y}^{(i)}\right).$$

Families of ML problems

Supervised learning

- Training data are actually X and Y coming as pairs

$$\boldsymbol{X} = \left\{\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right)\right\}_{i=1}^{n_{\text{train}}}, \qquad \left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right) \in \mathbb{X} \times \mathbb{Y}$$

- If $\mathbb{Y}$ is continuous, then the task is called regression, and if $\mathbb{Y}$ is discrete, then it is a classification problem.

Examples of tasks

Classification | Regression | Timeseries prediction | Segmentation

Examples of models

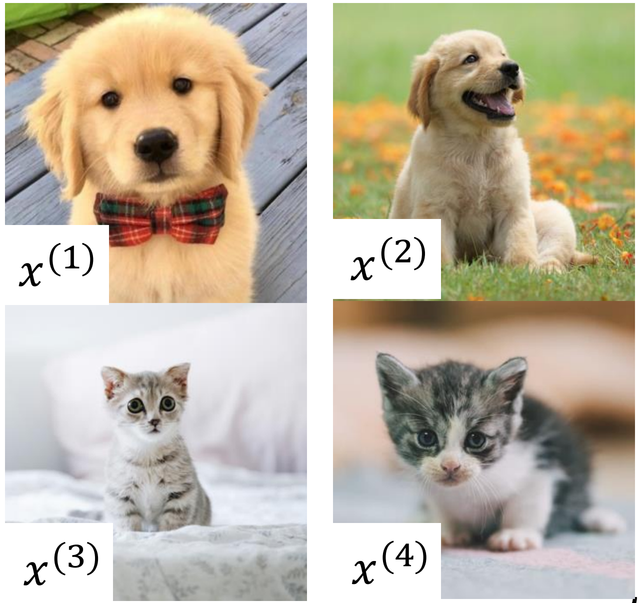
Artificial Neural network | Random forest | Linear regression | Logistic regression | Naïve Bayes | Nearest neighbours

Families of ML problems

Unsupervised learning

- Training data are the set of $\boldsymbol{x}^{(i)}$'s only; no known results to predict
- In unsupervised learning, one seeks **patterns or structures** in X without prior labels
- Usually boils down to model the probability distribution of the dataset

$X =$



Example: Generate new images of cats and dogs (called a **sampling** task)

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = p_{\theta}(\boldsymbol{x}) \longrightarrow f_{\widehat{\theta}}$$

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, R(\boldsymbol{X}, \boldsymbol{\theta})$$

$$\text{such that } p_{\theta}(\boldsymbol{x}) \approx p(\boldsymbol{x})$$

Training data

Model

Optimisation

Families of ML problems

Unsupervised learning

- Training data are the set of $\boldsymbol{x}^{(i)}$'s only; no known results to predict
- In unsupervised learning, one seeks **patterns or structures** in X without prior labels
- Usually boils down to model the probability distribution of the dataset

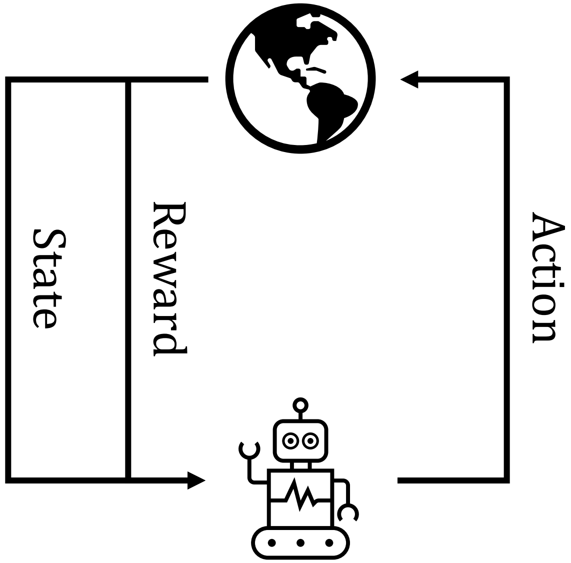
Examples of tasks: Clustering · Data augmentation · Dimensionality reduction · Sampling

Examples of models: Autoencoder · Boltzmann Machine · Diffusion models · Gaussian mixture model · Generative Adversarial network

Families of ML problems

Reinforcement learning

- The philosophy is different: the model does not try to "imitate" like in supervised learning nor to find patterns but "tries" things
- It is based on an **agent** interacting with an **environment**
- The agent tries to find the best possible sequence of states and actions to **maximise a reward**

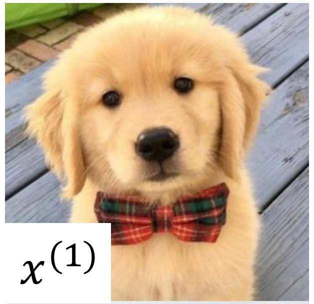


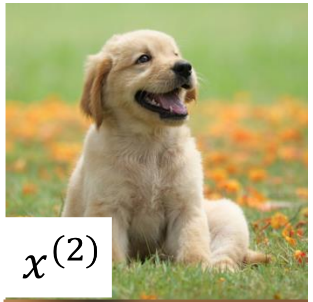
Examples of tasks: Game theory | Robotics | Autonomous driving

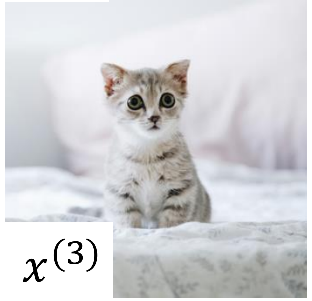
Examples of models: Markov decision processes | Q-networks | Deep policy gradient

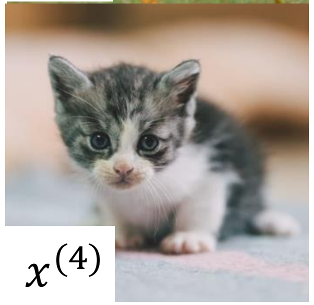
Not discussed in this course, but a very good reference is [Reinforcement Learning – An introduction, Sutton and Barto, 2018]

Why do we need ML?









$$d \approx 10^6$$

1-nearest neighbour

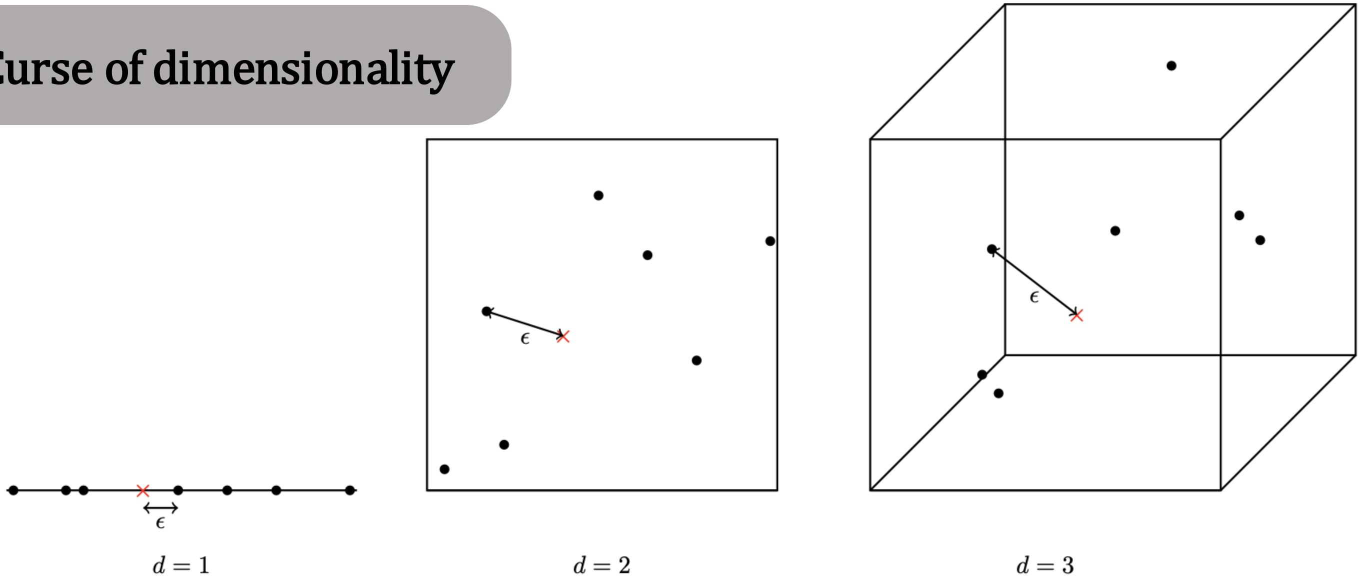
- A simple classification rule is for instance associating to a data the label of its closest neighbour in the d-dimensional space.

$$\hat{y} = y^{(m)} \text{ with } m = \text{argmin}_i \left\| x - x^{(i)} \right\|_2^2$$

- **In this case $R_{\text{train}} = 0$ but R_{test} is very large! Why?**

Curse of dimensionality

- To sample a $[0,1]^d$ space with a shortest distance to a test point at most ϵ, we need $n_{\text{train}} \geq \epsilon^{-d} = e^{-d \log \epsilon}$
- ***$d \approx 80$* requires more samples than the number of atoms in the universe**



Why do we need ML?

Traditional methods typically break down in high-dimensional spaces (**curse of dimensionality**) and it is impossible to design handcrafted decision rules for complex tasks.

The **curse of dimensionality** is the **central problem of machine learning.** To fight it, ML relies on **prior information** about the problem:

- **Reduce the dimensionality**: select a subset of meaningful features (or their interactions) through appropriate embeddings.
- Exploit **structures** in the data (invariances, sparsity, long-range correlations, etc.) to define the model,
- **Penalise** complex models leading to poor generalisation performances using regularisation.

Linear models on feature vectors

Contents:

- *Linear regression model*
- *L2-loss for regression and normal equations*
- *Linear classification model*
- *Softmax function, cross-entropy loss for classification*

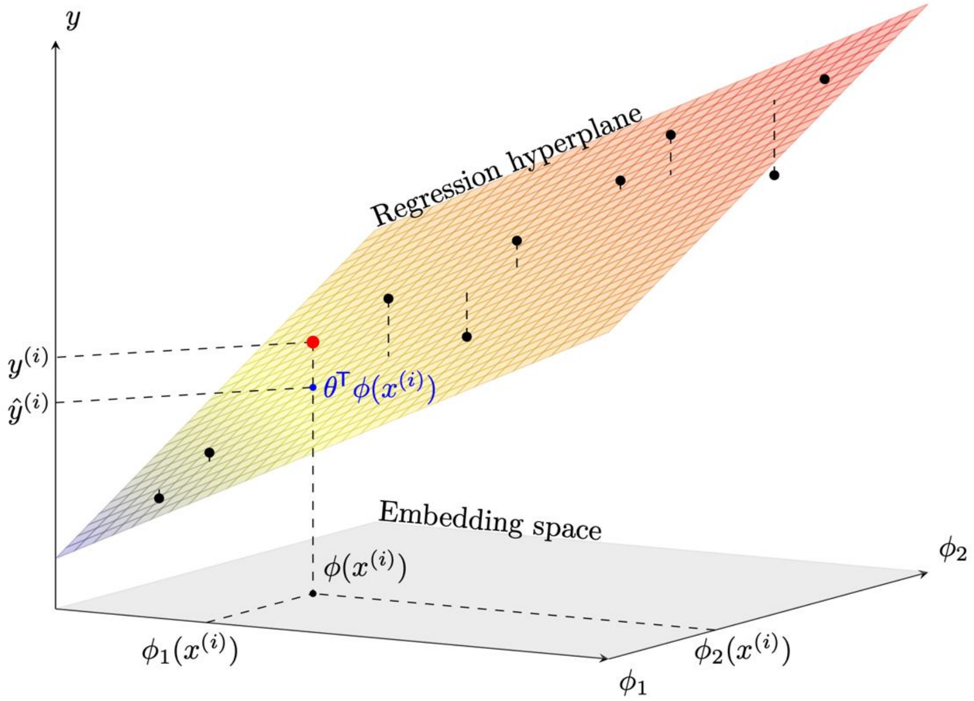
Linear regression: the model

Linear regression

- What kind of problems one can solve efficiently, even in large dimensions? → **Linear systems!**
- Let us talk first about regression: the answer is modelled as

$$f_{\boldsymbol{\theta}}\left(\phi_1^{(i)}, \phi_2^{(i)}, \cdots\right) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\phi}^{(i)} = \hat{y}^{(i)}$$

- It is sometimes convenient to add an affine term (also called **bias** in the neural network literature), which can be absorbed in the feature vector making it of dimension $d' + 1$ where $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, \cdots]^{\mathrm{T}}, \boldsymbol{\phi}^{(i)} = \left[1, \phi_1^{(i)}, \phi_2^{(i)}, \cdots\right]^{\mathrm{T}}$.
- **Geometric interpretation**: projection of an embedding vector onto a **hyperplane** parameterised by $\boldsymbol{\theta}$.



Linear regression: example and ERM

Linear regression

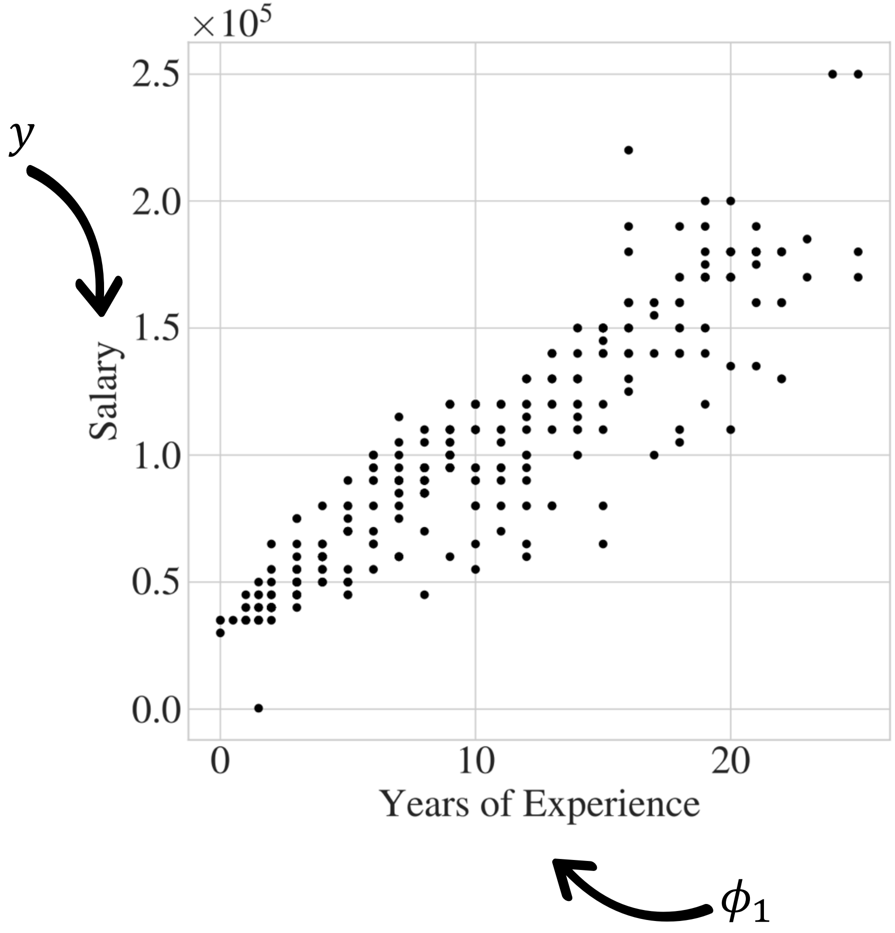
- Example: salary prediction based on the years of experience
- Data are $n = 373$ couples $\left(\phi^{(i)}, y^{(i)}\right) \Longrightarrow$ **Supervised learning**
- The target variable $y \in \mathbb{R}$ is continuous $\Longrightarrow$ **Regression**
- The linear model is

$$\hat{y}^{(i)} = \theta_0 + \theta_1 \phi_1^{(i)},$$

where $\phi_1^{(i)}$ is the nb. of years of experience of the i^{th} training example

- Now the model is fixed, how to find $\widehat{\boldsymbol{\theta}}$, the best possible parameters for our model and data?
- This is done using **empirical risk minimisation (ERM)**

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, R(\boldsymbol{X}, \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell\left(\hat{y}^{(i)}, y^{(i)}\right)$$



Linear regression: solution to ERM

Linear regression

- A common **choice** of loss for regression is a **squared loss function**

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

- ***Here***, the optimisation problem can be solved analytically in closed-form. Rewriting the risk matricially, we have

$$R(\boldsymbol{X}, \boldsymbol{\theta}) = \frac{1}{n_{\text{train}}} \|\boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2$$

Feature matrix

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_1^{(1)} & \cdots & \phi_1^{(d')} \\ \vdots & \ddots & \vdots \\ \phi_{n_{\text{train}}}^{(1)} & \cdots & \phi_{n_{\text{train}}}^{(d')} \end{pmatrix} \in \mathbb{R}^{n_{\text{train}} \times d'}$$

Target vector

$$\boldsymbol{y} = \left[y_1, \dots, y_{n_{\text{train}}}\right]^{\mathrm{T}} \in \mathbb{R}^{n_{\text{train}}}$$

The analytical minimisation of the squared loss in linear regression gives the unique solution (when $d' < n$) known as ***normal equations***

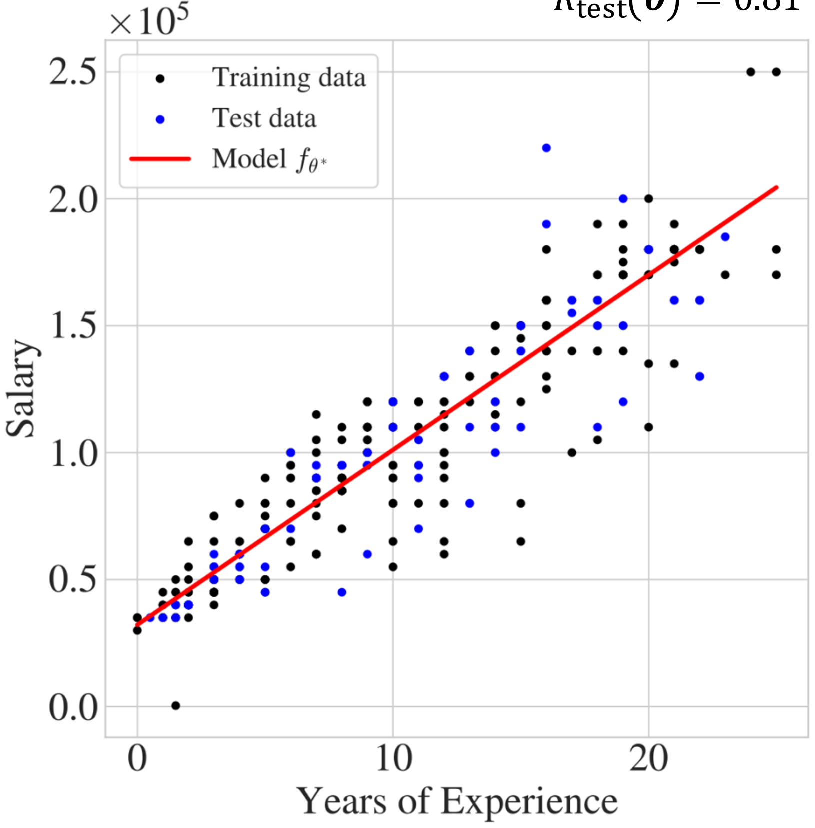
$$\widehat{\boldsymbol{\theta}} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

Linear regression: illustration in 1D

Linear regression

1. I **first** separated the dataset into **training and test sets**, $n_{\text{train}} = 0.8n$ and $n_{\text{test}} = 0.2n$ chosen randomly.
2. Then, I computed the optimal parameters minimising the empirical risk using the normal equations on the training features
$$\widehat{\boldsymbol{\theta}} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}.$$
3. I computed the risk on the train and test sets and found they are close.

$$R_{\text{train}}(\widehat{\boldsymbol{\theta}}) = 0.87$$
$$R_{\text{test}}(\widehat{\boldsymbol{\theta}}) = 0.81$$



+ Exactly solvable model, low variance

− Cannot represent local relationships, may be biased

Linear regression: ERM and MLE

Linear regression

Remark

- The choice of the squared loss can also be motivated from a probabilistic point of view
- Assuming a Gaussian distribution for the error $\epsilon^{(i)} = \hat{y}^{(i)} - y^{(i)} \sim \mathcal{N}(0, \sigma^2)$ and **independent** observations, the ***likelihood*** can be written

$$p(\boldsymbol{X}|\boldsymbol{\theta}) = \prod_i p\left(y^{(i)}\middle|\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right)$$

- Maximising the log-likelihood to obtain the parameters of the model gives

$$\max_\theta \log p(\boldsymbol{X}|\boldsymbol{\theta}) = \max_\theta -\frac{1}{2\sigma_\epsilon^2}\sum_i \left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

→ **The maximum likelihood estimator (MLE) is the same as the empirical risk minimiser under a squared loss function to measure the error of the model**

Linear classification: the model

Linear classification

- Consider the problem of **classifying** data into K distinct classes for which we have a mean to compute features $\phi\left(x^{(i)}\right) \in \mathbb{R}^{d'}$ allowing linear separability of the classes.

- A natural loss function for classification is counting the number of wrong answers, called the **0-1 loss**

$$\ell(y, \hat{y}) = \begin{cases} 1 & \text{if } \hat{y}^{(i)} \neq y^{(i)}, \\ 0 & \text{otherwise.} \end{cases}$$

- The optimal classification decision (in Bayes sense) minimising the risk is therefore

$$\hat{y} = \operatorname{argmax}_k p(y = k | \boldsymbol{\phi}).$$

- We thus need a **model** $p_{\boldsymbol{\theta}}(y = k | \boldsymbol{\phi})$ of the conditional probability distribution to perform classification!

Linear classification

- The simplest models assume a linear log probability

$$\log p_\theta\left(y^{(i)} = k \middle| \boldsymbol{\phi}^{(i)}\right) = \boldsymbol{\theta}_k^{\mathrm{T}} \boldsymbol{\phi}^{(i)} - \log Z$$

where Z is a normalizing constant so that probabilities sum to one.

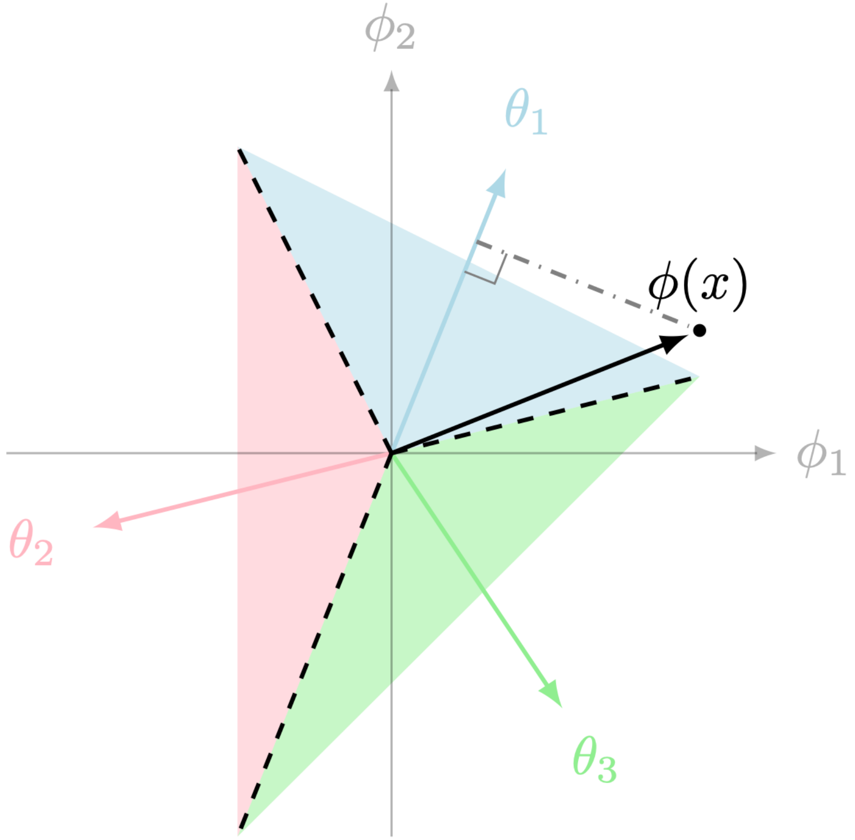
- It means that

$$p_\theta\left(y^{(i)} = k \middle| \boldsymbol{\phi}^{(i)}\right) = \frac{\exp\left(\boldsymbol{\theta}_k^{\mathrm{T}} \boldsymbol{\phi}^{(i)}\right)}{\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^{\mathrm{T}} \boldsymbol{\phi}^{(i)}\right)}$$

which is called the **softmax function** allowing to turn the linear responses for each class into probabilities.

- And the classification rule is

$$\hat{y} = \operatorname{argmax}_k \boldsymbol{\theta}_k^{\mathrm{T}} \boldsymbol{\phi}^{(i)}$$



Geometrically, it corresponds to computing the overlap between the feature $\boldsymbol{\phi}^{(i)}$ and a vector representative for each class, $\boldsymbol{\theta}_k$, and associating **the class maximising the dot product**, leading to **linear decision** boundaries shown as hyperplanes.

Linear classification: fitting parameters

Linear classification

- Now the model is specified, we need minimise the risk to obtain the parameters $\boldsymbol{\theta}_k$ using some training data

- For optimisation, we cannot use the 0-1 loss since it is not differentiable, but we can relax it using the previous probabilities, and write the empirical risk as

$$R(\boldsymbol{X}, \boldsymbol{\theta}) = -\sum_{i=1}^{n_{\text{train}}} \sum_{k=1}^{K} 1_{y^{(i)}=k} \log p_\theta(y^{(i)} = k | \boldsymbol{\phi}^{(i)})$$

which is **now differentiable and convex** (the second derivative is positive definite), suitable for optimisation.

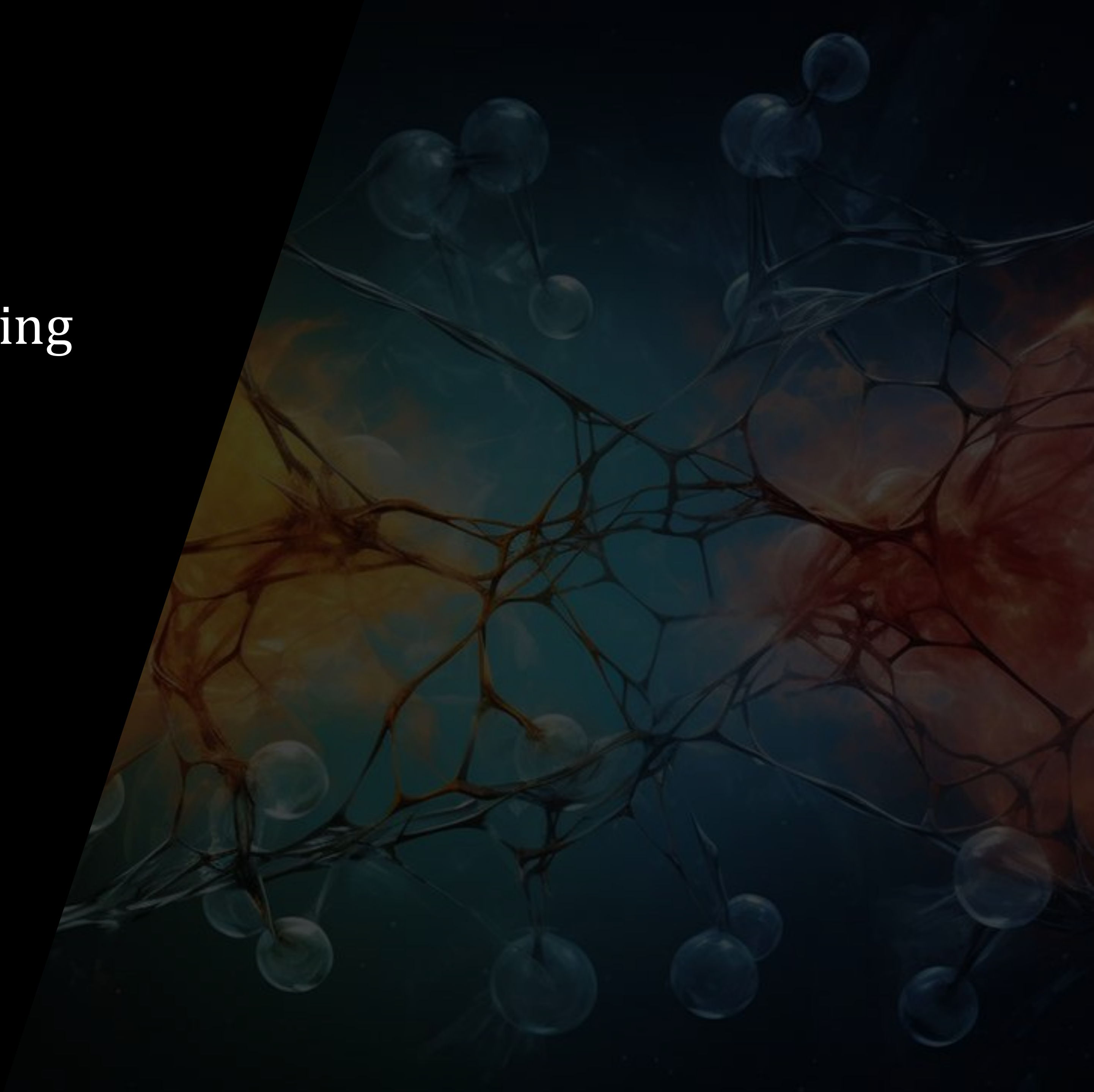
This risk is referred to as ***cross-entropy*** and it is the most widely used cost function for classification problems. The parameters of the model are then obtained by minimising the risk, i.e.

$$\widehat{\boldsymbol{\theta}} = \text{argmin}_\theta \, R(\boldsymbol{X}, \boldsymbol{\theta}).$$

Principles of Supervised Learning

Contents:

- *Bias-variance trade-off for supervised problems*
- *Overfitting, underfitting and test set*
- *Explicit regularisation*



Linear regression: ERM and MLE

In the previous chapter, we have:

1. Specified different models for different supervised learning tasks (regression and classification),
2. Specified loss functions and associated empirical risks,
3. Used a finite training set to minimise the empirical risk.
4. Found parameters of our models $f_{\boldsymbol{\theta}}(\phi^{(i)})$

Now what could possibly go wrong with our models?

Generalisation!

Is it able to work on new, independent from training, data?

The bias-variance trade-off

- The whole aim of training supervised models is to generalise well on unseen data. In practice, we would like to minimise $\mathbb{E}_{\boldsymbol{x},y}\big(\ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y)\big)$ that we approximate by $\frac{1}{n_{\text{train}}}\sum_{i=1}^{n_{\text{train}}} \ell\left(\left(f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)\right)$
- In a regression context, suppose there exists f such that $y^{(i)} = f\left(\boldsymbol{x}^{(i)}\right) + \epsilon^{(i)}$, with $\mathbb{E}\left[\epsilon^{(i)}\right] = 0, \mathbb{E}\left[\epsilon^{(i)^2}\right] = \sigma_\epsilon^2$
- We build a model f_θ of f minimising the squared error $\ell\left(\left(f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)\right) = \left(y^{(i)} - f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right)^2$
- We can show that the expected risk on a test example $\tilde{\boldsymbol{x}}$ decomposes as

$$\mathbb{E}\left[\left(y - f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})\right)^2\right] = \mathbb{E}[f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}) - f(\tilde{\boldsymbol{x}})]^2 + \mathbb{E}[(f(\tilde{\boldsymbol{x}}) - \mathbb{E}[f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})])^2] + \sigma_\epsilon^2$$

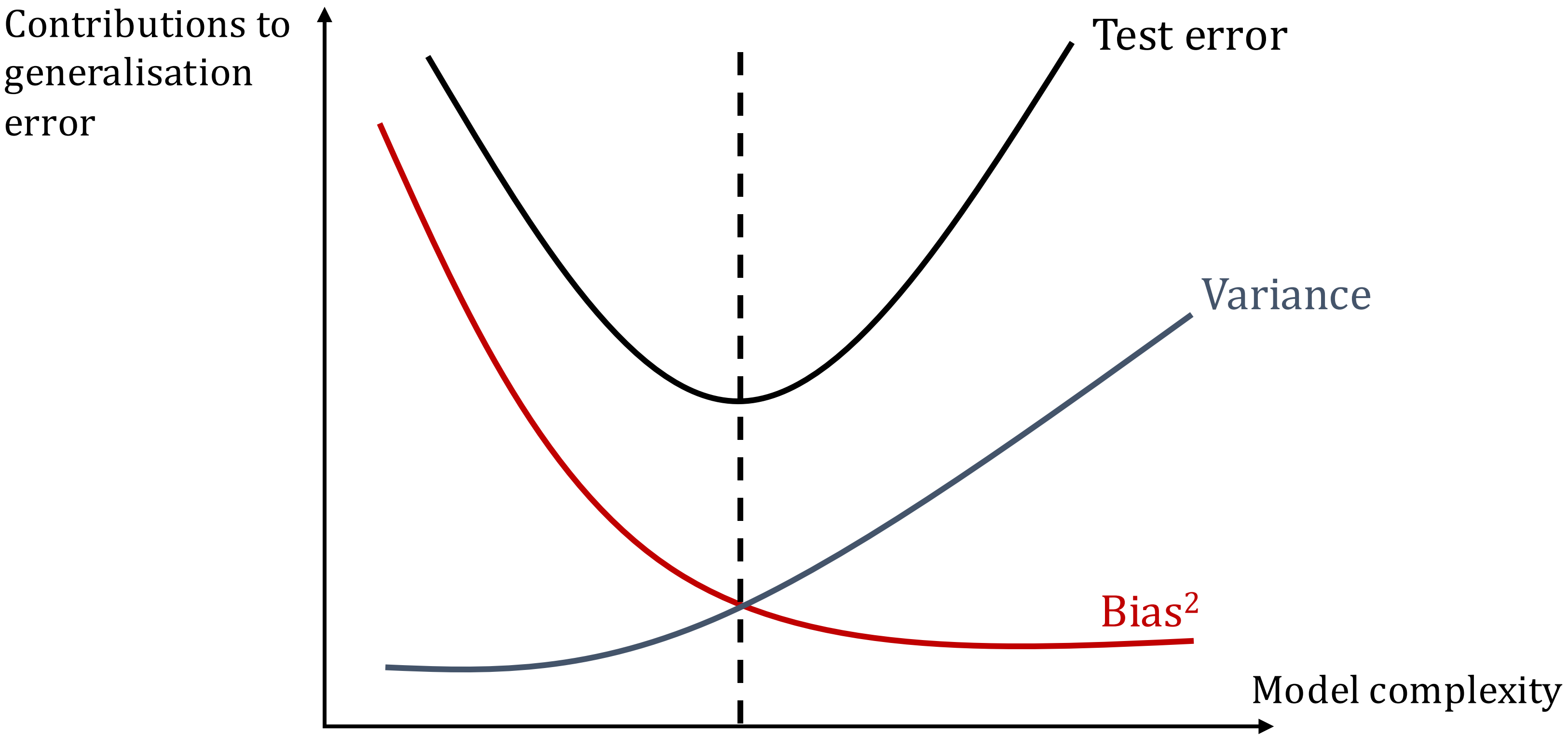
$$\mathbb{E}\left[\left(y - f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})\right)^2\right] = \underbrace{\text{Bias}[f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})]^2}_{\text{Modelling error}} + \underbrace{\text{Var}[f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})]}_{\text{Model variability}} + \underbrace{\sigma_\epsilon^2}_{\text{Irreducible error}}$$

Note: Every terms are >0

Note: a similar expression holds for classification

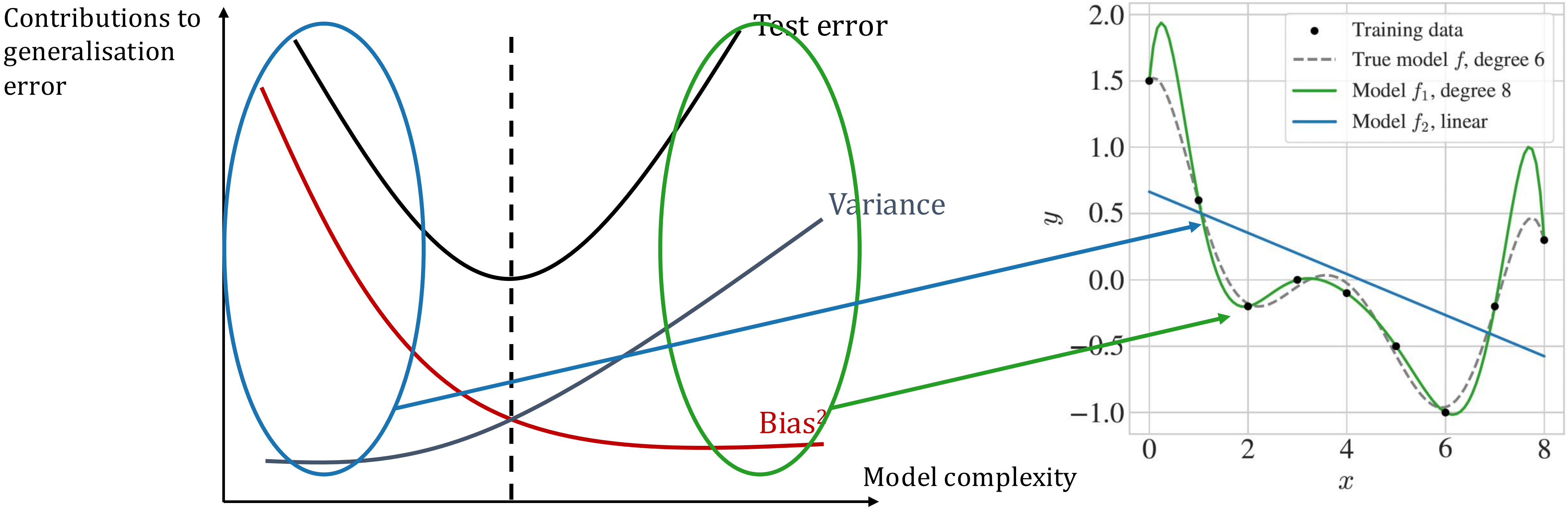
Model complexity and generalisation error

- **"Simple" models** have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- **"Complex" models** (with a lot of parameters for instance) have **small bias** but **large variance**



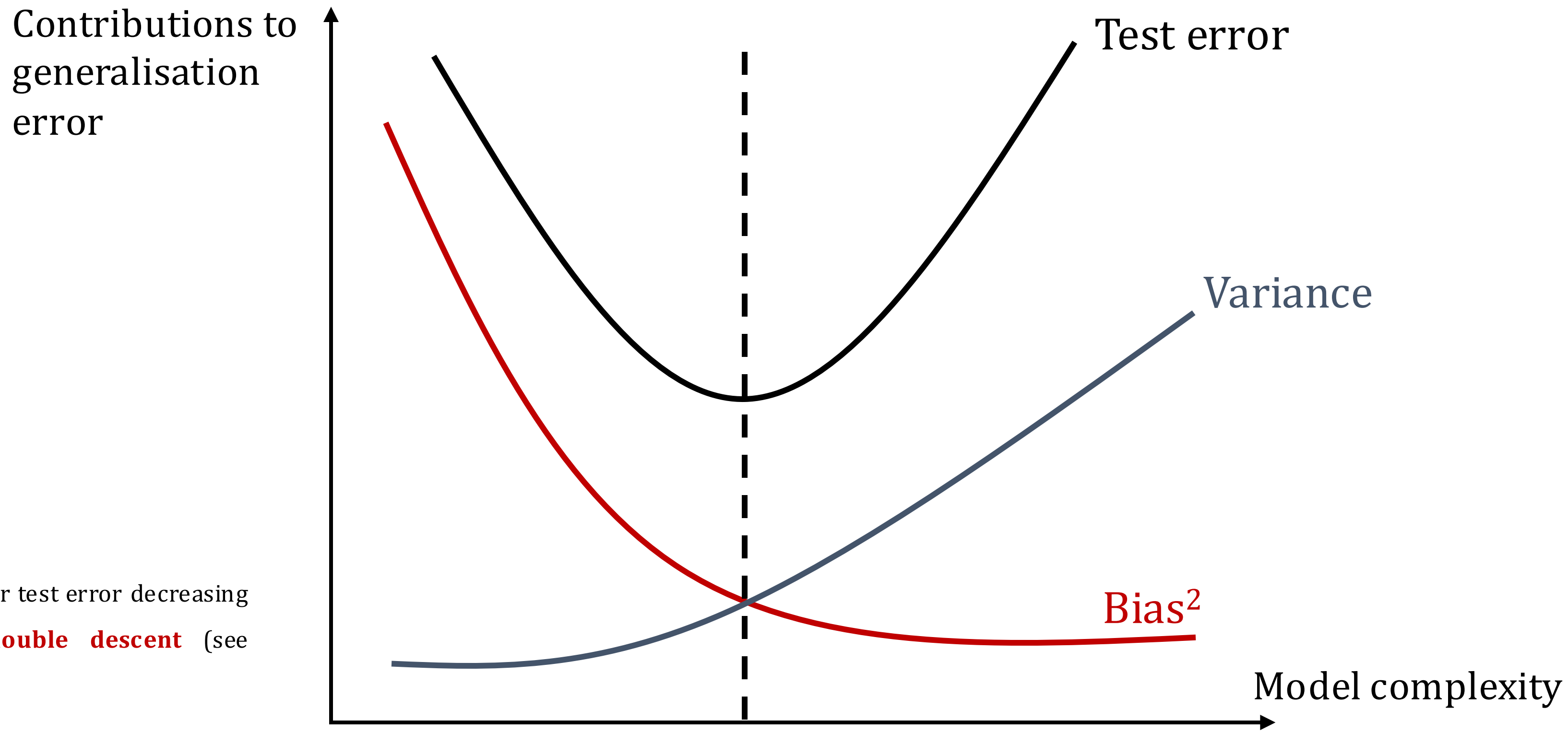
Model complexity and generalisation error

- **“Simple” models** have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- “**Complex” models** (with a lot of parameters for instance) have **small bias** but **large variance**



Model complexity and generalisation error

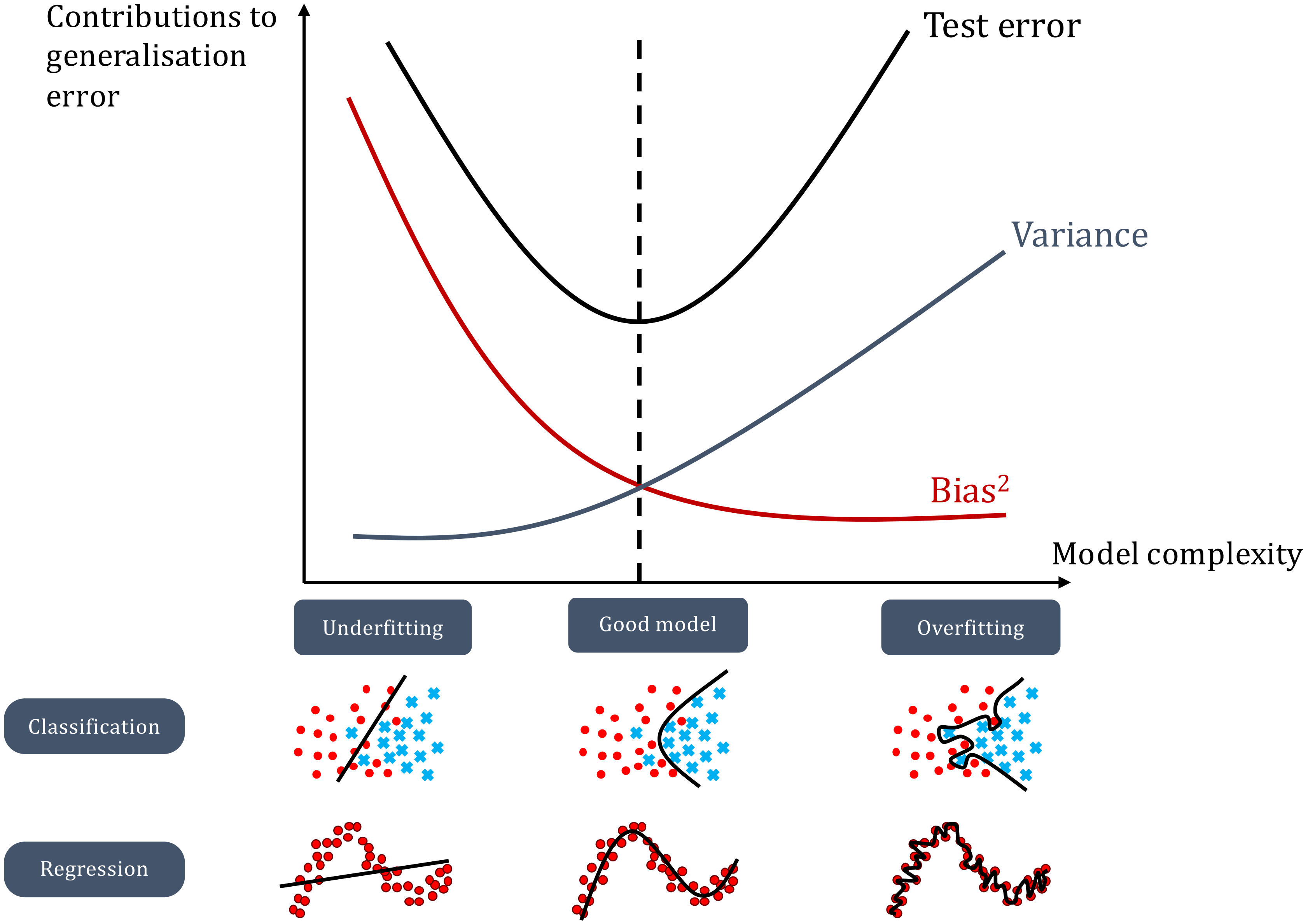
- **“Simple” models** have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- **“Complex” models** (with a lot of parameters for instance) have **small bias** but **large variance***



*Heavily overparametrized have their test error decreasing again, a phenomenon dubbed **double descent** (see Belkin+18 and Nakkiran+19)

Models need to be built such that they are not too flexible to fit the noise in the data but also not too restrictive to avoid bias

Model complexity and generalisation error



Underfitting

Good model

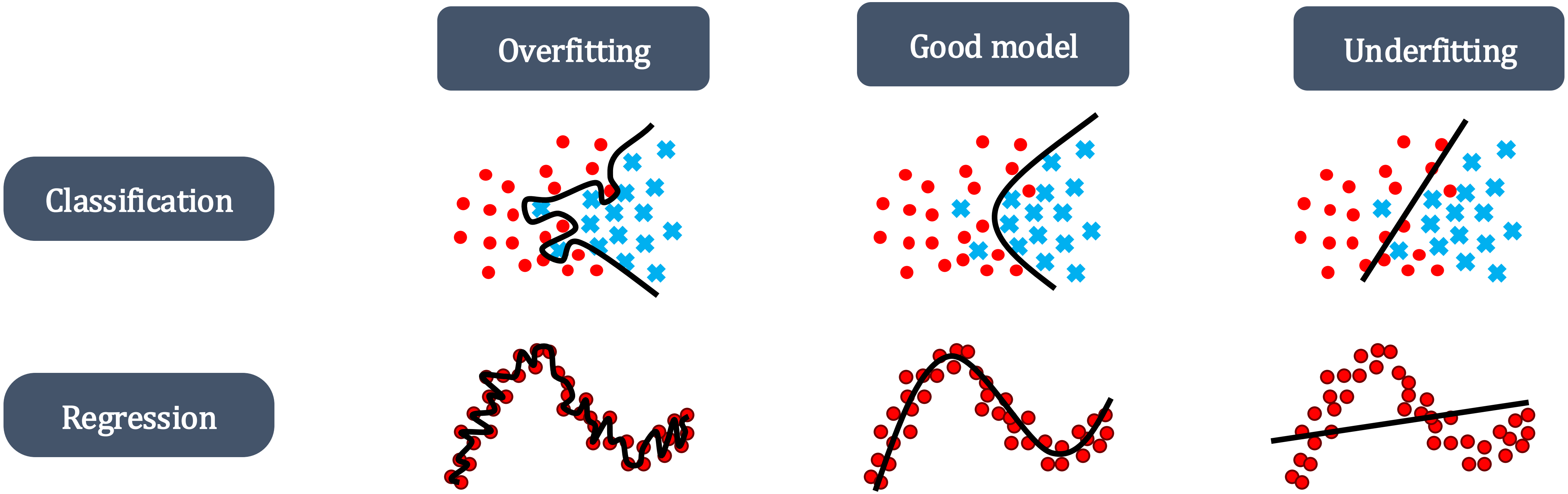
Overfitting

Classification

Regression

Generalisation in practice

- To measure if we really learnt something useful in supervised learning in practice, we use a **test dataset** that the model has never seen but for which we know the labels and check that $R_{\text{train}}(\hat{\theta})$ and $R_{\text{test}}(\hat{\theta})$ are of the same order
- Based on the previous view, there are two regimes where things could go wrong: high variance and low bias models vs high bias and low variance models, respectively defining **overfitting** and **underfitting**



Overfitting | Good model | Underfitting

Classification

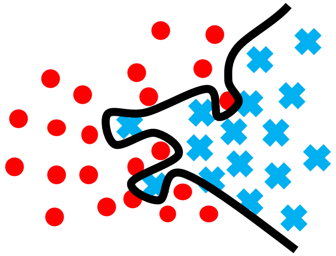
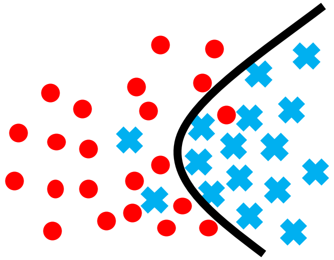
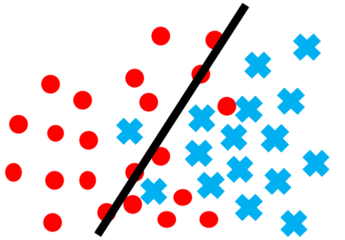
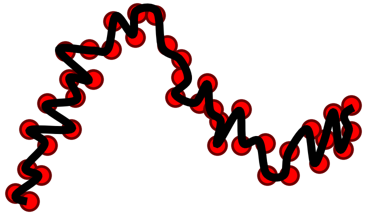
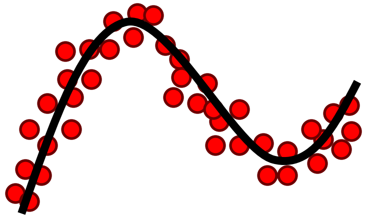
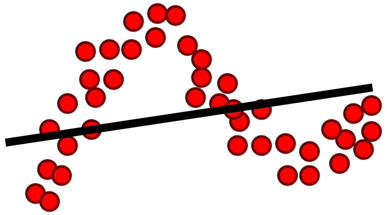
Regression

$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \ll R_{\text{test}}(\hat{\boldsymbol{\theta}})$

$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \approx R_{\text{test}}(\hat{\boldsymbol{\theta}})$

$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \approx R_{\text{test}}(\hat{\boldsymbol{\theta}})$
but large

Some fixes to under and over fitting

	Overfitting	Good model	Underfitting
Classification			
Regression			
	$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \ll R_{\text{test}}(\hat{\boldsymbol{\theta}})$	$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \approx R_{\text{test}}(\hat{\boldsymbol{\theta}})$	$R_{\text{train}}(\hat{\boldsymbol{\theta}}) \approx R_{\text{test}}(\hat{\boldsymbol{\theta}})$ but large
Possible fixes	• Add more data • Remove features • Stop the training earlier • Add regularisation	• You did a good job!	• Try a more complex model • Train your model longer

A gentle introduction to regularisation

- One generic way to deal with **overfitting** is by penalising ‘complex’ models and restrain the class of learned functions: this is called explicit **regularization** and appears in the optimization problem as a constraint on parameter space

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}}\, R(\boldsymbol{\theta})$$

Unregularized optimization

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}}\, R(\boldsymbol{\theta}) \;\; \text{s.t.}\, P(\boldsymbol{\theta}) \leq \epsilon$$

Regularized optimization

- The function $P(\boldsymbol{\theta})$ is called **penalty function** and the regularized problem can in fact be written equivalently as (by Lagrange duality)

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}}\, R(\boldsymbol{\theta}) + \lambda P(\boldsymbol{\theta})$$

1. λ is a **regularization** (hyper)**parameter**
2. $P(\boldsymbol{\theta})$ can take different forms depending on the penalty we chose to impose on the set of parameters

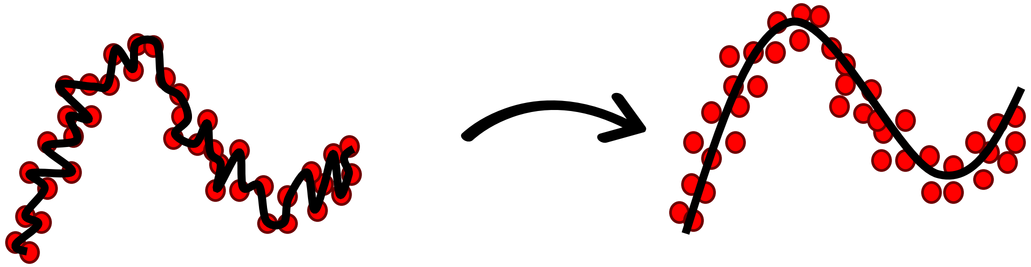
- Common penalty functions are the L_p-norms with $p = 1$ or 2

$$P(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_p$$

Regularisation as a smoothness constraint

- By solving the regularized optimization with **an appropriate value of λ**, we can reduce the overfitting

Unregularized optimization



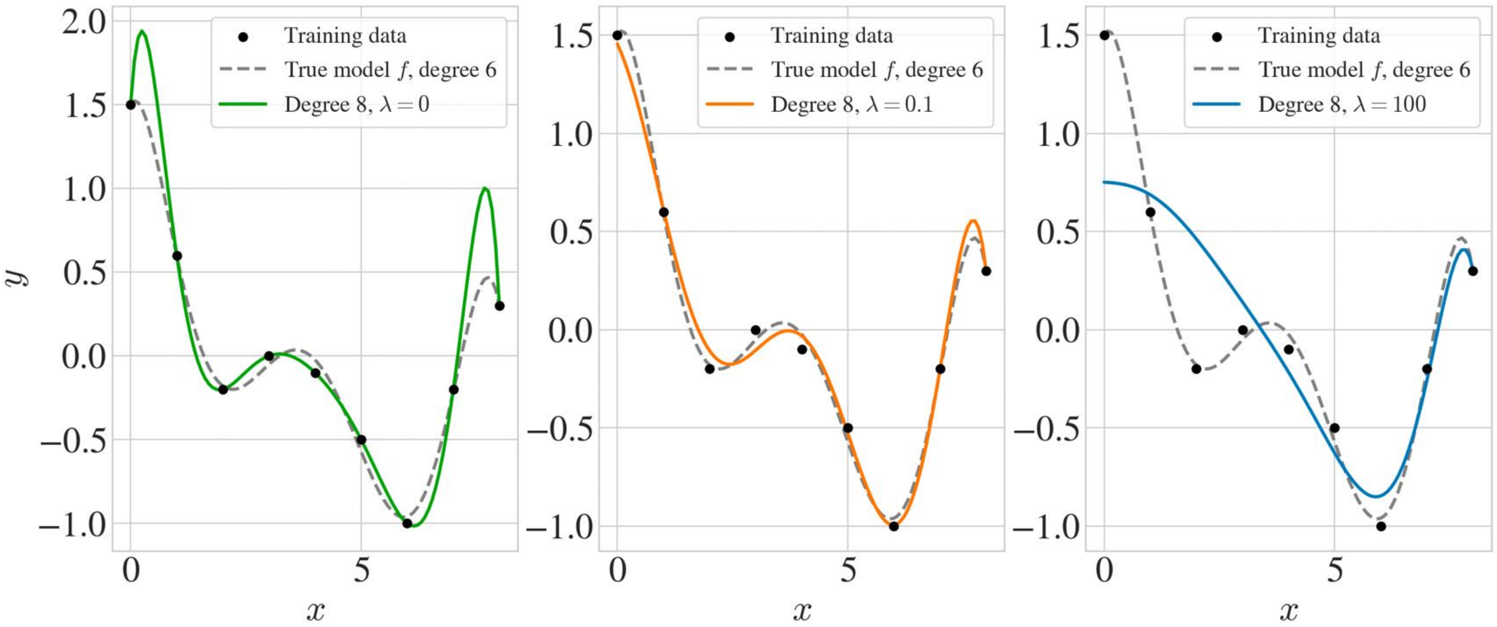
Regularized optimization

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}}\, R(\boldsymbol{\theta})$$

$$\hat{\theta} = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}}\, R(\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|_2^2$$

- **The unregularized models lacks smoothness** which is introduced by the L_2 penalty

- The additional term here penalizes large weight values and **reduces the variance** of the estimator



A word about hyperparameters

- One **hyperparameter**: the regularisation parameter λ

Hyperparameter: parameter that is **not learned** during the optimisation.
Examples include regularisation parameter, depth of trees, learning rate in optimisation.

- For the regularisation parameter: a value that is too large introduces a large bias, while if too small and close to zero, we do not solve the overfitting issue
- Usually, we use **grid search** to find the hyperparameter performing best on a third dataset: the **validation set**

- **Training set**: training data used to to learn parameters of the model during optimisation,
- **Test set**: independent set used to evaluate and compare the models (should in principle be used once by a model)
- **Validation set**: used to fit hyperparameters of the model by varying it and keeping the value minimising the validation error R_{valid}.

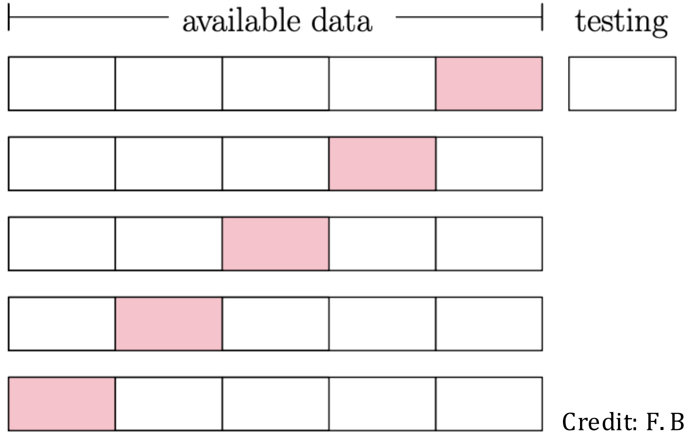
A word about hyperparameters

- One **hyperparameter**: the regularisation parameter λ

Hyperparameter: parameter that is **not learned** during the optimisation.
Examples include regularisation parameter, depth of trees, learning rate in optimisation.

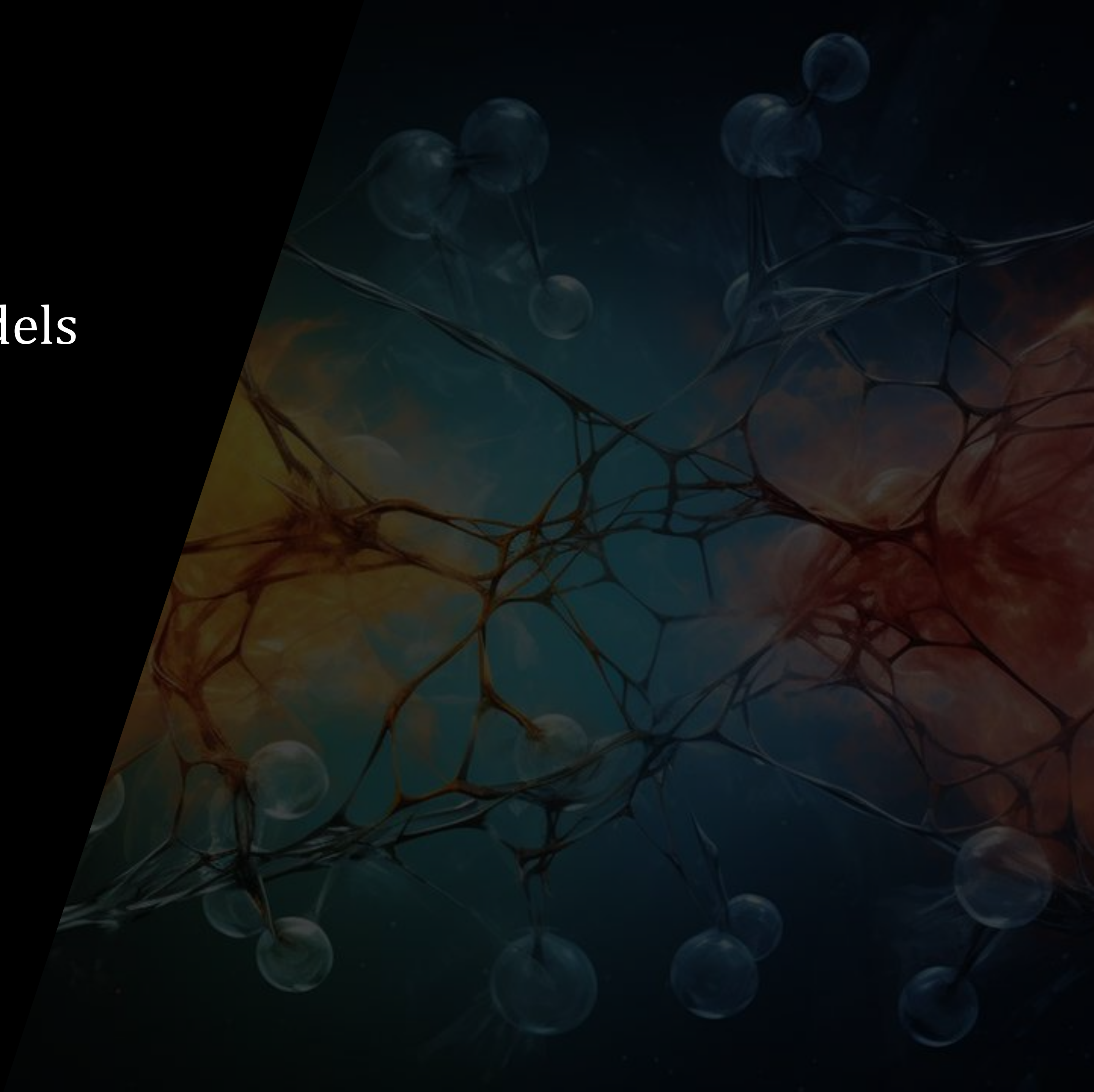
- For the regularisation parameter: a value that is too large introduces a large bias, while if too small and close to zero, we do not solve the overfitting issue
- Usually, we use **grid search** to find the hyperparameter performing best on a third dataset: the validation set or **cross-validation**

Cross-validation: Split the available data into k folds and train the model k times changing the chunk of validation set. At the end, average the obtained errors.



Credit: F. Bach

Other supervised learning models



Contents:

- *A first non-linear model: decision trees*
- *Ensembling: bagging and boosting*
- *Random forest and boosted trees*
- *Feed-forward neural networks*

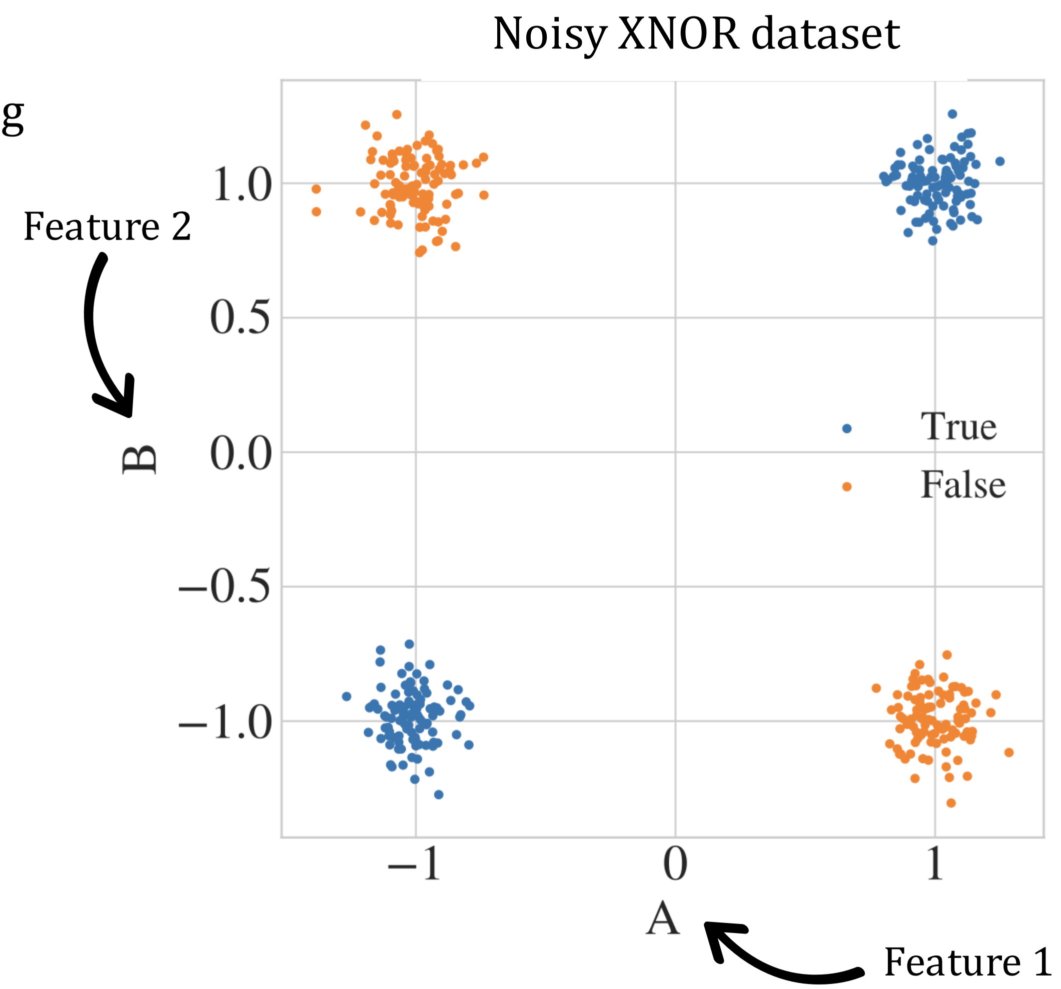
A classification task: the XOR dataset

Decision trees

- Consider a **classification task** on an artificial dataset replicating the XNOR function

A	B	XNOR
True	True	True
True	False	False
False	True	False
False	False	True

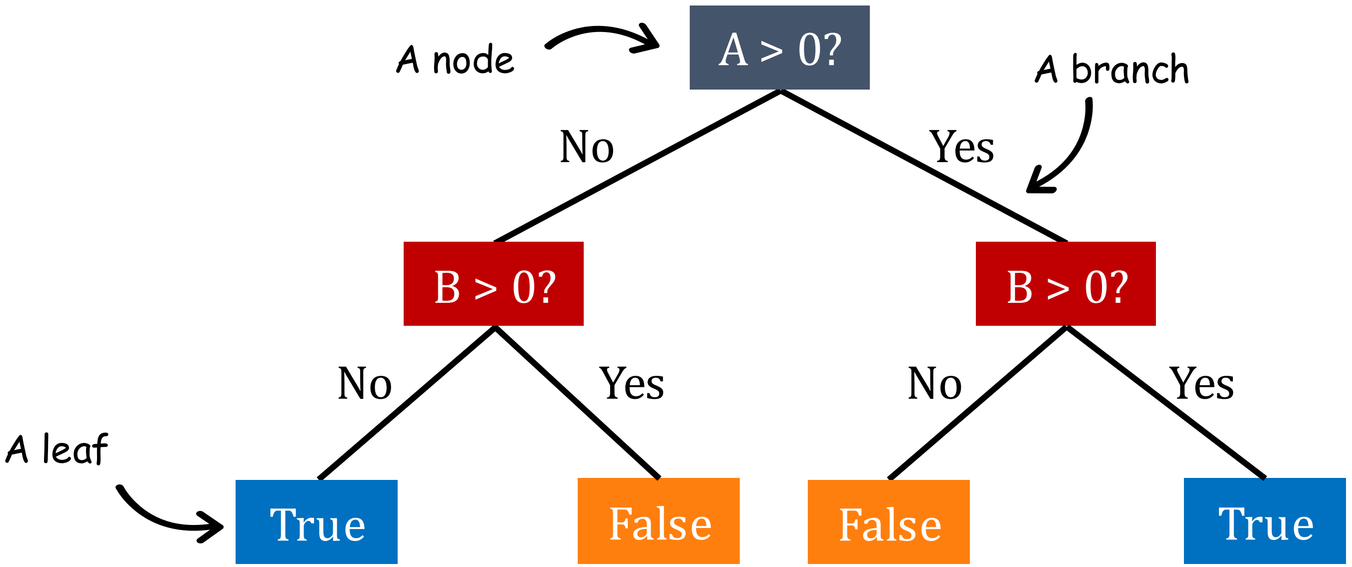
- Data are $n = 500$ couples $\left(\boldsymbol{\phi}^{(i)}, y^{(i)}\right) \Longrightarrow$ **Supervised learning**
- The target variable $y \in \{-1,1\}$ is discrete $\Longrightarrow$ **Classification**
- A linear classification would not be able to learn such a function*
- **Decision trees** to the rescue!



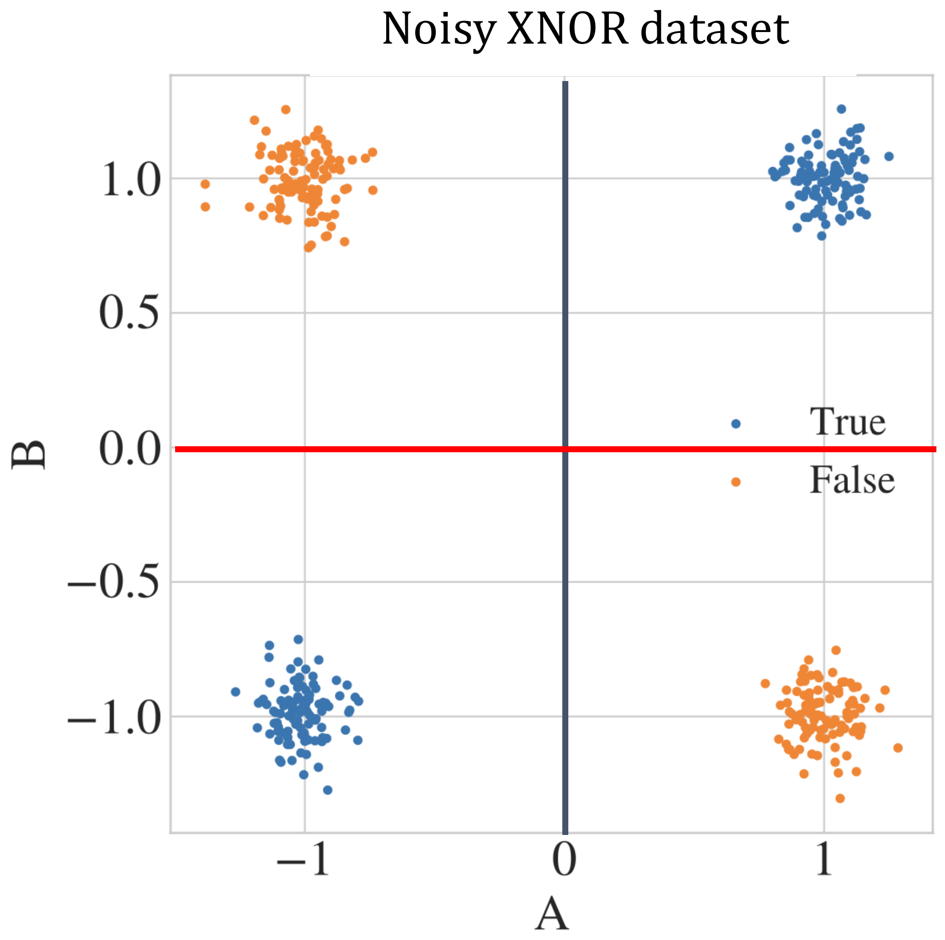
*In fact, an alternative (not discussed here) would be to use **kernels** to find a higher dimensional space in which the data separates linearly and then use a linear classifier.

Decision trees

- Decision trees incrementally ask questions about the features to split the problem into smaller, simpler (binary) decisions



- All root and inner nodes question the value of a feature, and branches split the dataset into **different regions to which a datapoint can belong uniquely**



Decision trees

- More formally, at a given node in parent region R_t asking a question about the j^{th} feature, we create two regions:

$$R_1 = \{\boldsymbol{x} \mid x_j < \alpha_t^j, \boldsymbol{x} \in R_t\}$$
$$R_2 = \{\boldsymbol{x} \mid x_j \geq \alpha_t^j, \boldsymbol{x} \in R_t\}$$

- The parameters $\boldsymbol{\theta}$ of decision trees are the threshold values at each nodes (the sequence of α)
- Decision tree minimise a loss function at each node of the tree: the **cross-entropy** or the **squared error** (classification vs regression)

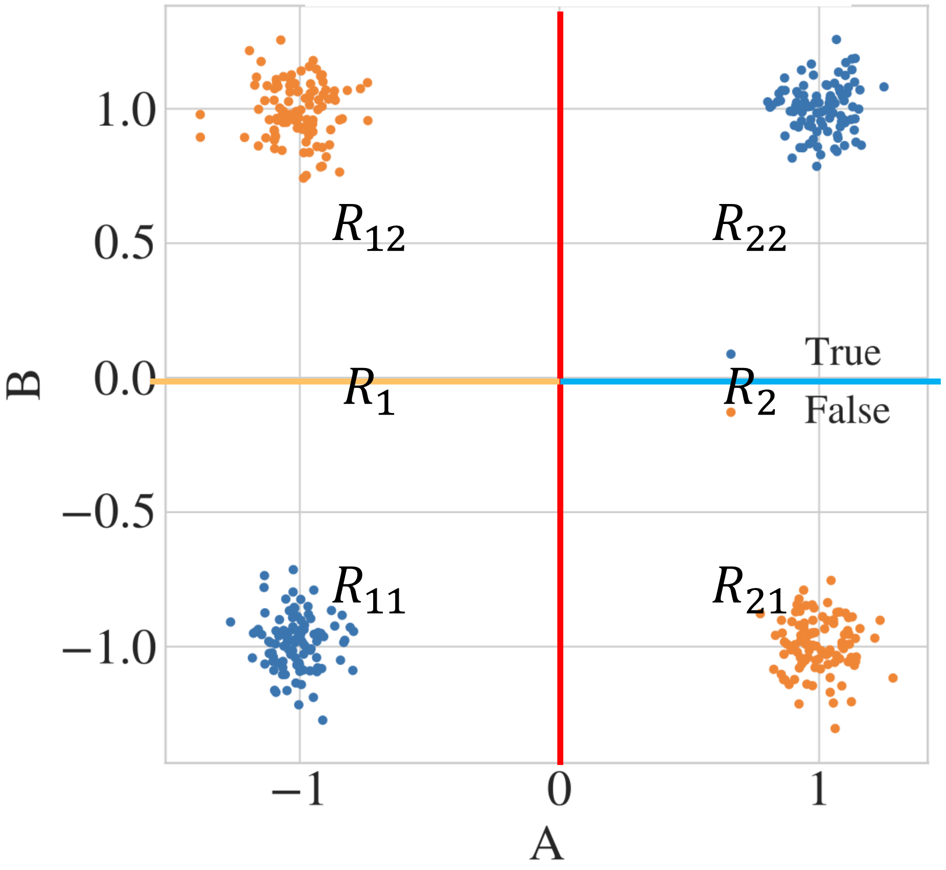
Classification

$$\ell(R_i) = -\sum_{k=1}^{q} \rho_k^i \log_2 \rho_k^i\,, \qquad \rho_k^i = \frac{\left|\{\boldsymbol{x}^{(j)} \mid \boldsymbol{x}^{(j)} \in R_i,\ \ y^{(j)} = k\}\right|}{\left|\{\boldsymbol{x}^{(j)} \mid \boldsymbol{x}^{(j)} \in R_i\}\right|}$$

Regression

$$\ell(R_i) = \frac{1}{N}\sum_{j=1}^{N}(y_i - m)^2\,, \qquad m = \frac{1}{N}\sum_{x^{(j)} \in R_i}^{N} y^{(j)}$$

Noisy XNOR dataset



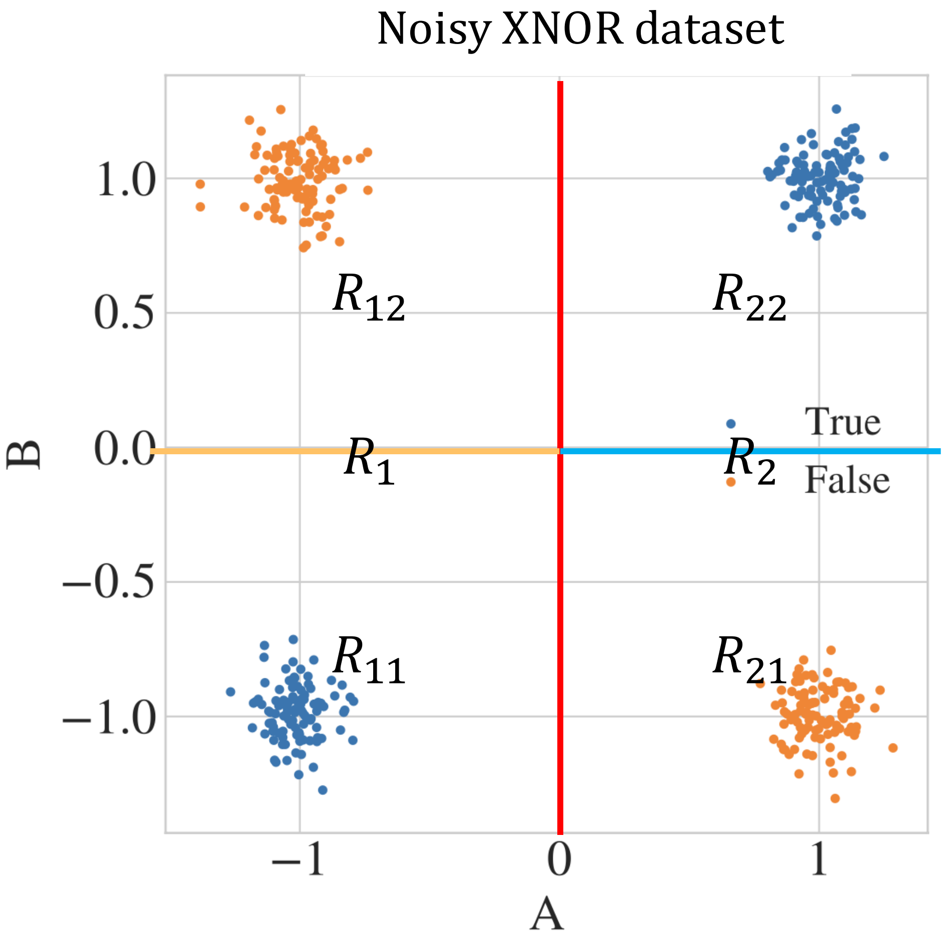
A more formal view of decision trees

Decision trees

- More formally, at a given node in parent region R_t asking a question about the j^{th} feature, we create two regions:

$$R_1 = \{\boldsymbol{x} \mid x_j < \alpha_t^j, \boldsymbol{x} \in R_t\}$$
$$R_2 = \{\boldsymbol{x} \mid x_j \geq \alpha_t^j, \boldsymbol{x} \in R_t\}$$

- The parameters $\boldsymbol{\theta}$ of decision trees are the threshold values at each nodes (the sequence of α)
- Decision tree minimise a loss function at each node of the tree: the **cross-entropy** or the **squared error** (classification vs regression)


Noisy XNOR dataset

⊕ can handle categorical values, easy to interpret, fast to compute, both regression and classification

⊖ Shallow trees: high bias estimators (underfit), deep trees: high variance estimators (overfit)

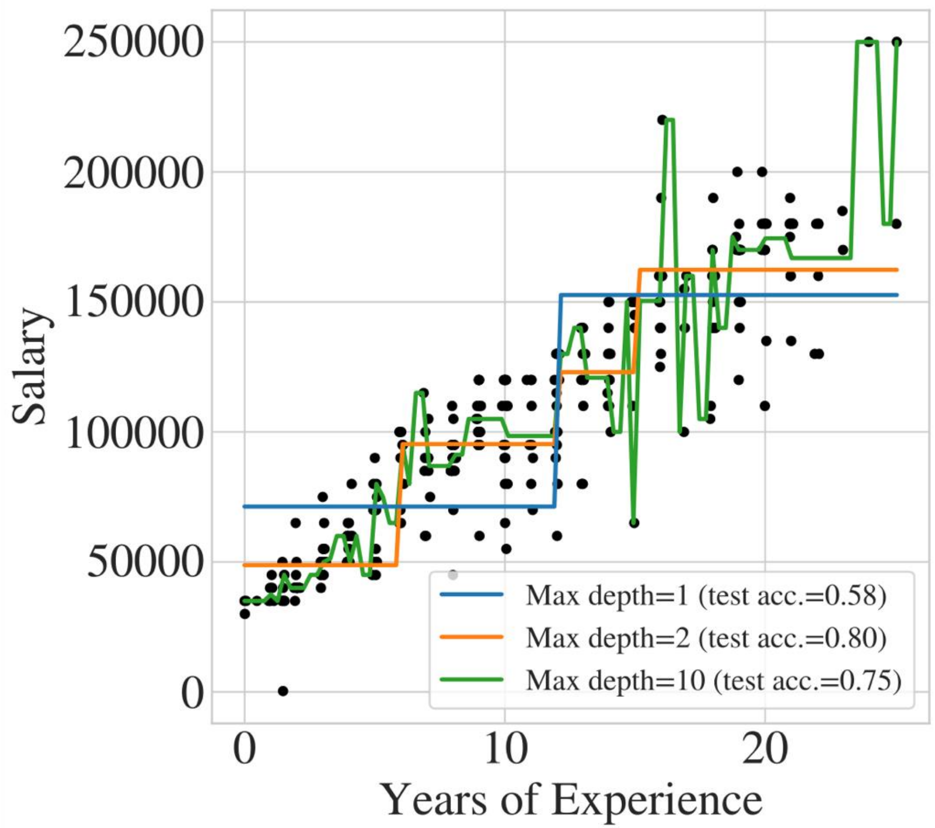
A more formal view of decision trees

Decision trees

- More formally, at a given node in parent region R_t asking a question about the j^{th} feature, we create two regions:

$$R_1 = \{\boldsymbol{x} \mid x_j < \alpha_t^j, \boldsymbol{x} \in R_t\}$$
$$R_2 = \{\boldsymbol{x} \mid x_j \geq \alpha_t^j, \boldsymbol{x} \in R_t\}$$

- The parameters $\boldsymbol{\theta}$ of decision trees are the threshold values at each nodes (the sequence of α)
- Decision tree minimise a loss function at each node of the tree: the **cross-entropy** or the **squared error** (classification vs regression)

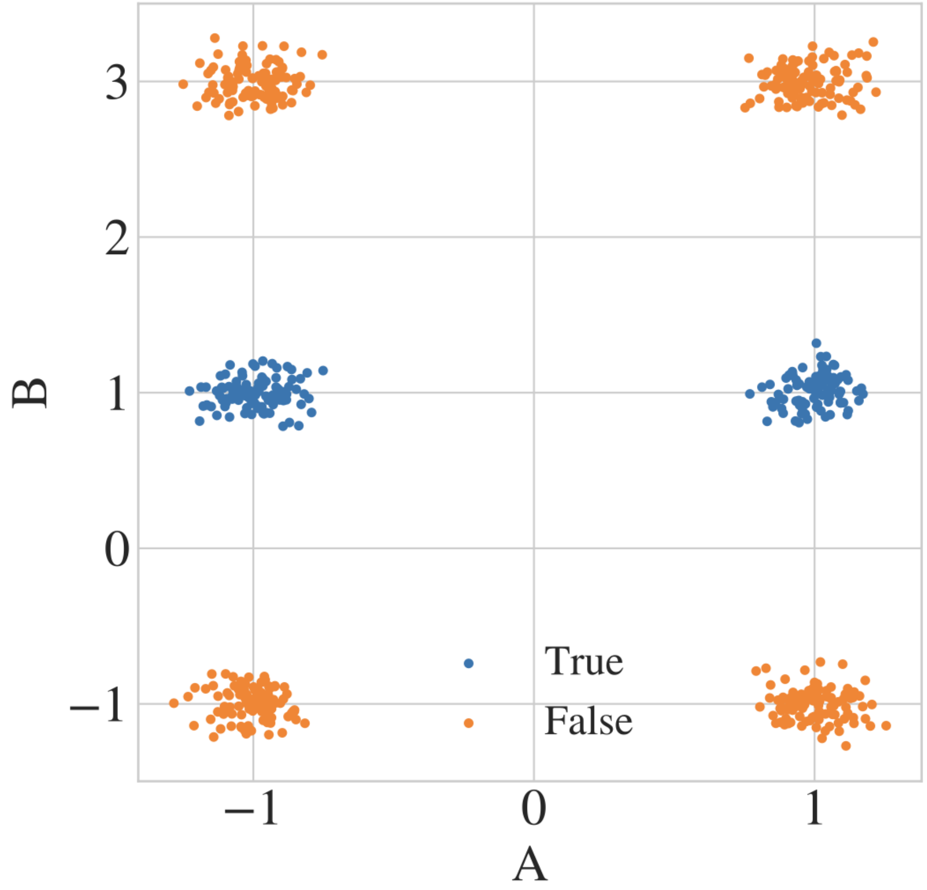


+ can handle categorical values, easy to interpret, fast to compute, both **regression** and classification

− Shallow trees: high bias estimators (underfit), deep trees: high variance estimators (overfit)

A more formal view of decision trees

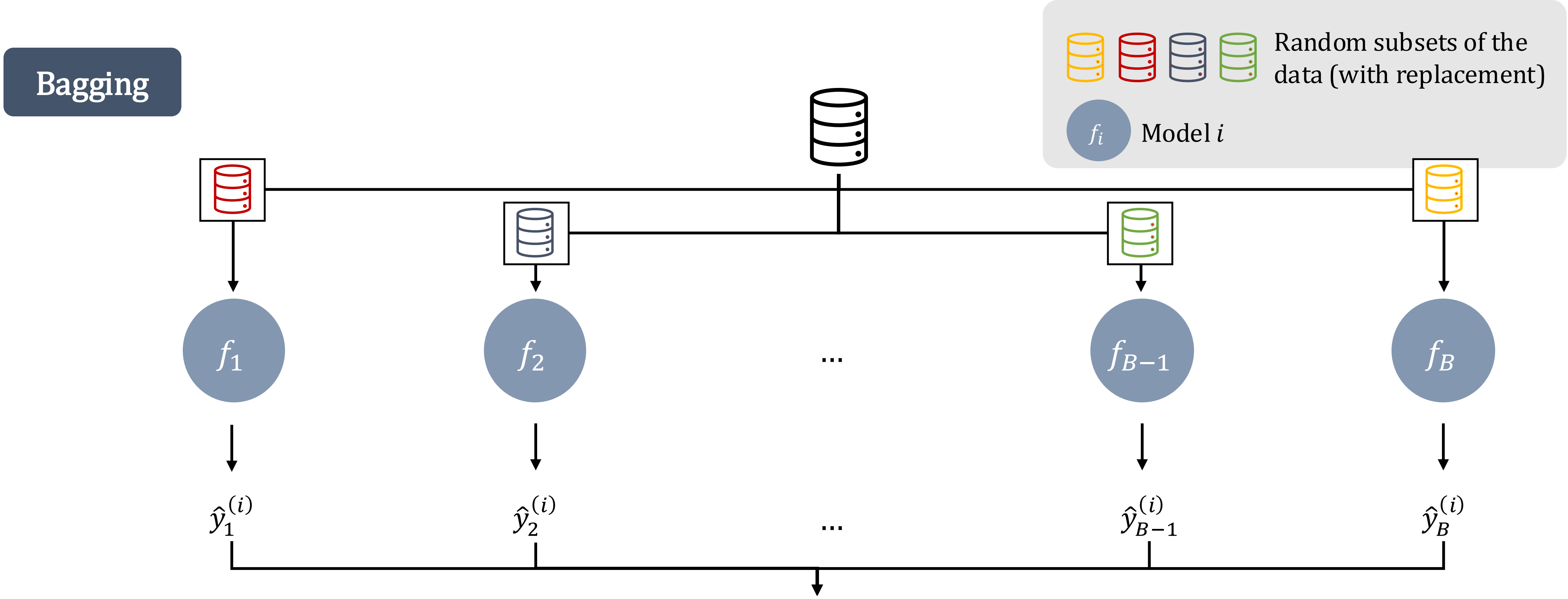
Decision trees



Practice: Can you build a decision tree solving the binary classification problem?

Ensembling methods: bagging

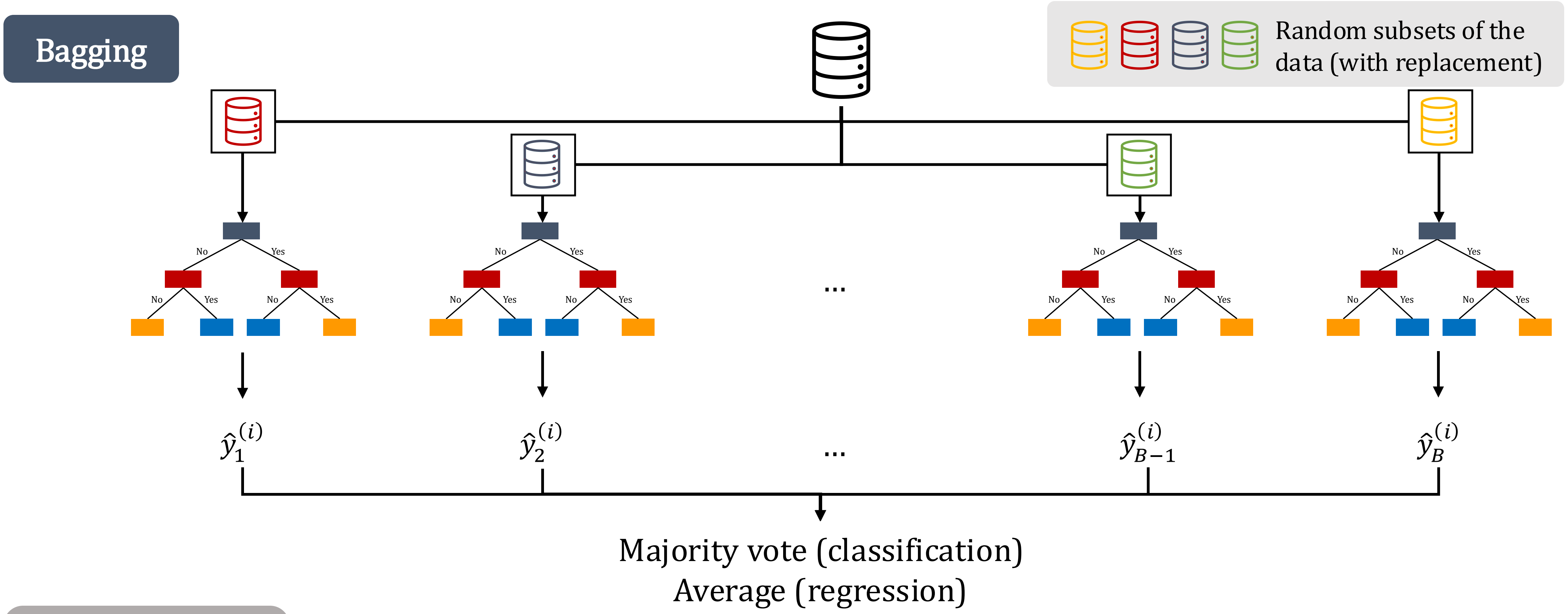
- To circumvent the problems that can have weak learners like decision trees, **ensembling methods** were proposed



Majority vote (classification)
Average (regression)

- To reduce the variance, models need to be **uncorrelated**: this is achieved by using **random sampling of the dataset**

Ensembling methods: random forest



Majority vote (classification)
Average (regression)

Random Forest = Bagging of decision trees + feature bagging

- ➕ Can still handle categorical values, both regression and classification, **reduced variance**
- ➖ More expansive to compute (need to train B trees instead of one), harder to interpret