

Artificial intelligence and chemistry

Tony Bonnaire

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



Given by: Tony Bonnaire (+ **Pablo Mas** for the project)



Format: Lectures + hands-on sessions then data challenge (in chemistry)



Exam: Paper analysis (50%) + oral presentation for the challenge (50%)



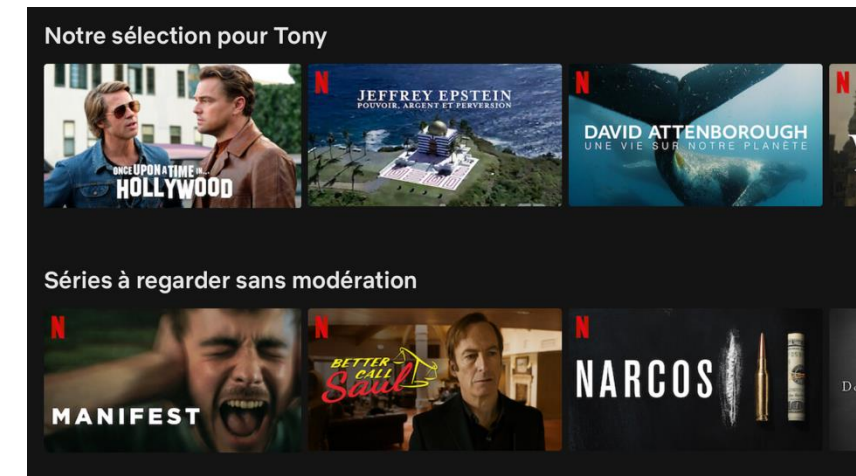
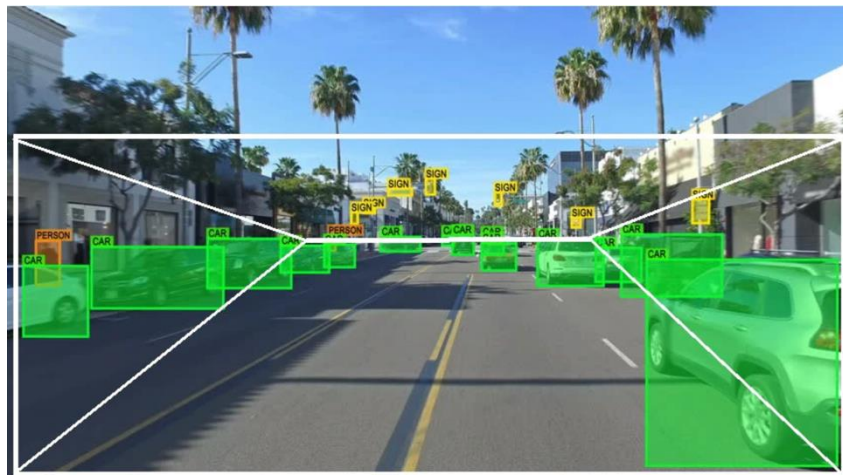
Aim: Introduce you to the basics of ML principles and carry out a project

Some references:

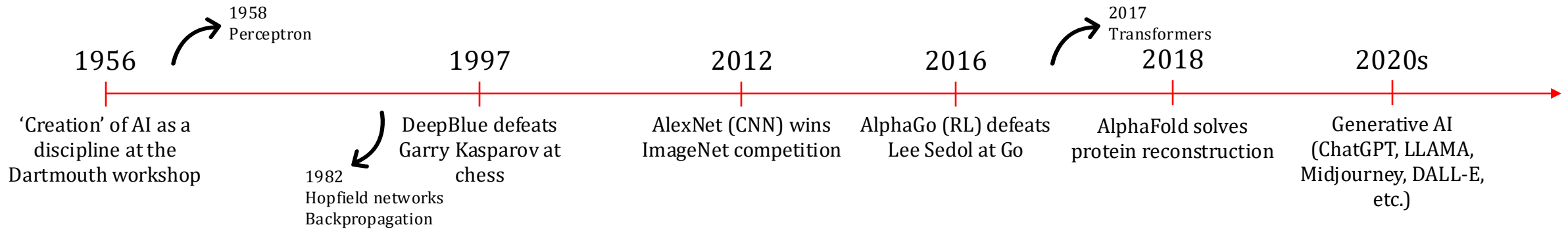
- [Deep Learning: Foundations and Concepts](#), Bishop & Bishop, 2023,
- [Deep Learning](#), Goodfellow et al., 2016,
- [Deep Learning with Python](#), Chollet, 2016,
- [Learning Theory from First Principles](#), Francis Bach, 2024,
- <https://challengedata.ens.fr>: a bank of data science challenges to apply all the things we will learn in this course

AI goal

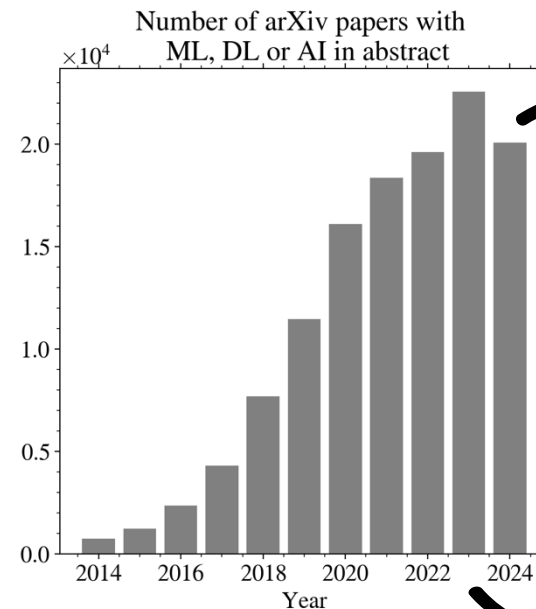
Design **systems** capable of performing complex tasks requiring *intelligence* (i.e. using reasoning, perception or language) **to take decisions** and **make predictions**.



Some (selected) AI breakthroughs



AI in science



2024 not over yet!

60 papers a day in average in 2023 (!)

Some scientific applications

Healthcare

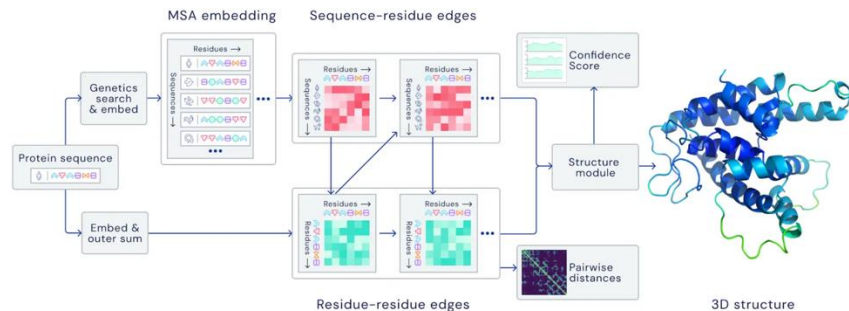
- Drug discovery
- Protein structure reconstruction

Astrophysics and cosmology

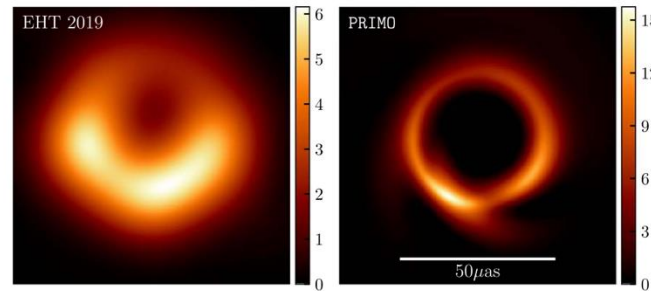
- Galaxy deblending
- Image restoration
- Source separation

Theoretical physics

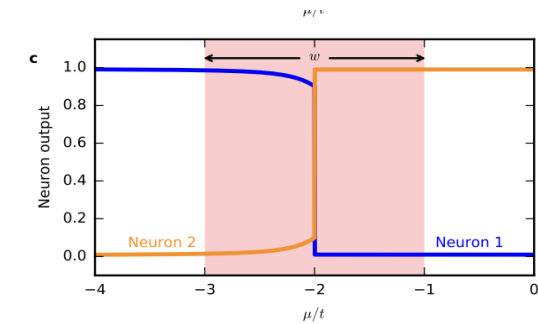
- Study phase transitions
- Discover experiments and equations



[Jumper et al., 2021](#)



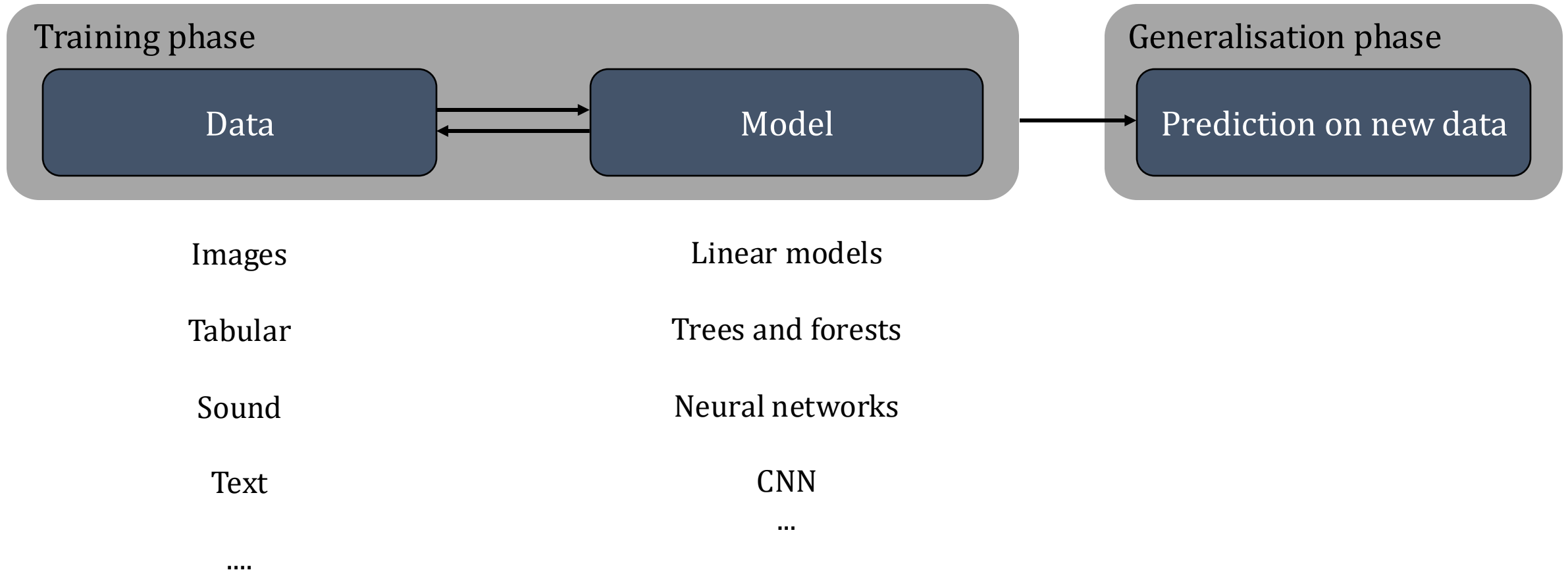
[Medeiros et al., 2023](#)



[Van Nieuwenburg et al., 2017](#)

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

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**.



What is “learning”? An example

6

Introduction to ML

Linear models

SL principles

Trees and neural networks

Risk optimisation

Training phase

Data

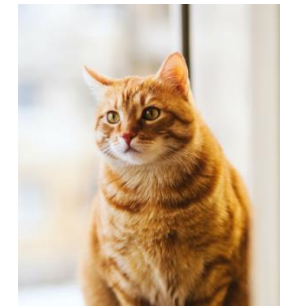
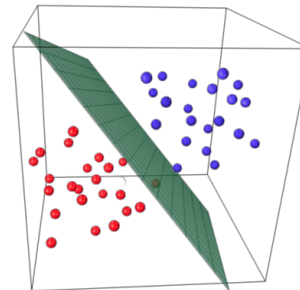
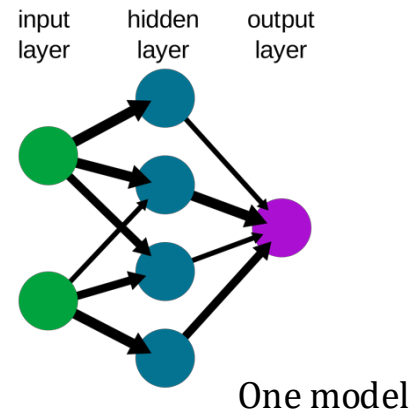
Model

Generalisation phase

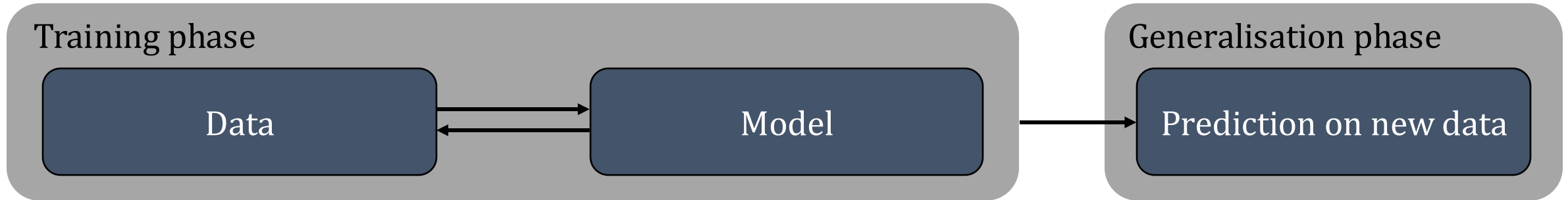
Prediction on new data



Images of a “cat” or “dog”

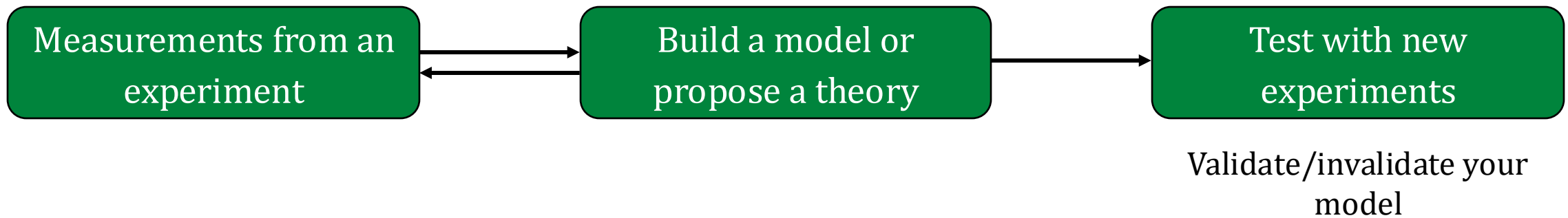


“cat” or “dog” ?

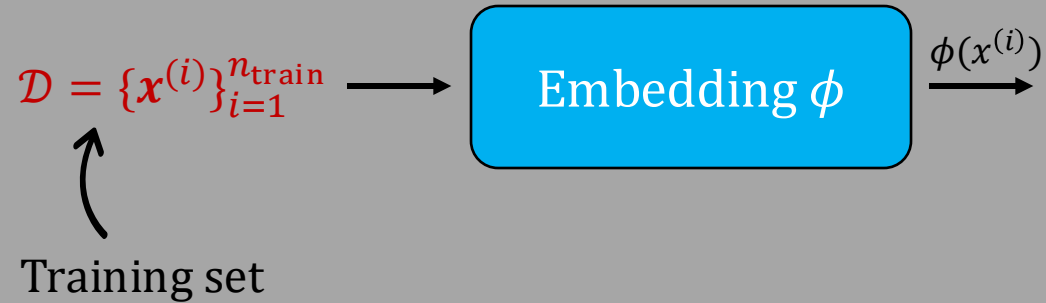


...In fact, all this is close to what you know!

The scientific method



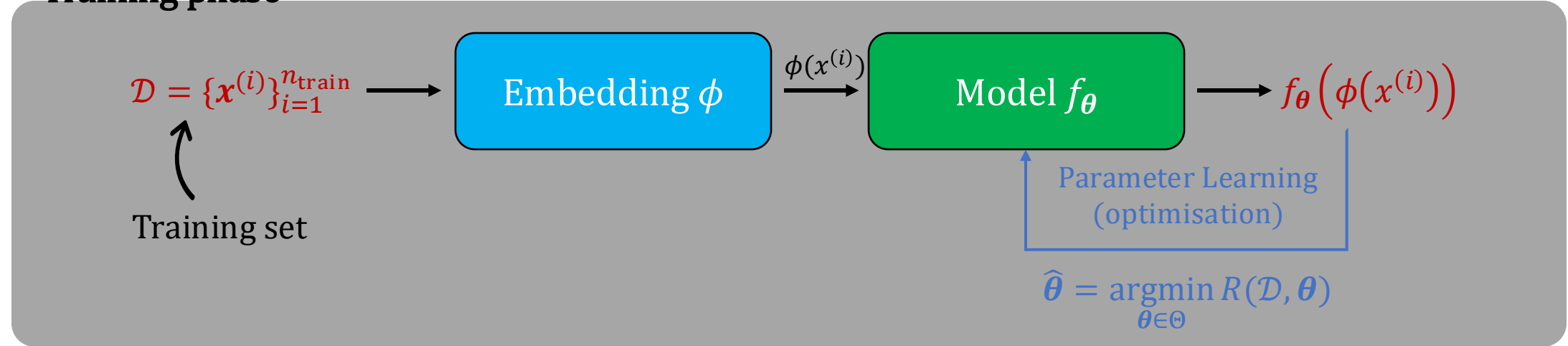
Training phase



1. Data are **unstructured**, sometimes **noisy** and **unprocessed** like pixels of an image or sequence of characters or words.
2. The embedding $\phi(\mathbf{x}^{(i)})$ is a **structured, numerical** vector representation of the data whose elements are **meaningful features** that 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 but makes it solvable using simple models.

Training phase

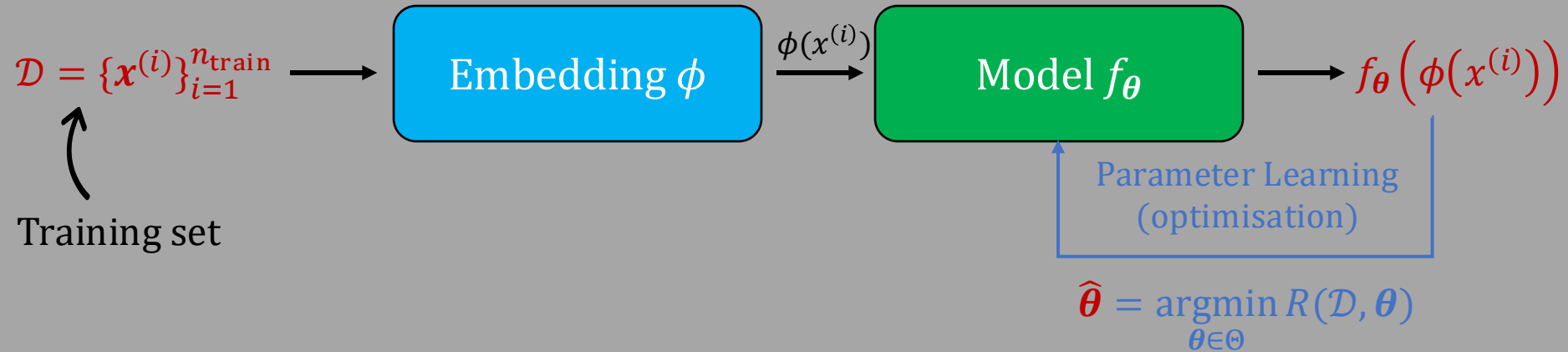


Some notations and terminologies:

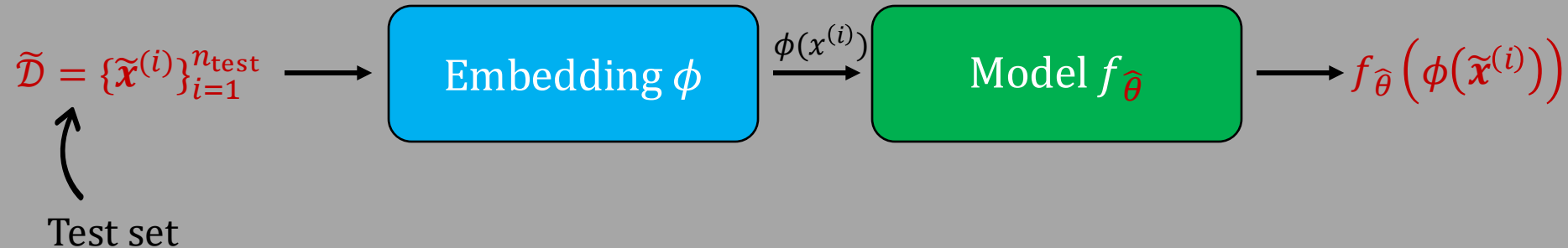
- $\mathbf{x}^{(i)} \in \mathbb{R}^d$ is *one training data* (there are n_{train} of them),
- $\phi(\mathbf{x}^{(i)}) \in \mathbb{R}^{d'}$ is an embedding of $\mathbf{x}^{(i)}$ sometimes called *feature vector*,
- $\theta \in \Theta \subset \mathbb{R}^p$ are the *parameters* of the model,
- $R(\mathcal{D},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

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

Training phase



Generalisation phase



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

Supervised learning

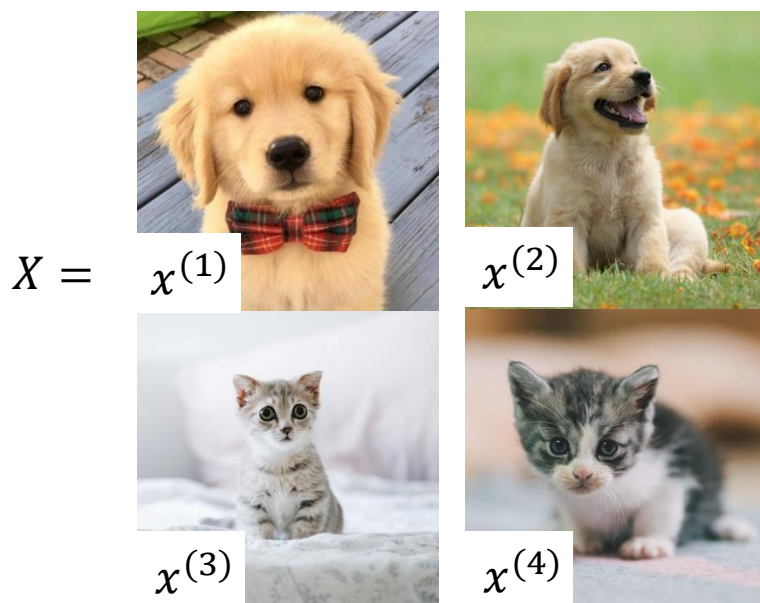
- Training data are actually X and Y coming as pairs

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n_{\text{train}}}, \quad (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in \mathbb{X} \times \mathbb{Y}$$

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



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



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

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

$$f_{\theta}(\mathbf{x}^{(i)}) = \hat{\mathbf{y}}^{(i)}$$



$$f_{\hat{\theta}}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} R(\mathcal{D}, \theta)$$

Training data

Model

Optimisation

Supervised learning

- Training data are actually X and Y coming as pairs

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n_{\text{train}}}, \quad (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in \mathbb{X} \times \mathbb{Y}$$

- 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** $\ell(y, \hat{y})$

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

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

$$\hat{R}(\mathcal{D}, \boldsymbol{\theta}) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}).$$

Supervised learning

- Training data are actually X and Y coming as pairs

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n_{\text{train}}}, \quad (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in \mathbb{X} \times \mathbb{Y}$$

- 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

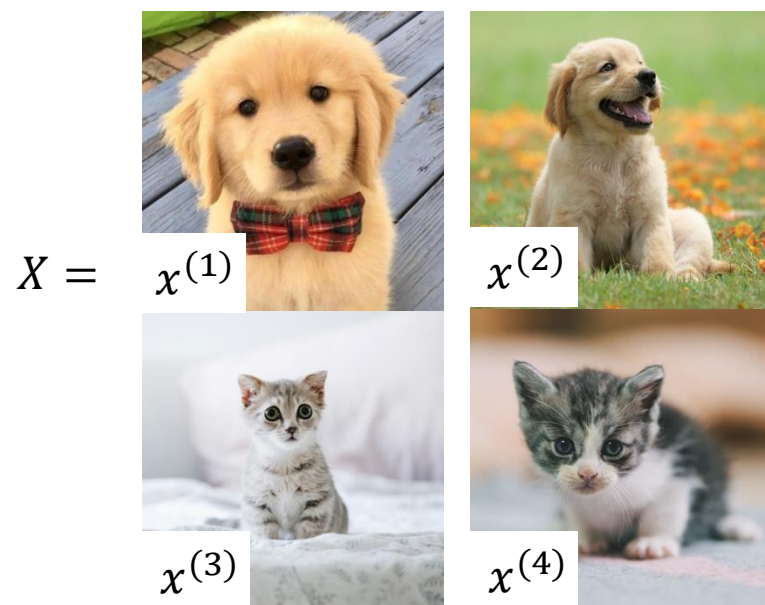
Naïve Bayes

Nearest neighbours

Unsupervised learning

- Training data are the set of $\mathbf{x}^{(i)}$'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

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



Training data

$$f_{\theta}(\mathbf{x}) = p_{\theta}(\mathbf{x})$$



$$f_{\hat{\theta}}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} R(\mathcal{D}, \theta)$$

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

Model

Optimisation

Unsupervised learning

- Training data are the set of $\mathbf{x}^{(i)}$'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

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

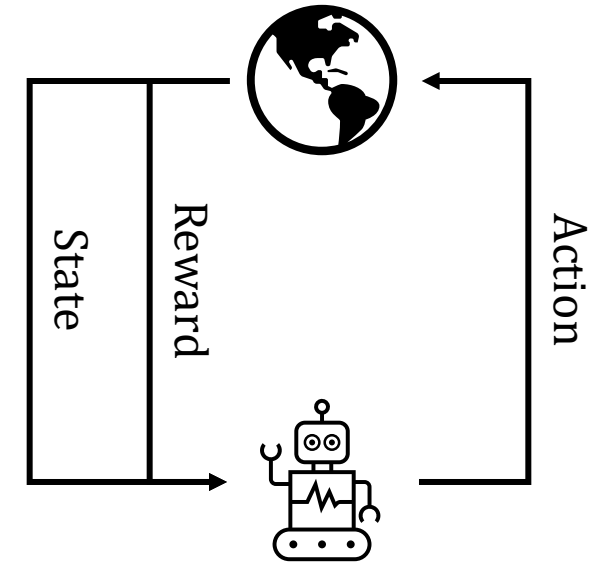
Autonomous driving

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](#)



$d \approx 10^6$

- 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

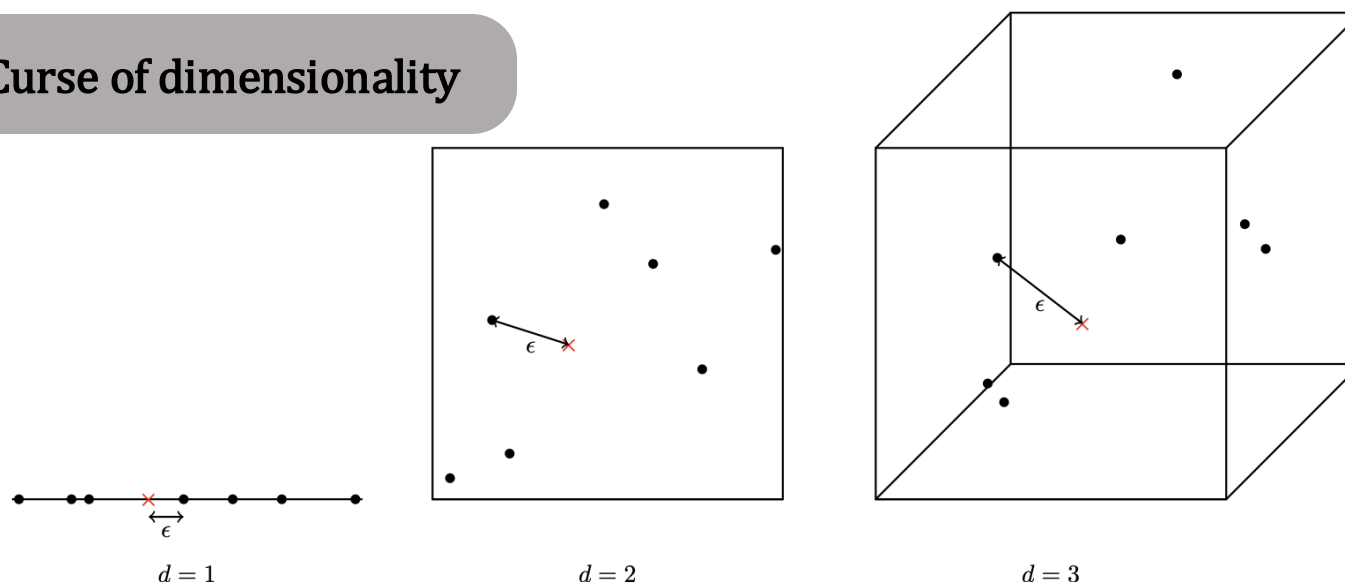
1-nearest neighbour

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

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

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

Curse of dimensionality





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:

- **Find an appropriate embedding** or feature representation of the data to simplify the problem,
- 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*

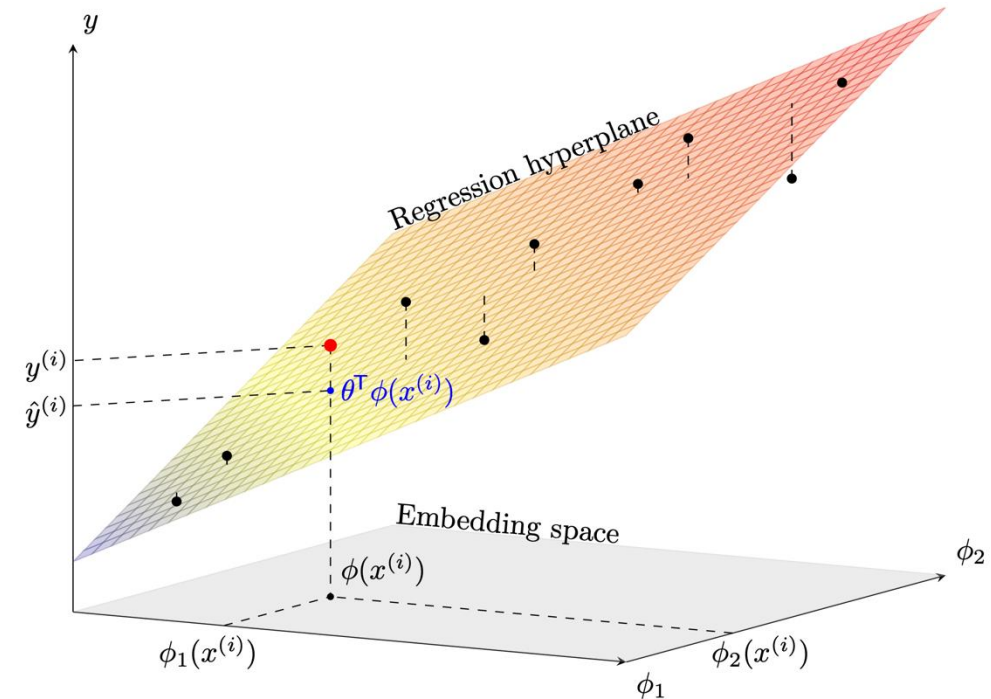


Linear regression

- 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

$$f_{\theta}(\phi_1^{(i)}, \phi_2^{(i)}, \dots) = \theta^T \phi^{(i)} = \hat{y}^{(i)}$$

- 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 $\theta = [\theta_0, \theta_1, \theta_2, \dots]^T$, $\phi^{(i)} = [1, \phi_1^{(i)}, \phi_2^{(i)}, \dots]^T$.
- **Geometric interpretation:** projection of an embedding vector onto a **hyperplane** parameterised by θ .



Linear regression

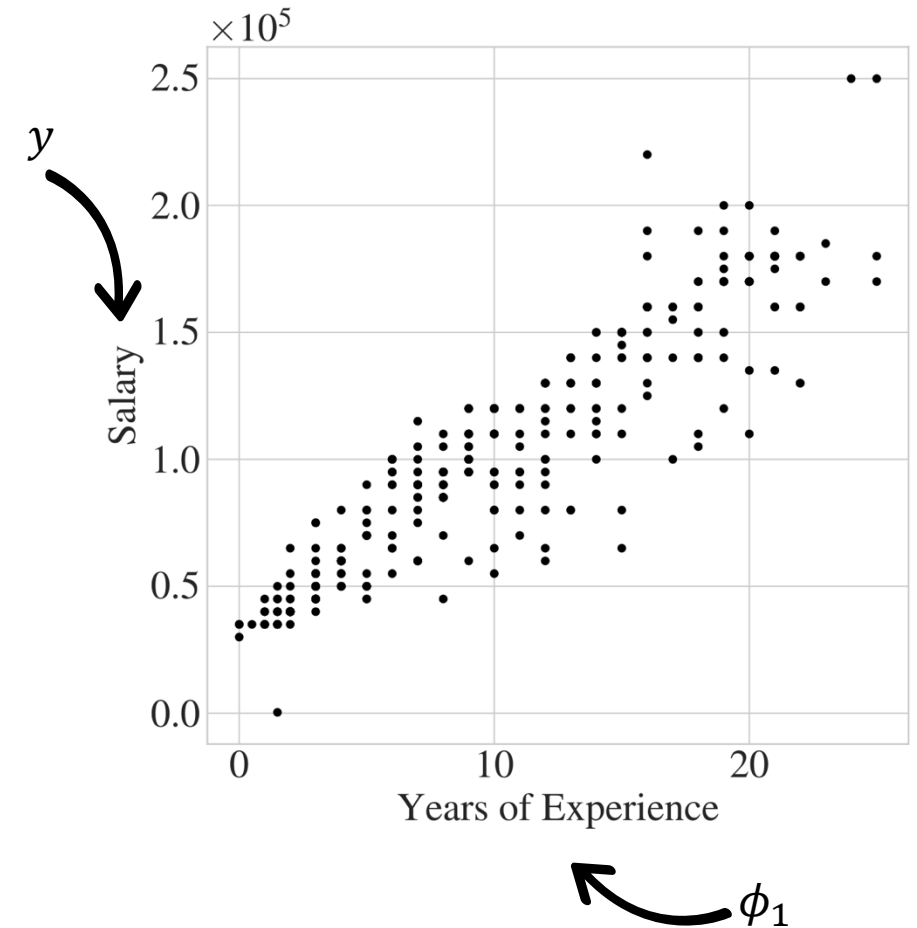
- Example: salary prediction based on the years of experience
- Data are $n = 373$ couples $(\phi^{(i)}, y^{(i)}) \Rightarrow$ **Supervised learning**
- The target variable $y \in \mathbb{R}$ is continuous \Rightarrow **Regression**
- The linear model is

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

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 $\hat{\theta}$, the best possible parameters for our model and data?
- This is done using **empirical risk minimisation (ERM)**

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} R(\mathcal{D}, \theta) = \underset{\theta}{\operatorname{argmin}} \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell(\hat{y}^{(i)}, y^{(i)})$$



Linear regression

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

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

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

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

Feature matrix

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

Target vector

$$\mathbf{y} = [y^{(1)}, \dots, y^{(n_{\text{train}})}]^T \in \mathbb{R}^{n_{\text{train}}}$$



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

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

Linear regression



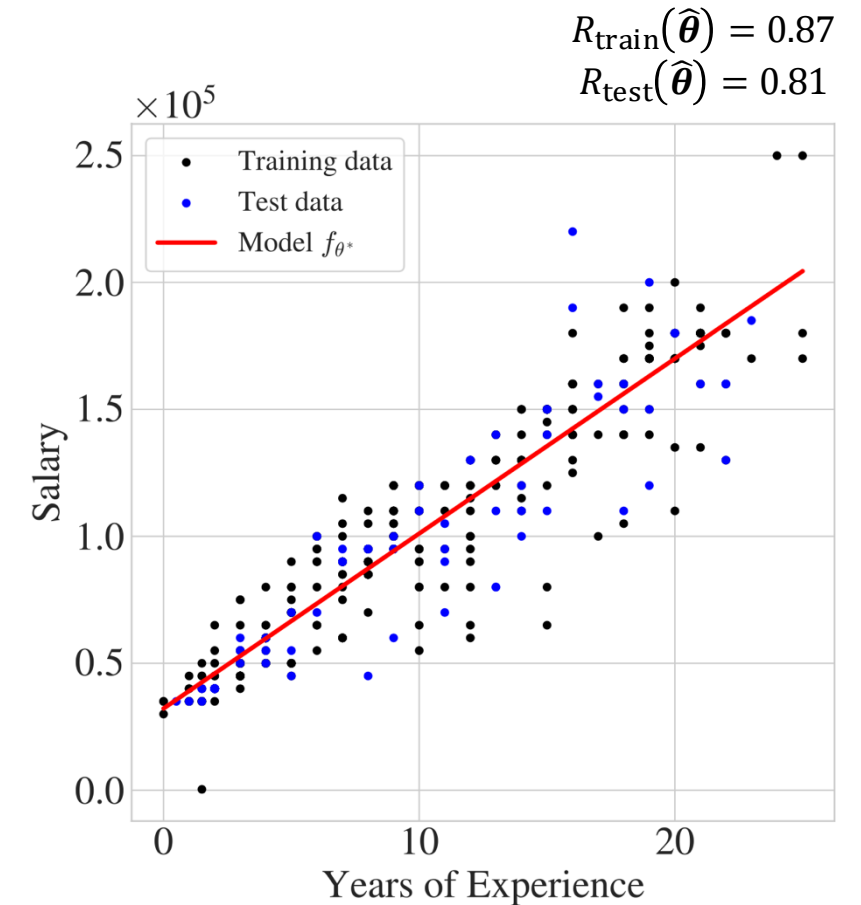
1. I **first** separated the dataset into **training and test sets**, $n_{\text{train}} = 0.8n$ and $n_{\text{test}} = 0.2n$ chosen randomly.

2. Then, I computed the optimal parameters minimising the empirical risk using the normal equations on the training features

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T y.$$

3. I computed the risk on the train and test sets and found they are close.

- + Exactly solvable model, low variance
- Cannot represent local relationships, may be biased



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

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

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

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

→ **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**

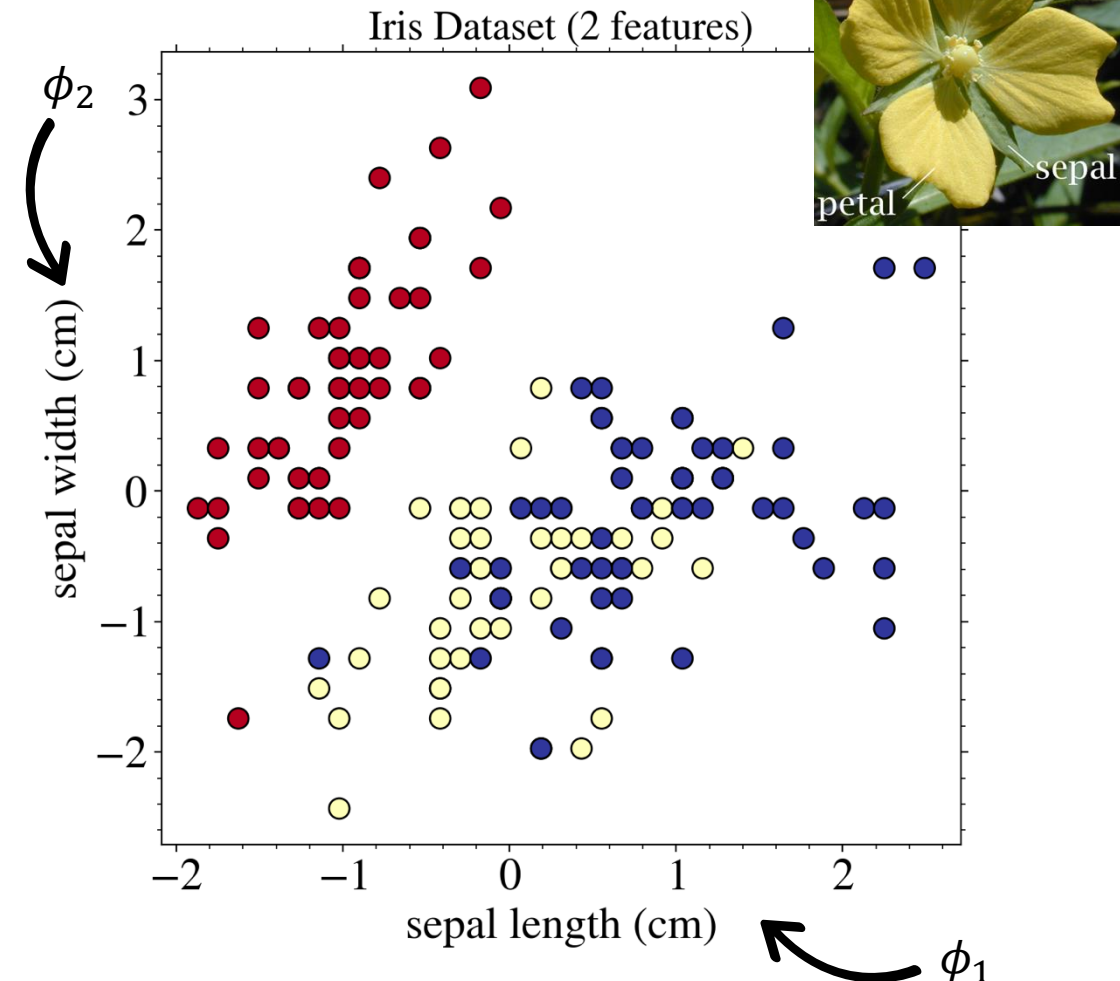
Linear classification

- Consider a K -class classification problem for which features ϕ allow linear separability
- Example: Iris dataset with $n = 150$ couples $(\phi^{(i)}, y^{(i)}) \Rightarrow$ **Supervised learning**
- The target variable $y \in \{0,1,2\} \Rightarrow$ **Classification**
- A natural loss function for classification is 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 decision rule (in Bayes sense) is

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



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

Linear classification

- The simplest models assume a linear log probability

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

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

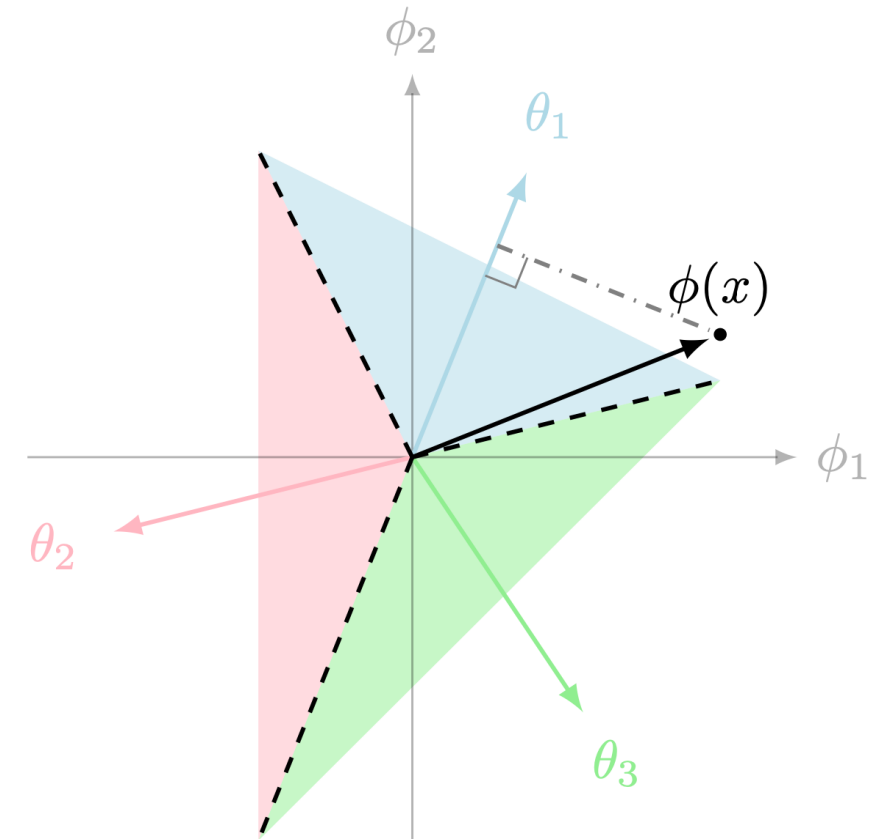
- It means that

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

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

- And the classification rule is

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



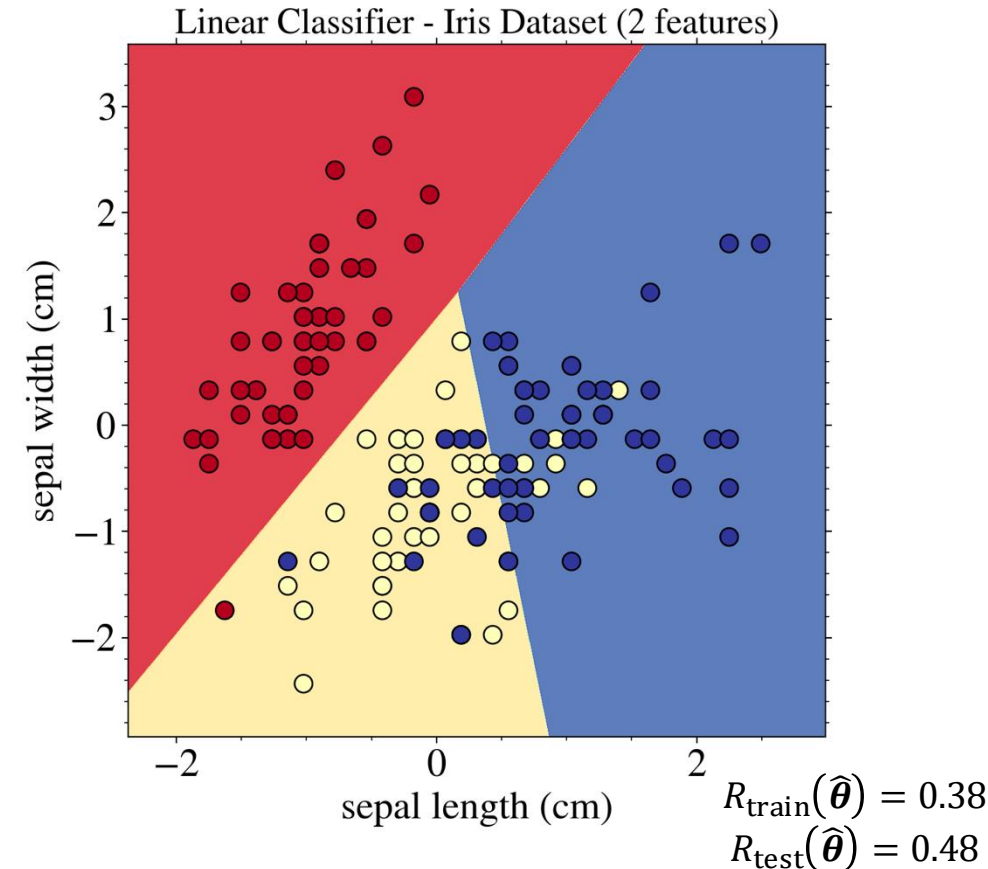
Geometrically, it corresponds to computing the overlap between the feature $\boldsymbol{\phi}^{(i)}$ and a vector representative for each class, $\boldsymbol{\theta}_k$, and associating **the class maximising the dot product**, leading to **linear decision** boundaries shown as hyperplanes.

Linear classification

- Now the model is specified, we need minimise the risk to obtain the parameters θ_k using some training data
- For optimisation, the 0-1 loss is not suitable since it is not differentiable, but we can relax it using the probabilities

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

which is **now differentiable and convex**.



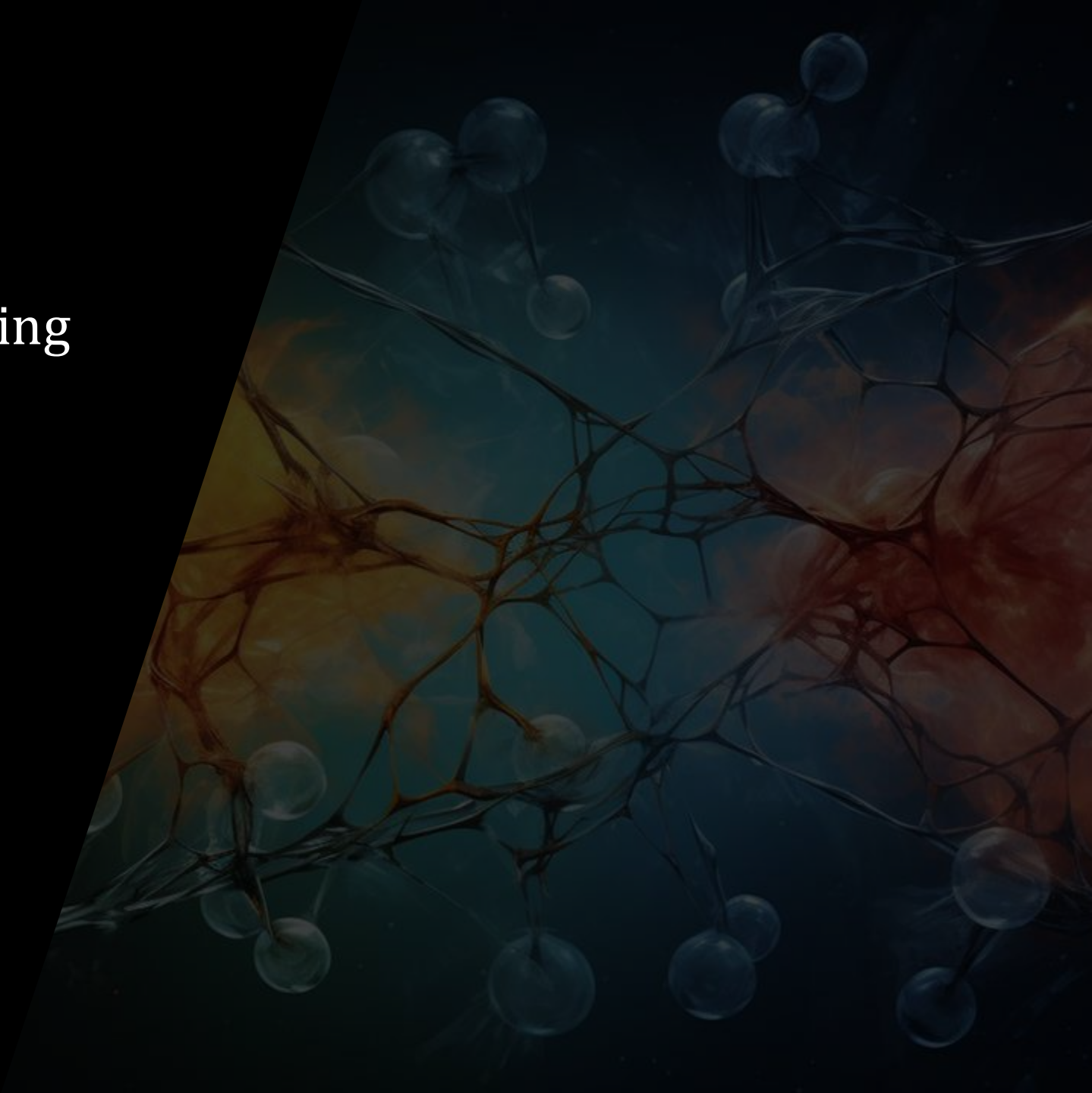
This risk is referred to as **cross-entropy** and it is the most widely used cost function for classification problems. The parameters of the model are then obtained by minimising the risk, i.e.

$$\hat{\theta} = \operatorname{argmin}_{\theta} R(\mathcal{D}, \theta).$$

| Principles of Supervised Learning

Contents:

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



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 $f_{\theta}(\phi^{(i)})$

Now what could possibly go wrong with our models?

Generalisation!

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

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

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

$$\mathbb{E}[(y - f_{\theta}(\tilde{\mathbf{x}}))^2] = \text{Bias}[f_{\theta}(\tilde{\mathbf{x}})]^2 + \text{Var}[f_{\theta}(\tilde{\mathbf{x}})] + \sigma_{\epsilon}^2$$

Modelling
error

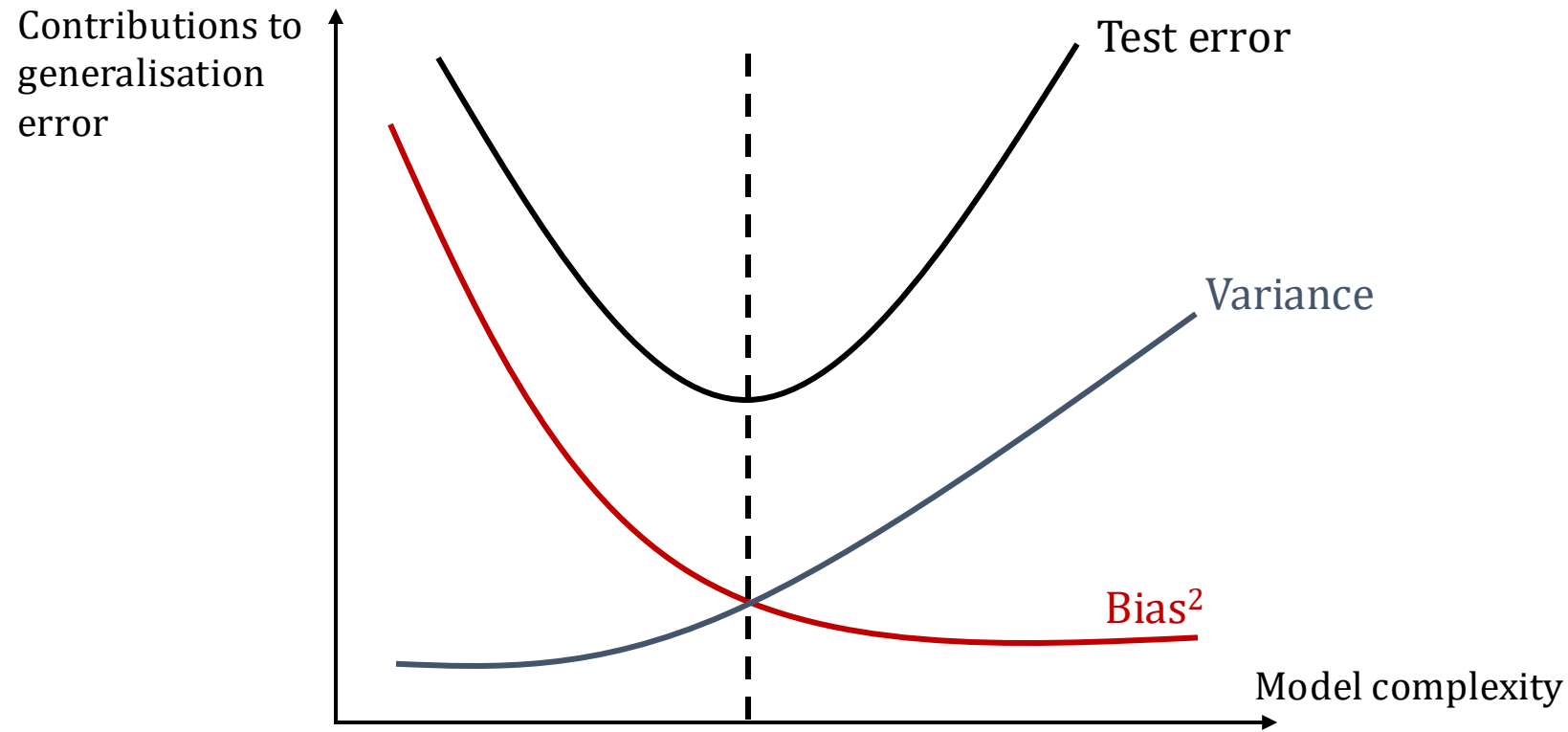
Model
variability

Irreducible
error

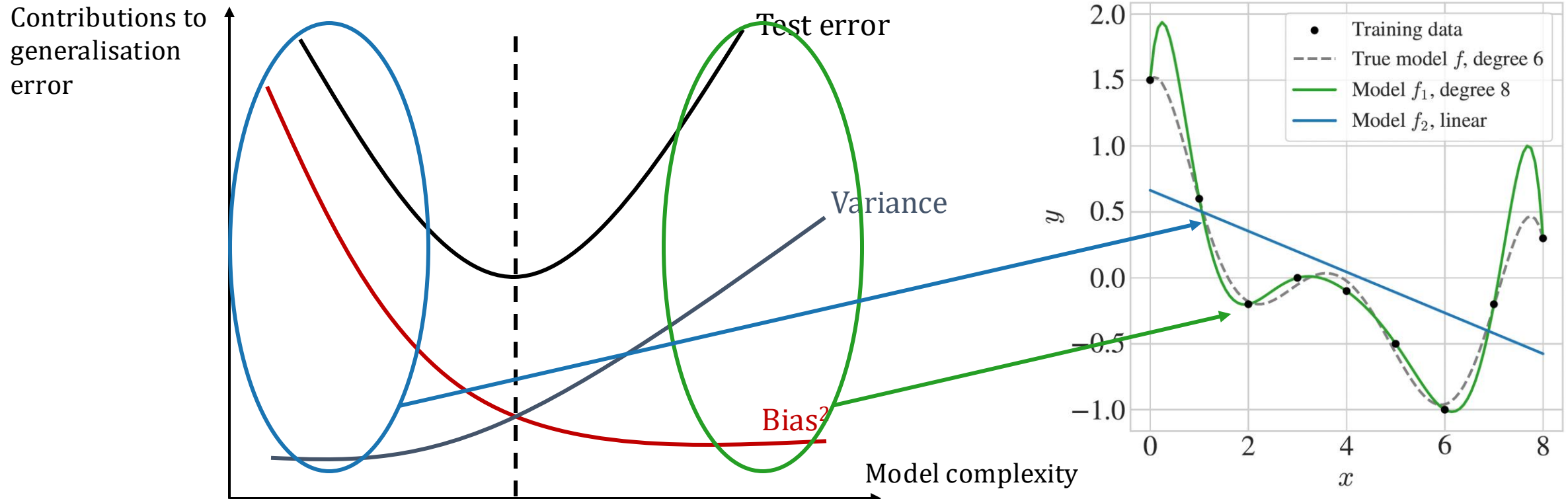
Note: Every terms are >0

Note: a similar expression holds for classification

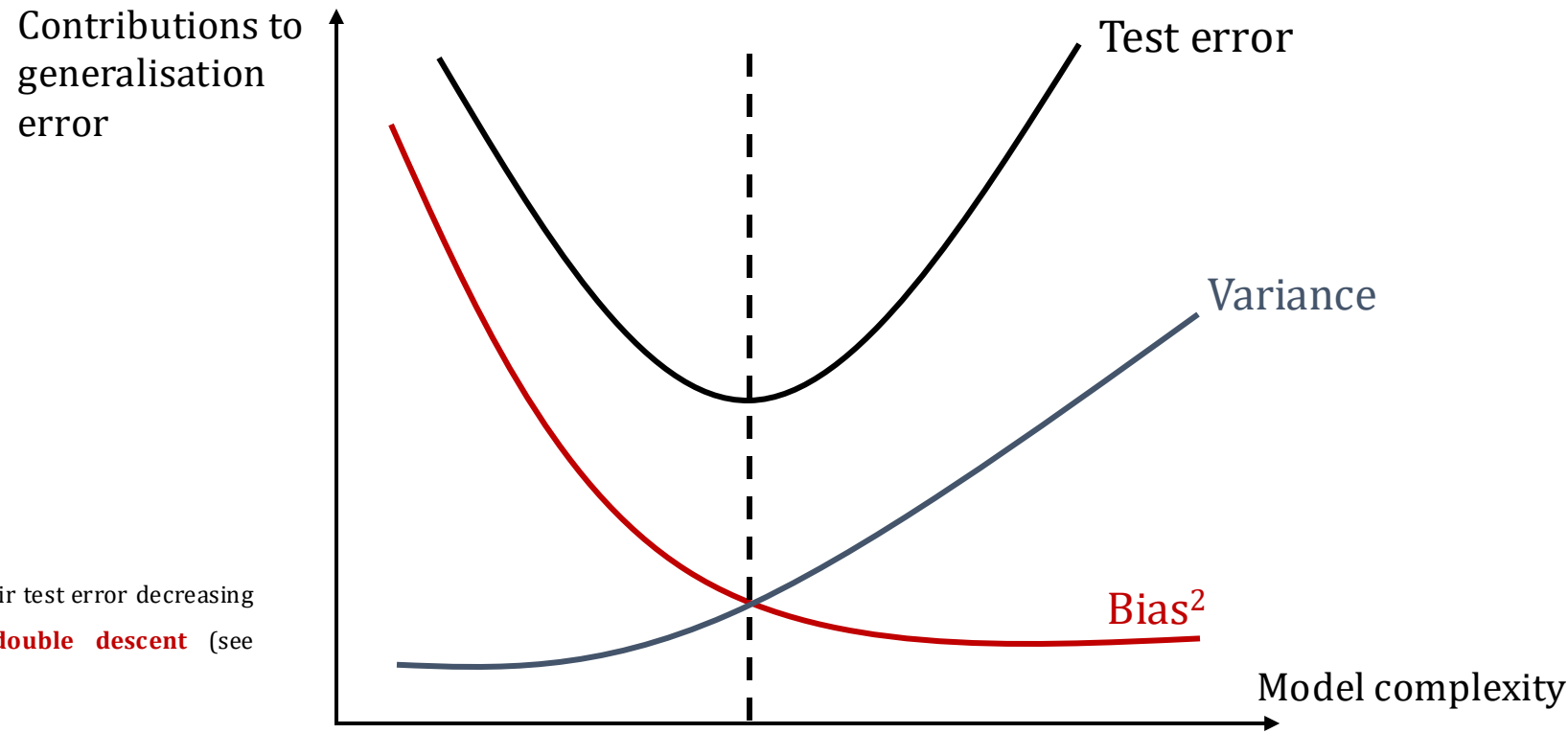
- **“Simple” models** have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- **“Complex” models** (with a lot of parameters for instance) have **small bias**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



- **“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**



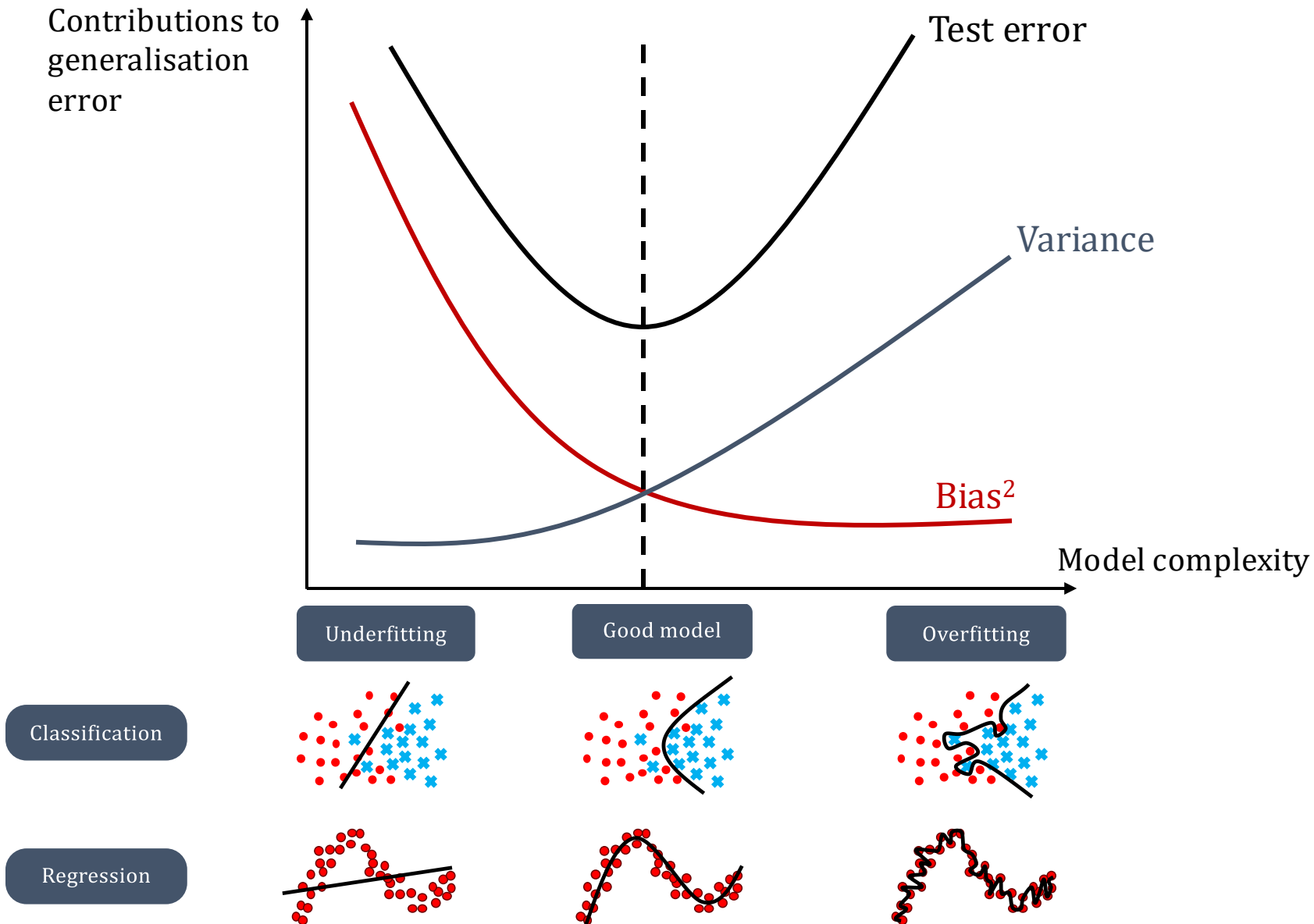
- **“Simple” models** have **large bias** because they constrain very much the function class that is therefore far from the truth, but they usually have **low variance** and are robust to variations of the training set
- **“Complex” models** (with a lot of parameters for instance) have **small bias**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



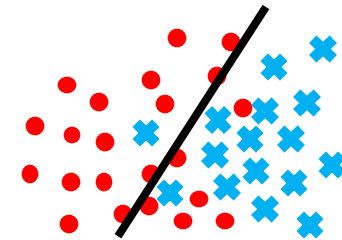
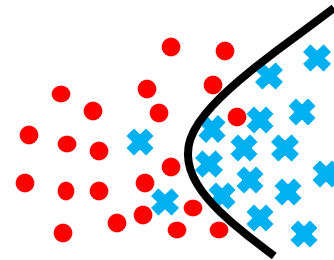
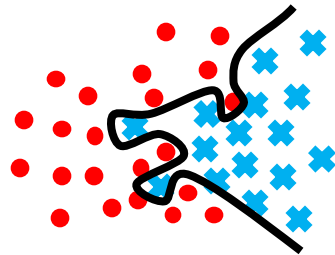
- To measure if we really learnt something useful in supervised learning in practice, we use a **test dataset** that the model has never seen but for which we know the labels and check that $R_{\text{train}}(\hat{\theta})$ and $R_{\text{test}}(\hat{\theta})$ are of the same order
- Based on the previous 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

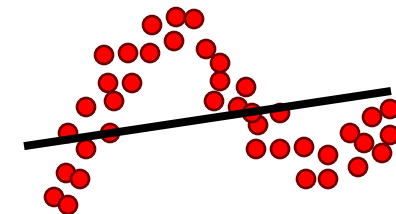
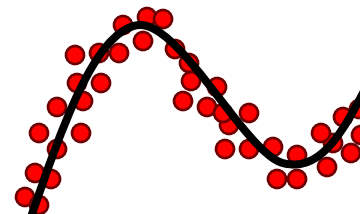
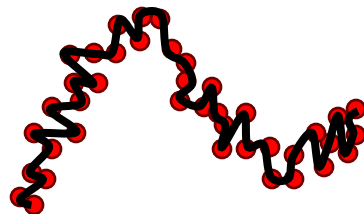
Good model

Underfitting

Classification



Regression



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

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

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

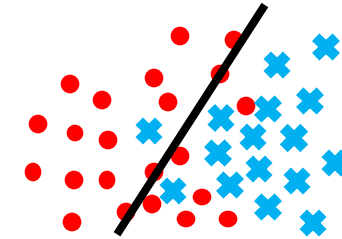
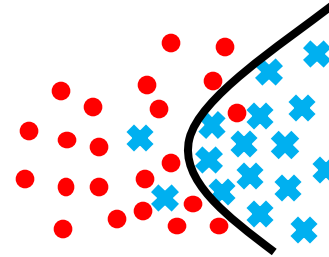
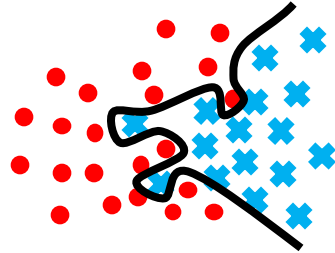
but large

Overfitting

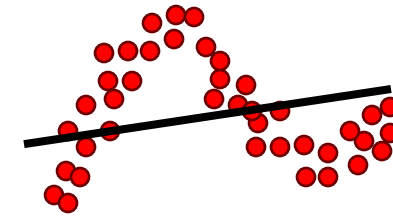
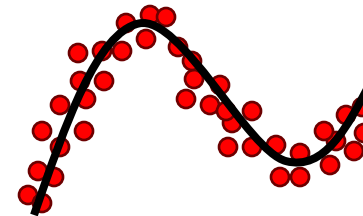
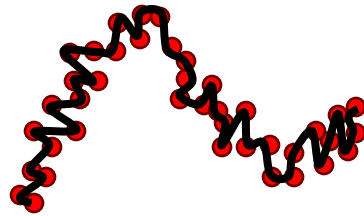
Good model

Underfitting

Classification



Regression



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

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

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

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

- Back to our regression problem: what happens when $n_{\text{train}} \approx d'$? Remember the solution

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

