

Data Science - Machine Learning

Msc Supply Chain

Guillaume Metzler

Université Lumière Lyon 2
Laboratoire ERIC, UR 3083, Lyon

guillaume.metzler@univ-lyon2.fr

Fall 2026-2027

Plan of the session (3 hours)



1.

Introduction

what is ML,
data, types of
learning



2. Learning

risk, loss,
validation,
regularisation



3. Algorithms

regression,
 k -NN, trees,
SVM



4. Guarantees

generalisation
bounds

Two running examples: **will this shipment arrive late?** (supply chain)
and **is this picture a cat or a dog?** (the classic).

Each part ends with three comprehension questions; the answers are on the next slide.

What is Data Science?

What is Machine Learning?

Before starting: working definitions

- **Data Science:** extract knowledge and decisions from data. It covers *collecting, cleaning, exploring, modelling, communicating*.
- **Machine Learning:** the part of data science where a model is *learned from examples* instead of being programmed by hand.

Data science question: *why are some shipments late?*

Machine learning question: *will **this** shipment be late?*

→ ML is the *predictive engine* inside a data-science project.

Introduction

What is machine learning?

What is Machine Learning? I

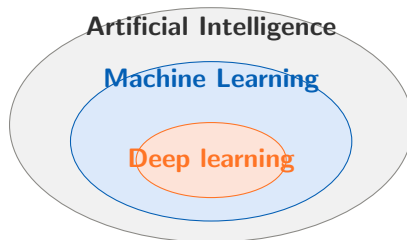
Definition 1.1: learning (Mitchell, 1997)

A program **learns** from experience E with respect to a task T and a performance measure P , if its performance on T , measured by P , **improves with experience** E .

Task T : predict whether a shipment will be late
Experience E : 3 000 past shipments, with their outcome
Performance P : share of correct predictions on *new* shipments

→ If one of T , E , P is missing, it is not a learning problem.

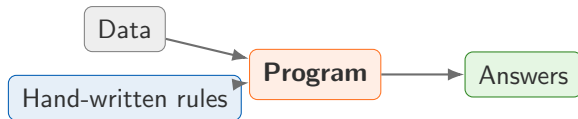
What is Machine Learning? II



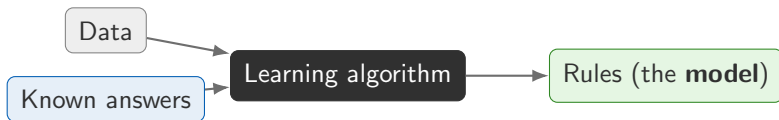
- ML is a **sub-field of AI**, at the frontier of computer science and applied mathematics: **statistics, optimisation, linear algebra**.
- It is part of **Data Science**: it also requires collecting, cleaning and analysing data.
- **Deep learning** = ML with many-layer neural networks.

The big shift: from rules to learning

Classical programming:



Machine learning:



→ In ML the rules are the *output*, not the input.

A formal view of learning

Definition 1.2: learning algorithm

A **learning algorithm** $\mathcal{A}$ takes a training sample S and returns a function (a **hypothesis**, or model) $h_S \in \mathcal{H}$:

$$\mathcal{A}: S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m \mapsto h_S: \mathcal{X} \rightarrow \mathcal{Y}$$

- $\mathcal{H}$ is the **hypothesis space** (e.g. all linear functions, all trees of depth ≤ 5).
- h_S depends on the random sample S : h_S is itself random.
- Goal: $h_S(\mathbf{x}) \approx y$ on *new* pairs $(\mathbf{x}, y)$.

The three ingredients of an ML project



1. The task

which problem, which
decision?



2. The data

examples available for
learning



3. The algorithm

formulate the problem
mathematically, then
solve it

Example	Task	Data
Cats & dogs	name the animal	labelled photos
Delivery	late or on time?	past shipments
Demand	units next month	sales history

→ Identify the task precisely before choosing any algorithm.

Machine learning in supply chain & risk



Demand forecasting

regression: units per week



Late-delivery alert

classification: late or not



Supplier default

classification: probability of failure



Segmenting suppliers

clustering: no labels

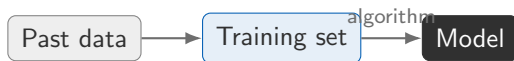
Not every problem is a learning problem: a **first-ever** event (a new tariff) has no history; use scenarios and simulation instead.

Question: the two phases

What are, for you, the two main phases of a machine learning project?

Answer: learn, then use

Phase 1: learn



Phase 2: test, deploy, monitor



→ Phase 1 builds the model on past data; phase 2 uses it on data it has never seen.

Phase 1: learning (training)

- Learn the **parameters** of a model previously chosen to solve our problem.
- It requires data: more precisely, a **subset** of the available data (the training set).
- Tasks can be very different: classify documents, tell cats from dogs, predict stock prices, predict a delay.
- It is a preparation phase: the model is learned and tested before being delivered to the customer.

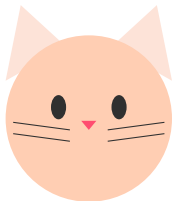
→ A model good on the training data is not necessarily good on future data.

Phase 2: test and production

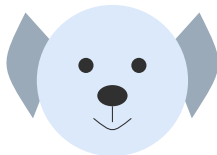
- **Test:** putting the model *in production, but virtually*, on data it has never seen, to be sure of its performance.
- **Deployment:** the system, developed, validated and tested, is used in practice.
- **Monitoring:** effectiveness must remain in line with what was observed at training time.
- **Re-training** on more recent data when performance drops (data drift).

→ A model is not finished when it is trained: it must be monitored and updated.

Data : cats and dogs I



a cat (label $y = -1$)

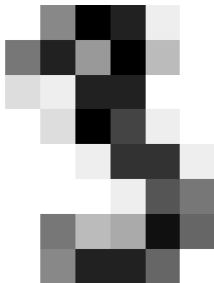


a dog (label $y = +1$)

An algorithm does not “see” a cat: it receives **numbers**. How do we turn a picture into numbers?

Data : cats and dogs II

What we see



What the computer sees

0	0	7	15	13	1	0	0
0	8	13	6	15	4	0	0
0	2	1	13	13	0	0	0
0	0	2	15	11	1	0	0
0	0	0	1	12	12	1	0
0	0	0	0	1	10	8	0
0	0	8	4	5	14	9	0
0	0	7	13	13	9	0	0

- A grey-level image is a **matrix** of pixel values (0 = black, 255 = white).
- A colour image superposes **three matrices** (red, green, blue): a tensor of size height $\times$ width $\times$ 3.

Data : cats and dogs III

Flatten the three channels into one vector: $R = (1, 1, 0.9, 0, \dots)$, $G = (1, 1, 0.1, 0.3, \dots)$, $B = (1, 1, 0, 0.7, \dots)$ give

$$\mathbf{x} = (R, G, B) \in \mathbb{R}^d, \quad d = 3 \times \text{height} \times \text{width}.$$

- A small 224×224 photo already gives $d = 150\,528$ coordinates.
- Two photos of the same cat have very different vectors (light, angle):
pixels are a poor representation.

→ Representation matters as much as the algorithm.

Data : cats and dogs IV

Data are at the heart of learning algorithms. They can be:

- **raw** (not transformed): pixels, a timestamp, a free-text comment;
- **transformed or created**: average colour, weekday, distance in km;
- **learned**: a representation found by the algorithm (neural networks, auto-encoders).

A major difficulty specific to learning: [these data come from an unknown distribution](#), which opens many research avenues in ML.

→ [We never see the whole world, only a sample of it.](#)

Data : cats and dogs V

Descriptors are **not invariant**: they depend on the collection conditions.

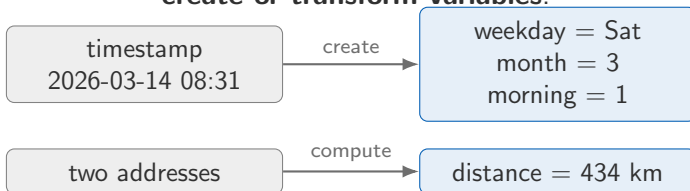
Images: size (distance to the camera), shape (viewpoint), colour (light). Some algorithms, such as SIFT, find identical elements across different images.

Shipments: “lead time” in days or hours? “distance” in km or miles? Measured at the dock or at the customer?

→ Always ask how, when and by whom a variable was measured.

From raw data to features

It is rarely enough, especially in an industrial context, to use raw data: we **create or transform variables**.



- with **business knowledge**: experts know the discriminating factors;
- with **mathematical or learned transformations**: kernels, metric learning, neural networks, auto-encoders.

Input, output and training set

Definition 1.3: training set

A **training set** is a finite sample $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$: $\mathbf{x}_i$ is the **input** (features) of example i , y_i its **output** (label).

distance (km)	mode	order size	hub load (%)	reliability	peak	lead time (d)	late
434	road	41	91.70	0.85	0	7.02	1
348	air	49	42.40	0.74	1	3.46	0
298	sea	32	71.50	0.94	1	9.53	0
454	sea	46	75.80	0.81	0	8.79	0
288	rail	45	91.90	0.83	1	8.13	1

Each row is an example $(\mathbf{x}_i, y_i)$; the last column is the label y_i (cats and dogs: $\mathbf{x}_i$ = image vector, $y_i \in \{\text{cat}, \text{dog}\}$).

→ Rows = examples (m), columns = features (d).

Data and representation

Data come in many formats: structured or not, sometimes with anomalies or **missing values**. In ML we view them as m examples of the same nature, each a vector

$$\mathbf{x} = (x_1, \dots, x_d) \in \mathbb{R}^d, \quad X = \begin{pmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{md} \end{pmatrix} \in \mathbb{R}^{m \times d}.$$

- **Categories** become 0/1 columns (*one-hot*): road = (1, 0, 0), air = (0, 1, 0), sea = (0, 0, 1).
- **Missing values**: remove, impute (mean, median, model), or add an indicator.

Shipment

(434 km, road, 41, 91.7, 0.85, 0) $\Rightarrow \mathbf{x} = (434, 1, 0, 0, 41, 91.7, 0.85, 0) \in \mathbb{R}^8$.

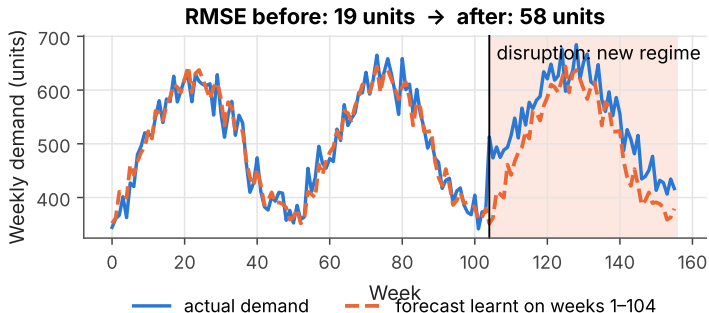
Other representations

The meaning of the vector depends on the nature of the data:

- **images**: pixels, or the output of a CNN (after vectorisation);
- **gene sequences**: dimension = length, vector = encoding of the bases A, C, G, T;
- **sounds**: sequences of signals (dimension = frequency, vector of amplitudes);
- **text**: dimension = vocabulary size, vector of word counts, e.g. “late late storm” → (late: 2, storm: 1, ok: 0);
- **metadata**: authors, dates, ...

→ Real data are complex, redundant and variable.

Will it work on new data?



Trained on the left period, applied on the right one: the model has never seen such a peak.

Covariate shift: the distribution of $\mathbf{x}$ changes. **Concept shift:** the relation $\mathbf{x} \mapsto y$ changes.

Three kinds of learning



Supervised

examples come with the
right answer (the label)



Unsupervised

no answers: find
structure

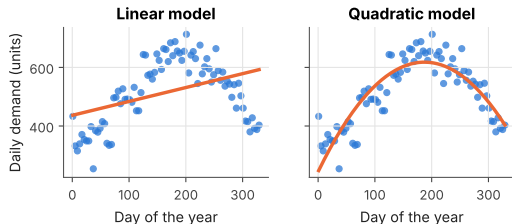


Reinforcement

act, receive a reward,
adapt

- **Supervised:** classification (SVM, k -NN, ...), regression, similarity learning (metric / transfer learning).
- **Unsupervised:** clustering (k -means, hierarchical), density estimation, new representations.

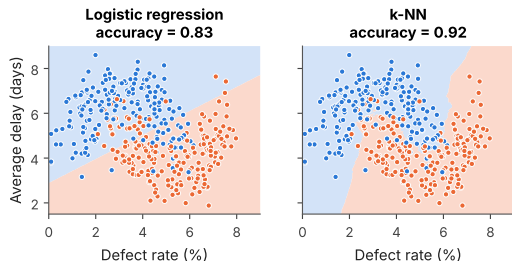
Supervised learning: regression



The system has descriptors and a **label provided by an oracle**. We learn a rule predicting a **real value**, for the data and for new data.

$\mathcal{Y} = \mathbb{R}$: lead time, demand, price.

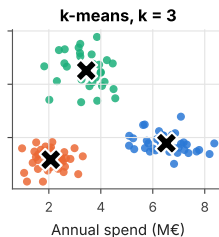
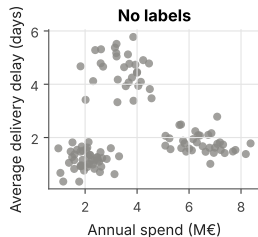
Supervised learning: classification



Same setting, but we predict a **category**. The classifier draws a **decision boundary** (linear on the left, non-linear on the right).

$\mathcal{Y} = \{-1, +1\}$ or $\{1, \dots, C\}$: late / on time, cat / dog.

Unsupervised learning: clustering

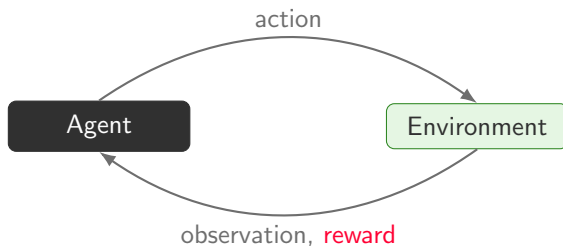


No labels during learning.

Group together examples showing **similarities**; also used for density estimation or a new representation.

Example: segment suppliers by size and reliability.

Reinforcement learning



The model must choose the right **actions** given the **context**, using **rewards**. No data say which action is optimal: it discovers by **trial and error**.

Example: a replenishment policy rewarded by low cost and few stock-outs.

Question: which kind of problem?

1. Predict the units sold next month.
2. Will this supplier default within a year?
3. Group customers with similar purchasing behaviour.
4. A robot learns to pick parcels faster, from a time reward.

Answer: which kind of problem?

1. Supervised, regression ($y \in \mathbb{R}$).
2. Supervised, binary classification ($y \in \{0, 1\}$).
3. Unsupervised, clustering (no label).
4. Reinforcement learning (actions and rewards).

Normalise the data I

Besides representation, we must look at the **values** taken by each descriptor, so as not to give undue weight to some of them: harmful for **distance-based** algorithms.

Definition 1.4: Euclidean distance

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2}$$

Example: $A = (400 \text{ km}, 2 \text{ d})$, $B = (410 \text{ km}, 9 \text{ d})$:

$$d(A, B) = \sqrt{10^2 + 7^2} \approx 12.2, \quad \text{almost only the kilometres.}$$

With $x_1 \in [0, 1]$ and $x_2 \in [500, 1000]$ the distance is essentially $|x_2 - x'_2|$.

Normalise the data II

Normalising increases performance in most cases by giving each variable the same weight.

■ **min-max scaling:** $x' = \frac{x - \min(x)}{\max(x) - \min(x)} \in [0, 1]$ (or

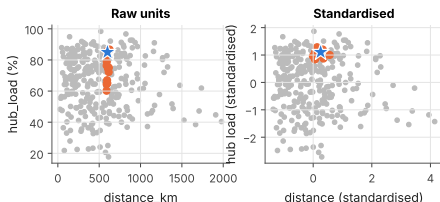
$1 - 2x' \in [-1, 1]$)

■ **standardisation:** $x' = \frac{x - \mu(x)}{\sigma(x)}$ (mean 0, std 1)

Example, lead times (2, 4, 6, 8): $\mu = 5$, $\sigma = \sqrt{5} \approx 2.24$.

raw	2	4	6	8
min-max	0	0.33	0.67	1
standardised	-1.34	-0.45	0.45	1.34

Normalise the data III



normalisation (unit norm): divide each *vector* by its norm so that $\|\mathbf{x}\| = 1$:

$$\mathbf{x}' = \frac{\mathbf{x}}{\|\mathbf{x}\|}.$$

The former act on *columns* (features), this one on *rows* (examples). Useful when only the direction matters (text counts).

Normalise the data IV: a classical trap

Definition 1.5: data leakage

Leakage happens when information from the validation or test data influences the training.

Wrong: standardise the whole dataset, then split.

Right: split first; compute μ, σ on the **training set only**; apply the same μ, σ to validation, test and production data.

$$x'_{\text{new}} = \frac{x_{\text{new}} - \mu_{\text{train}}}{\sigma_{\text{train}}}$$

→ Every preprocessing step is part of the model: learn it on the training set.

An important assumption

We never have access to the whole distribution, only to a **small sample**. To generalise to new data, we assume that they are *i.i.d.*

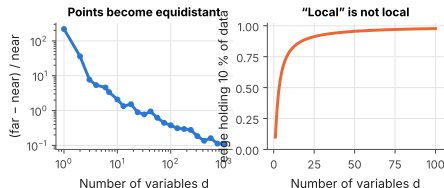
Definition 1.6: i.i.d. sample

A training set S is *i.i.d.* if all examples come from the same **unknown** distribution $\mathcal{D}$ and are mutually independent.

- **OK**: shipments drawn at random from one year of operations.
- **Not OK**: consecutive days of sales (dependent); 2019 training, 2021 use (different distribution).

→ A fundamental assumption in machine learning.

Data and dimension I



With enough examples in a space of *limited* dimension, we can approximate the distribution and generalise.

In high dimension: the **curse of dimensionality**.

Data and dimension II

A computation: in the unit cube $[0, 1]^d$, what share of the volume lies within 0.01 of the boundary?

$$1 - (1 - 2 \times 0.01)^d = 1 - 0.98^d$$

d	10	100	1000
share near the boundary	18 %	87 %	≈ 100 %

- In high dimension almost all data are **near the edge**; volumes behave unexpectedly, distances between points become similar.
- Consequences for distance-based algorithms; the number of examples needed grows **exponentially** with d .

→ Reduce the representation space: select, combine or compress the features.

Check your understanding: part 1

1. A retailer trains a demand model on 2019 data and deploys it in 2021, after the pandemic. Which assumption is violated, and what do you do?
2. You use k -NN with two features: distance (0–2000 km) and delay (0–10 days). Why standardise, and on which data do you compute μ and σ ?
3. (a) “Group SKUs with similar demand patterns”; (b) “predict whether an order will be cancelled”. Which type of learning?

Check your understanding: part 1, answers

1. The **i.i.d. / same-distribution** assumption (shift). Monitor performance, **re-train** on recent data.
2. Kilometres dominate the Euclidean distance. Standardise with μ, σ computed on the **training set only**, and reuse them for new data (no leakage).
3. (a) **unsupervised**, clustering; (b) **supervised**, binary classification.

Learning

Risk, loss, validation, regularisation

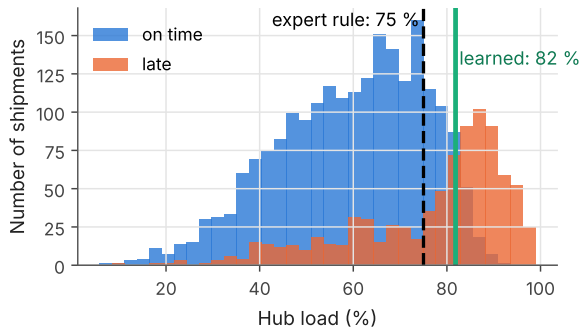
The setting I

Definition 2.1: data distribution

The data follow an **unknown** distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$: $\mathcal{X} \subset \mathbb{R}^d$ is the **input** (feature) space, $\mathcal{Y}$ the **output** (label) space.

$\mathcal{Y} = \emptyset$	unsupervised learning
$\mathcal{Y} = \mathbb{R}$	regression (lead time)
$\mathcal{Y} = \{0, 1\}$	binary classification (late / on time)
$\mathcal{Y} = \{1, \dots, C\}$	multi-class classification (cat, dog, bird)
$\mathcal{Y} = \{0, 1\}^C$	multi-label classification (several tags)

The setting II



- **Objective:** find an algorithm $\mathcal{A}$ that returns a hypothesis $h \in \mathcal{H}$ able to solve the task.
- **Problem:** in practice $\mathcal{D}$ is **unknown**. We only have a finite i.i.d. sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m \sim \mathcal{D}^m$.

The goal

We want $h(\mathbf{x}) = y$: powerful on the sample S , **but also on new instances** $(\mathbf{x}, y)$.

How do we measure “powerful”?

We need a criterion: a **performance measure** (to maximise) or, traditionally, a **risk** (to minimise).

On the whole world $\mathcal{D}$: the real risk,
never computable.

On our sample S : the empirical risk,
computable.

Real risk and empirical risk

Definition 2.2: 0–1 risks

Real risk (risk in generalisation) of a hypothesis h : the average error over the whole distribution,

$$\mathcal{R}(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} [\mathbb{1}_{\{h(\mathbf{x}) \neq y\}}].$$

Empirical risk: the average error on the training set,

$$\mathcal{R}_S(h) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{\{h(\mathbf{x}_i) \neq y_i\}}.$$

Analogy: $\mathcal{R}$ = opinion of the whole country, $\mathcal{R}_S$ = a poll of m people.

→ We minimise $\mathcal{R}_S$, but we care about $\mathcal{R}$.

Empirical risk: a worked example

Five shipments, $m = 5$:

truth y_i	late	ok	late	ok	ok
prediction $h(\mathbf{x}_i)$	late	ok	ok	ok	late
error $\mathbb{1}_{\{h(\mathbf{x}_i) \neq y_i\}}$	0	0	1	0	1

$$\mathcal{R}_S(h) = \frac{0 + 0 + 1 + 0 + 1}{5} = 0.4$$

- $\mathcal{R}_S(h)$ is the **frequency of errors** on S , an *estimate* of $\mathbb{P}[h(\mathbf{x}) \neq y]$.
- Accuracy = $1 - \mathcal{R}_S(h) = 60\%$.

What is the best we can hope for?

Even a perfect learner may err: the labels contain **noise**. The best possible hypothesis is the **Bayes** one:

- classification (0–1 loss): $h^*(\mathbf{x}) = \arg \max_c \mathbb{P}(y = c \mid \mathbf{x})$;
- regression (squared loss): $h^*(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}]$.

Definition 2.3: Bayes risk

$\mathcal{R}^* = \mathcal{R}(h^*)$ is the **irreducible error**: no algorithm, however much data it has, can do better.

Two shipments with identical features, one late and one on time: any rule is wrong on at least one of them.

Learning in two steps: ERM

We cannot compute $\mathcal{R}$, so we apply **empirical risk minimisation** (ERM):

$$\hat{h} = \arg \min_{h \in \mathcal{H}} \mathcal{R}_S(h), \quad h_{\mathcal{H}} = \arg \min_{h \in \mathcal{H}} \mathcal{R}(h).$$

Split the excess risk $\mathcal{R}(\hat{h}) - \mathcal{R}^*$:

$$\underbrace{\mathcal{R}(\hat{h}) - \mathcal{R}(h_{\mathcal{H}})}_{\text{estimation error}} + \underbrace{\mathcal{R}(h_{\mathcal{H}}) - \mathcal{R}^*}_{\text{approximation error}}$$

- **approximation**: is $\mathcal{H}$ rich enough? (too small $\Rightarrow$ underfitting);
- **estimation**: is S large enough to find the best element of $\mathcal{H}$? (too rich $\Rightarrow$ overfitting).

One can show $\mathcal{R}(\hat{h}) - \mathcal{R}(h_{\mathcal{H}}) \leq 2 \sup_{h \in \mathcal{H}} |\mathcal{R}(h) - \mathcal{R}_S(h)|$: this is the quantity bounded in the last part.

Risk without labels

In **unsupervised** learning the notion of risk is also present, but no longer refers to an error: it is the quantity to minimise.

Definition 2.4: example: k -means

$$\min_{C_1, \dots, C_K} \sum_{k=1}^K \sum_{\mathbf{x} \in C_k} \|\mathbf{x} - \boldsymbol{\mu}_k\|^2$$

total squared distance of the points to the centre $\boldsymbol{\mu}_k$ of their cluster (**within-cluster variance**).

Other examples: a distance between distributions in unsupervised domain adaptation.

A computational problem

Minimising the number of errors is a **hard** problem (NP-hard in general): we rarely try to minimise the error rate as such.

- The 0–1 loss is a **staircase**: flat, then a jump. Its slope gives no direction in which to improve.
- We replace it by a **surrogate**: smooth, convex and above the 0–1 loss, so that the minimisation is efficient.

In practice we minimise a risk based on a **loss function**.

Loss functions I

Definition 2.5: loss function

A **loss** $\ell : \mathcal{H} \times \mathcal{Z} \rightarrow \mathbb{R}^+$ measures the disagreement between the prediction $h(\mathbf{x})$ and the value to be predicted y , where $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$.

regression	squared	$\ell = (y - h(\mathbf{x}))^2$
	absolute	$\ell = y - h(\mathbf{x}) $
classification	0-1	$\ell = \mathbb{1}[h(\mathbf{x}) \neq y]$

Example: errors (1, 2, 3, 20) days: $\text{MAE} = \frac{26}{4} = 6.5$, $\text{RMSE} = \sqrt{103.5} \approx 10.2$.
One outlier weighs much more under the squared loss.

Loss functions II: the 0–1 loss

Written with the margin $z = y h(\mathbf{x})$ for $y \in \{-1, +1\}$:

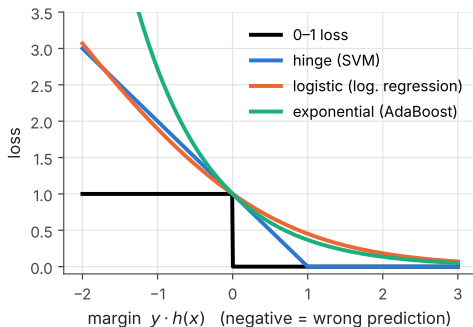
$$\ell_{0-1}(z) = \begin{cases} 1 & \text{if } z < 0 \text{ (wrong sign),} \\ 0 & \text{otherwise.} \end{cases}$$

- **not convex, not differentiable**: of little interest for optimisation algorithms;
- it is **very rarely used** in practice.

→ use **surrogates**: substitution functions that are **convex** and sometimes even **differentiable**.

They are presented as **upper bounds** of the 0–1 loss: $\ell_{0-1}(z) \leq \ell(z)$ for all z .

Loss functions III: the picture



Each surrogate is **large** when $z < 0$ (wrong side) and **small** when $z \gg 0$ (confidently right).

Different surrogates are used for different algorithms.

Loss functions IV: the three classical surrogates

- **hinge loss** (SVM): $\ell(h(\mathbf{x}), y) = \max(0, 1 - y h(\mathbf{x}))$
- **logistic loss** (logistic regression): $\ell(h(\mathbf{x}), y) = \frac{1}{\ln 2} \ln(1 + e^{-y h(\mathbf{x})})$
- **exponential loss** (boosting): $\ell(h(\mathbf{x}), y) = e^{-y h(\mathbf{x})}$

Check: at $z = 0$ the hinge loss is 1, the logistic loss $\frac{\ln 2}{\ln 2} = 1$, the exponential 1: all are $\geq 1 = \ell_{0-1}$ for $z \leq 0$, and ≥ 0 elsewhere. The $1/\ln 2$ is there precisely to make it an upper bound.

→ Same idea, three algorithms: SVM, logistic regression, boosting.

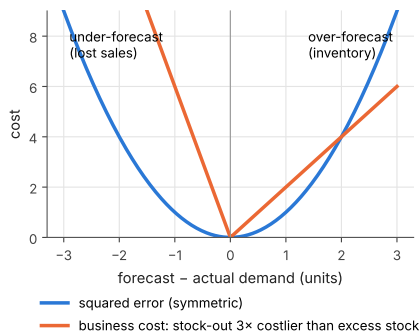
A new version of the risk

Definition 2.6: risks for a loss ℓ

$$\mathcal{R}(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} [\ell(h(\mathbf{x}), y)], \quad \mathcal{R}_S(h) = \frac{1}{m} \sum_{i=1}^m \ell(h(\mathbf{x}_i), y_i)$$

The **loss** is a modelling choice: it says what an error costs. The 0–1 risk is the special case $\ell = \ell_{0-1}$.

The loss should reflect business costs



Missing a defaulting supplier (false negative) costs far more than a false alarm:
use an **asymmetric** loss.

Costs and the decision threshold

With a probabilistic model $p(\mathbf{x}) = \mathbb{P}(\text{late} \mid \mathbf{x})$ and costs c_{FN} (miss) and c_{FP} (false alarm), the expected cost is lowest if we predict *late* exactly when

$$p(\mathbf{x}) > \tau = \frac{c_{FP}}{c_{FP} + c_{FN}}.$$

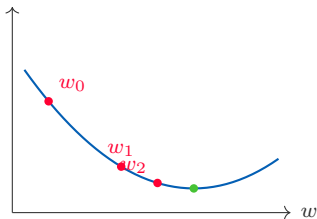
- $c_{FN} = c_{FP}$: $\tau = 0.5$ (the usual rule).
- $c_{FN} = 10 c_{FP}$: $\tau = \frac{1}{11} \approx 0.09$: we alert much more readily.

→ Train with a surrogate; decide with the business threshold.

How do we minimise the risk?

Most algorithms minimise $\mathcal{R}_S(\mathbf{w})$ by **gradient descent**: repeat a small step downhill,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \mathcal{R}_S(\mathbf{w}_t), \quad \eta > 0 \text{ the learning rate.}$$



For $\mathcal{R}_S(w) = (w - 3)^2$, $\eta = 0.25$:
 $w_0 = 0 \rightarrow w_1 = 1.5 \rightarrow w_2 = 2.25 \rightarrow 2.63 \dots \rightarrow 3$.

For a linear model with the squared loss: $\nabla \mathcal{R}_S(\mathbf{w}) = \frac{2}{m} X^\top (X\mathbf{w} - \mathbf{y})$. A **convex** loss guarantees that the descent reaches a global minimum.

What is a good model?

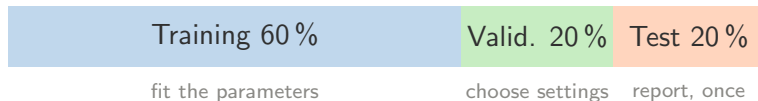
We have data S , a loss ℓ and an algorithm. How do we know whether the model performs well?

- we **minimise** the empirical risk $\mathcal{R}_S(h) = \frac{1}{m} \sum_i \ell(h(\mathbf{x}_i), y_i)$;
- we want to **estimate** the real risk $\mathcal{R}(h) = \mathbb{E}_{\mathcal{D}}[\ell(h(\mathbf{x}), y)]$.

But we do not know the distribution. . . so we use a **validation process**: estimate $\mathcal{R}$ on data the model has **never seen**.

→ $\mathcal{R}_S$ is optimistic: the model was built to make it small.

Validation I: hold-out

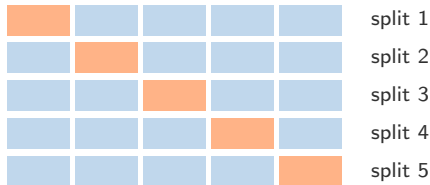


- Keep part of the data unused for learning the parameters: first evaluation (**training** / **validation**).
- The **test** set is used **once**, at the very end; otherwise it becomes part of the training.
- Time series: split **chronologically**, never at random.

Validation II: k -fold cross-validation

Definition 2.7: k -fold cross-validation

Split the data into k groups; learn on $k - 1$ of them and evaluate on the last; repeat k times, rotating the validation fold.



In the end the algorithm is evaluated on **all** the available data: a more reliable estimate.

Validation III: the CV error

Definition 2.8: mean error in cross-validation

With $\mathcal{R}_{S_j}$ the error on the validation fold S_j :

$$\hat{\mathcal{R}}_{CV} = \frac{1}{k} \sum_{j=1}^k \mathcal{R}_{S_j}(h_{-j})$$

Example, $k = 5$: errors (12, 9, 11, 10, 13) days $\Rightarrow \hat{\mathcal{R}}_{CV} = 11$ days, standard deviation 1.58, standard error $\frac{1.58}{\sqrt{5}} \approx 0.7$: **report** 11 ± 0.7 **days**.

- It quantifies how well the *procedure* generalises to new data from the same distribution.
- **Stratified** k -fold keeps the class proportions in each fold; $k = 5$ or 10 is the usual compromise.

Validation IV: leave-one-out

The same principle, but with **one** example for validation at each run: m trainings instead of k .

$$\hat{\mathcal{R}}_{LOO} = \frac{1}{m} \sum_{i=1}^m \ell(h_{-i}(\mathbf{x}_i), y_i)$$

- **Pros:** almost all the data are used for training; no randomness.
- **Cons:** m trainings, expensive for large m ; the m models are almost identical, so the estimate has *high variance*.

We keep the model with the best average result over the m validations.
Good for small datasets.

Test set discipline

- Hyper-parameters are chosen on **validation** / **CV**; the final score is measured on the **test set**, once.
- Every time we look at the test score and adjust the model, the test set leaks into training and the final estimate becomes **optimistic**.
- To tune *and* estimate without a separate test set: **nested cross-validation** (an inner CV to tune, an outer CV to evaluate).

→ The test set simulates the future: you only get to use the future once.

Classification metrics I: the confusion matrix

	predicted late	predicted on time
actually late	TP = 210	FN = 90
actually on time	FP = 70	TN = 630

$$\text{accuracy} = \frac{TP + TN}{m} = 0.84$$

$$\text{precision} = \frac{TP}{TP + FP} = 0.75, \quad \text{recall} = \frac{TP}{TP + FN} = 0.70$$

precision: among flagged shipments, how many really are late? **recall**: among late shipments, how many do we catch?

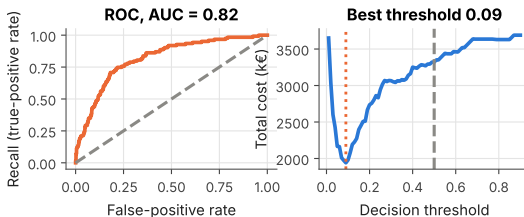
$$F_1 = \frac{2PR}{P+R} \approx 0.72 \text{ (harmonic mean).}$$

Classification metrics II: the accuracy trap

100 suppliers, 8 will default. A model answering “no default” for everyone gets

$$\text{accuracy} = 92\%, \quad \text{recall} = 0\%.$$

- With **imbalanced classes**, accuracy is misleading: compare with a **naive baseline**.
- Use **recall**, **precision**, F_1 , the **ROC curve** / AUC (all thresholds at once), or the **cost** of errors.



Question: train versus validation

The error is 5 days on the training set and 40 days in validation.

What happens, and what can you do?

Answer: the model has memorised

The gap training $\ll$ validation is the signature of **overfitting**: the model learned the training examples, including their noise.

- **simplify** the model (fewer parameters, shallower tree);
- **regularise** (next slides);
- get **more data**.

And if the validation error is *smaller* than the training error? Check for a small or unlucky validation set, or a leak.

Overfitting I

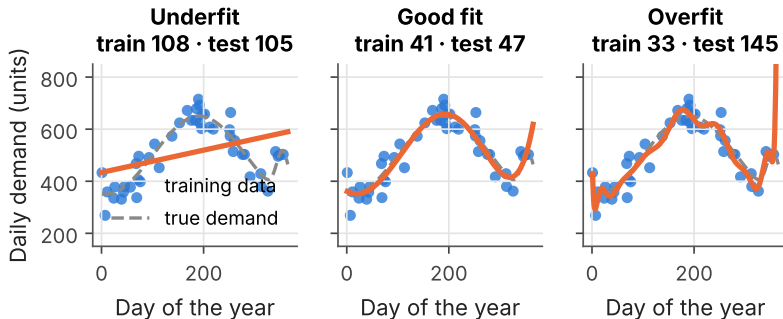
Student A understands the course: exam 80 % on a *new* exam

Student B learns past answers by heart:
100 % on past exams,
35 % on a new one

Definition 2.9: overfitting

In statistics, **overfitting** occurs when the model focuses so much on the data that it also learns the **noise**: the learned model is too complex, or the training set is too small relative to the number of parameters.

Overfitting II: the picture



Right panel: the curve passes through every point, **training error ≈ 0** , but is wrong between the points.

Underfitting

Definition 2.10: underfitting

Underfitting occurs when the model cannot capture the trend of the data, typically because it is far too simple, e.g. a linear model for a quadratic curve.

Example: a straight line for a seasonal demand with a year-end peak. **High error on training *and* validation.**

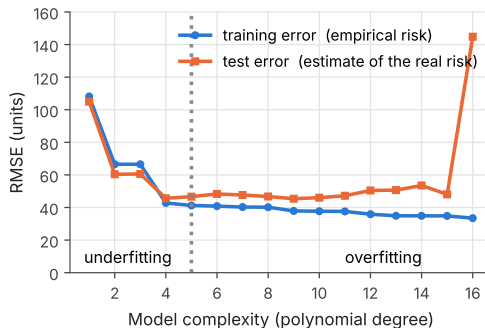
→ Overfit = too complex. Underfit = too simple.

How to know? Diagnosing a model

training error	CV error	diagnosis
low	much higher	overfitting
high	high (similar)	underfitting
low	low (close)	good fit

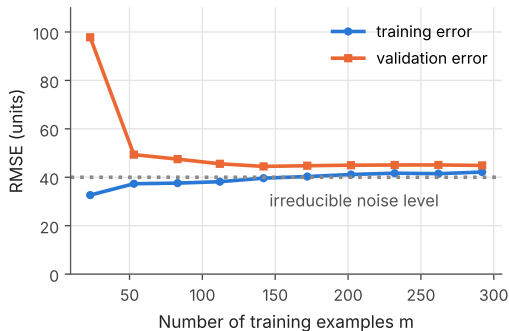
- **Overfitting:** the hypothesis space $\mathcal{H}$ is **too rich**: much too large, containing complex models.
- **Underfitting:** $\mathcal{H}$ is **poor**: too small, containing only simple models.

The complexity curve



As complexity grows, the training error **keeps falling** but the validation error **falls, then rises**: we want the minimum.

More data helps



With more examples the gap between training and validation closes: **more data fights overfitting.**

Bias and variance

For the squared loss, with $y = f(\mathbf{x}) + \varepsilon$, $\text{Var}(\varepsilon) = \sigma^2$, the expected error at a point splits as

$$\mathbb{E}[(y - \hat{h}(\mathbf{x}))^2] = \underbrace{\sigma^2}_{\text{noise}} + \underbrace{(\mathbb{E}[\hat{h}(\mathbf{x})] - f(\mathbf{x}))^2}_{\text{bias}^2} + \underbrace{\text{Var}(\hat{h}(\mathbf{x}))}_{\text{variance}}$$



low bias,
low var.



low bias,
high var.



high bias,
low var.



high bias,
high var.

Simple models: high bias, low variance (underfit). Complex models: low bias, high variance (overfit).

What to do?

To fight **underfitting**: change the model class (richer $\mathcal{H}$), add features.

To fight **overfitting**: *simplify* the learned model.

Definition 2.11: Occam's razor

The principle of **parsimony**: look for the simplest explanation (model) of the data.

"Entia non sunt multiplicanda praeter necessitatem" (William of Ockham)

In practice: add a **penalty** term to the optimisation problem.

Penalty

A penalty looks for a compromise between **performance** and **simplicity**:

$$h^* = \arg \min_{h \in \mathcal{H}} \mathcal{R}_S(h) + \text{pen}(m, d).$$

The penalty depends on the sample size m and the dimension d ; it is related to a nested sequence of spaces $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \dots \subset \mathcal{H}_n$ (**structural risk minimisation**).

This penalty is generally *hard to determine*. Instead, we use a very large set of hypotheses and **restrict the values of the parameters**.

Regularisation

Definition 2.12: regularisation

A **regularisation** is an additional term added to an optimisation problem to avoid over-learning. It **penalises models that are too complex**: it smooths the model or restricts the values of its parameters.

$$h^* = \arg \min_{h \in \mathcal{H}} \mathcal{R}_S(h) \quad \rightsquigarrow \quad h^* = \arg \min_{h \in \mathcal{H}} \underbrace{\mathcal{R}_S(h)}_{\text{fit}} + \lambda \underbrace{\|h\|}_{\text{complexity}}$$

Regularised empirical risk

- $\lambda \geq 0$ is the **regularisation parameter**, a **hyper-parameter** (as opposed to the *parameters* of h learned from data);
- $\|\cdot\|$ is a norm.

We return the hypothesis with the best compromise between performance and complexity.

$$\begin{array}{l|l} \lambda = 0 & \text{no penalty: risk of overfitting} \\ \lambda \text{ large} & \text{all parameters} \rightarrow 0: \text{underfitting (a constant model)} \end{array}$$

Already seen with linear models: $\min_{\beta \in \mathbb{R}^{d+1}} \|\mathbf{y} - X\beta\|^2 + \lambda \|\beta\|_1$ gives a **parsimonious** model, depending on few variables.

Usual norms

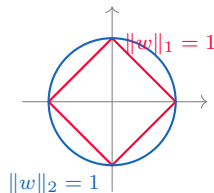
Definition 2.13: p -norm

$$\|\mathbf{w}\|_p = \left(\sum_{j=1}^d |w_j|^p \right)^{1/p}$$

- $p = 2$ (**ridge**): shrinks all weights.
- $p = 1$ (**lasso**): sets many weights *exactly* to 0.

Example: $\mathbf{w} = (3, -4)$: $\|\mathbf{w}\|_1 = 7$,
 $\|\mathbf{w}\|_2 = 5$.

The corners of the ℓ_1 ball lie on the axes: the optimum often falls there, hence **sparse** solutions.



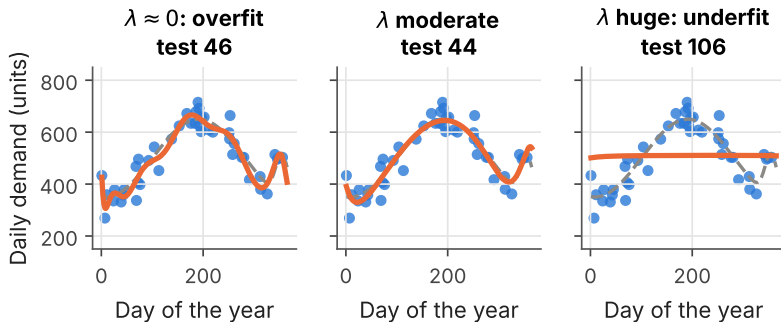
Ridge regression in one formula

Definition 2.14: ridge regression

$$\min_{\mathbf{w}} \|\mathbf{y} - X\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \implies \hat{\mathbf{w}} = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y}$$

- $\lambda = 0$: ordinary least squares; $\lambda \rightarrow \infty$: $\hat{\mathbf{w}} \rightarrow 0$.
- $X^\top X + \lambda I$ is **always invertible** for $\lambda > 0$: ridge also stabilises collinear features.
- The penalty treats all weights alike: **standardise the features first**.

Regularisation in pictures

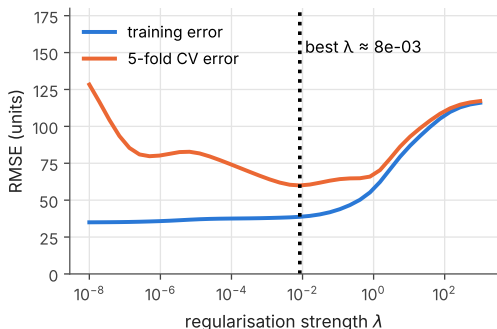


The same data fitted with increasing λ : from wiggly (overfit) to smooth, to flat (underfit).

Choice of the parameter λ

$$h^* = \arg \min_h \frac{1}{m} \sum_{i=1}^m \ell(h(\mathbf{x}_i), y_i) + \lambda \|h\|$$

λ controls the complexity of our hypothesis, and indirectly the **richness of $\mathcal{H}$** .



Choose λ at the **minimum** of the validation curve. But how?

Choose λ by validation

Our training set can also check the ability to generalise for a given hyper-parameter:

- **standard validation**: split into train / validation (e.g. $2/3 - 1/3$);
- **k -fold cross-validation**: $k - 1$ groups for learning, 1 for validation;
- **leave-one-out**: learn on all examples but one.

→ Same recipe for any hyper-parameter: λ , k of k -NN, tree depth, number of trees.

Grid search with k -fold CV

1. Define a grid $\lambda \in (\lambda_1, \dots, \lambda_p)$.
2. For each λ_i compute the mean CV error $\frac{1}{k} \sum_{j=1}^k \mathcal{R}_{S_j}(h, \lambda_i)$.
3. Keep the λ_i with the lowest mean CV error.

λ	0.01	0.1	1	10	100
$\widehat{\mathcal{R}}_{CV}$ (days)	14.2	10.1	9.4	11.8	18.5
	overfit		best		underfit

Then retrain on the *whole* training set with $\lambda = 1$ and evaluate once on the test set.

A few remarks

- To avoid over-learning we use one or more **regularisation terms**, combined with **k -fold CV** ($k = 10$ is common).
- **Attention**: these approaches are **empirical**. To do the job properly we should study **generalisation guarantees**:

$$\left| \mathbb{E}_{\mathcal{D}} \ell(h(\mathbf{x}), y) - \mathbb{E}_{\mathcal{S}} \ell(h(\mathbf{x}), y) \right| \leq \mathcal{O}(\sqrt{1/m}, \lambda^{-1}).$$

- Sometimes we must distinguish the **optimisation problem** (the surrogate we minimise) from the **performance criterion** (what we want to optimise).

Check your understanding: part 2

1. The training error is 2 % and the CV error 25 %. Diagnosis? Name two remedies.
2. Why can we not simply keep the hypothesis with the smallest empirical risk $\mathcal{R}_S$, and what do we use instead to estimate $\mathcal{R}$?
3. A missed late delivery costs 10 times a false alarm. At which probability threshold do you flag a shipment as late? And what happens to the weights of a ridge model when $\lambda \rightarrow \infty$?

Check your understanding: part 2, answers

1. **Overfitting** (training $\ll$ validation). Simplify the model, regularise (increase λ), or add data.
2. $\mathcal{R}_S$ is **optimistic**: the model was fitted to make it small, and $\mathcal{R}$ is not computable. We estimate $\mathcal{R}$ on **unseen data**: validation, CV, and a final test set.
3. Threshold $\tau = \frac{c_{FP}}{c_{FP} + c_{FN}} = \frac{1}{11} \approx 0.09$. When $\lambda \rightarrow \infty$ all weights $\rightarrow 0$: the model becomes constant, **underfitting**.

Fundamental algorithms

Regression, k -NN, trees, SVM

Four families, one map



Linear models

regression,
logistic
regression



Neighbours

k -NN: vote of
the closest
examples



Trees

a chain of
questions

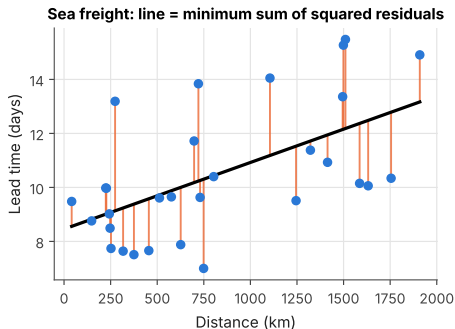


SVM

maximum
margin, kernels

For each one: **the model** h , **the loss / criterion** it minimises, its **hyper-parameters** and its **strengths and weaknesses**. They are the examples used in the practical sessions.

Linear regression I: the model



$$h(\mathbf{x}) = w_0 + w_1x_1 + \dots + w_dx_d = w_0$$

w_0 intercept, w_j : change of the prediction when x_j increases by one unit, the others being fixed.

Definition 2.15: least squares

Choose $\mathbf{w}$ to minimise the squared loss $\mathcal{R}_S(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m (y_i - w_0 - \mathbf{w}^\top \mathbf{x}_i)^2$.

Linear regression II: solution and example

The problem is convex and has a **closed form** (normal equations), with X augmented by a column of ones:

$$\hat{\mathbf{w}} = (X^\top X)^{-1} X^\top \mathbf{y}.$$

Worked example (one feature): points $(1, 2), (2, 3), (3, 5)$. Then $\bar{x} = 2$, $\bar{y} = \frac{10}{3}$,

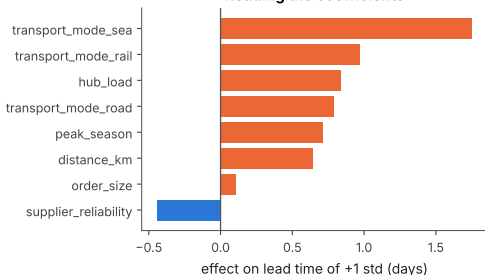
$$\hat{w}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{3}{2}, \quad \hat{w}_0 = \bar{y} - \hat{w}_1 \bar{x} = \frac{1}{3}$$

so the prediction at $x = 4$ is $\frac{1}{3} + \frac{3}{2} \times 4 = 6.33$.

→ Fast and readable. $X^\top X$ must be invertible: collinear features $\Rightarrow$ use ridge.

Linear regression III: reading and limits

Reading the coefficients



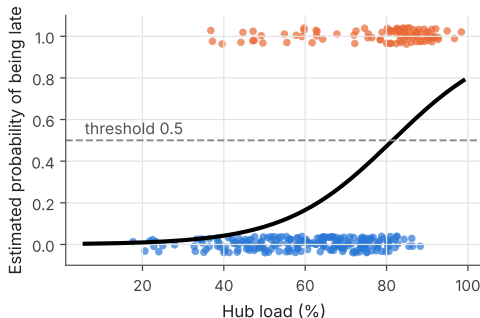
Standardised coefficients: effect on the lead time of **one standard deviation** of each feature (sea vs air: +2 days; hub load: +0.8 day per std).

- **Strengths:** interpretable, fast, solid baseline.
- **Weaknesses:** only **straight lines** (add polynomial or interaction features), sensitive to outliers (squared loss) and to collinearity.

Logistic regression I: from score to probability

For classification, a score $s = w_0 + \mathbf{w}^\top \mathbf{x} \in \mathbb{R}$ is turned into a probability by the **sigmoid**:

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \sigma(s) = \frac{1}{1 + e^{-s}}, \quad \ln \frac{p}{1 - p} = w_0 + \mathbf{w}^\top \mathbf{x} \quad (\text{log-odds}).$$



$s = 0 \Rightarrow p = 50\%$. We predict *late* if $p > \tau$ (threshold from the costs). The boundary $\mathbf{w}^\top \mathbf{x} + w_0 = 0$ is **linear**.

Logistic regression II: training and interpretation

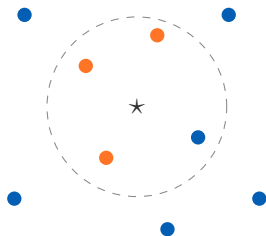
We minimise the **logistic loss** (cross-entropy, i.e. the negative log-likelihood), usually with a ridge or lasso penalty:

$$\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^m \ln(1 + e^{-y_i f(\mathbf{x}_i)}) + \lambda \|\mathbf{w}\|, \quad \nabla = \frac{1}{m} X^\top (\mathbf{p} - \mathbf{y}) \quad (y \in \{0, 1\}).$$

- No closed form: **gradient descent** (convex problem, so a global optimum).
- e^{w_j} is an **odds ratio**: $w_j = 0.7 \Rightarrow \text{odds} \times 2$ when x_j increases by one unit.

→ Linear regression for numbers, logistic regression for probabilities of a class.

k -nearest neighbours I: the rule



$k = 5$: 3 orange, 2 blue $\Rightarrow$ the new point $\star$ is **orange** (*late*).

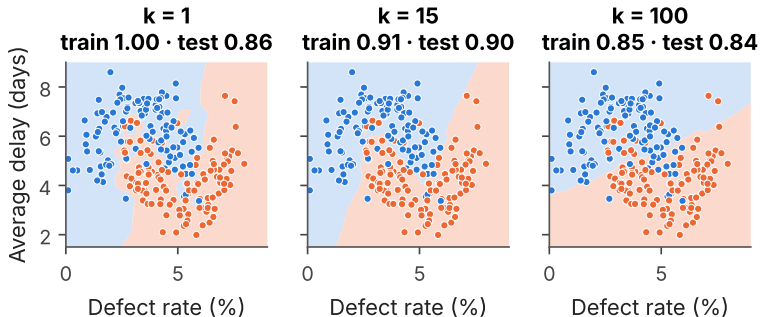
Definition 2.16: k -NN

$N_k(\mathbf{x})$ = the k training points closest to $\mathbf{x}$ (e.g. Euclidean distance).

$$\hat{y}(\mathbf{x}) = \frac{1}{k} \sum_{i \in N_k(\mathbf{x})} y_i \quad (\text{regression: average})$$

$$\hat{y}(\mathbf{x}) = \text{majority class in } N_k(\mathbf{x}) \quad (\text{classification: vote})$$

k -nearest neighbours II: choosing k



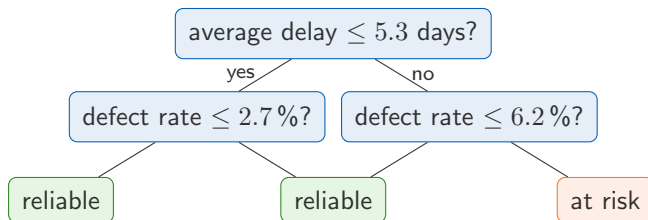
- $k = 1$: boundary follows every point, **training error = 0** (overfitting).
 $k = m$: always the majority class (underfitting).
- k is a **hyper-parameter**: choose it by cross-validation.

k -nearest neighbours III: practical remarks

- **Standardise** the features first: distances are meaningless otherwise (cf. normalisation).
- **No training step**: the data *are* the model; prediction costs $\mathcal{O}(md)$ per new point, so it is slow on large data.
- **Curse of dimensionality**: in high dimension all points are almost equally far away; reduce d first.
- Variants: **weighted vote** (weight $1/\text{distance}$), other distances (Manhattan, cosine), **metric learning**.

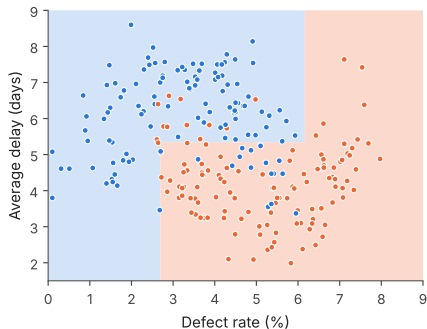
→ Simple, non-parametric, flexible; sensitive to scale and to dimension.

Decision trees I: a chain of questions



Each **internal node** asks a question on one feature; each **leaf** gives a prediction (majority class, or the mean for regression). The tree cuts the feature space into **rectangles**.

Decision trees II: the picture



Each question draws **one cut parallel to an axis**. The blue and orange regions are the leaves of the tree.

Decision trees III: which question?

Definition 2.17: Gini impurity

For a node with class proportions $p_1, \dots, p_K$: $G = 1 - \sum_{k=1}^K p_k^2$.
($G = 0$ pure; $G = 0.5$ for 50/50.)

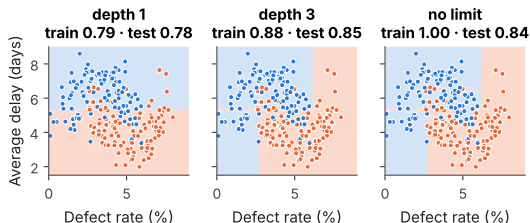
Among all questions " $x_j \leq t$?" keep the one that most reduces the weighted impurity of the two children:

$$\Delta G = G_{\text{parent}} - \left(\frac{m_L}{m} G_L + \frac{m_R}{m} G_R \right).$$

Example: 240 suppliers, $G = 0.50$; split into 137 ($G = 0.37$) and 103 ($G = 0.28$): $\Delta G = 0.50 - \left(\frac{137}{240} 0.37 + \frac{103}{240} 0.28 \right) \approx 0.17$.

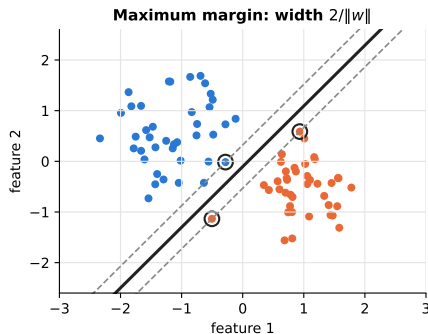
Alternative criterion: entropy $-\sum p_k \ln p_k$; regression trees use the variance.

Decision trees IV: overfitting and control



- Grown freely, a tree ends with pure leaves: **training accuracy 1.00**, lower test accuracy.
- **Hyper-parameters:** maximum depth, minimum leaf size, pruning.
- **Strengths:** readable rules, no scaling. **Weakness:** unstable, which motivates **ensembles**.

Support vector machines I: maximum margin



Among all separating lines, choose the one with the **largest margin** $2/\|\mathbf{w}\|$. The circled points, the **support vectors**, alone define the boundary.

Definition 2.18: hard-margin SVM ($y_i \in \{-1, +1\}$)

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad \forall i.$$

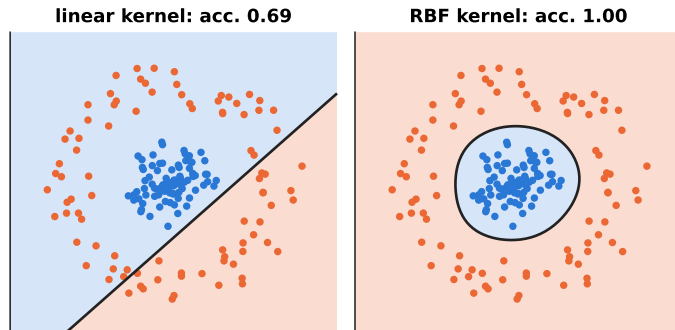
Support vector machines II: soft margin = hinge loss

Real data are not separable: allow margin violations, at a price.

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

- The second term is the **hinge loss** (surrogate of the 0–1 loss); the first is an ℓ_2 **regulariser**: **an SVM is regularised empirical risk minimisation**, with $C \approx 1/\lambda$.
- C large: few violations tolerated, narrow margin (risk of overfitting).
 C small: wide margin, more violations (risk of underfitting).

Support vector machines III: the kernel trick



A kernel $k(\mathbf{x}, \mathbf{x}')$ computes an inner product in a (huge) feature space without building it. Classical choice, the **RBF** kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2).$$

Hyper-parameters C and γ chosen by cross-validation (grid search); standardise the features first.

Which algorithm first?

algorithm	strengths	watch out
linear / logistic	fast, readable, baseline	only linear boundaries
k -NN	simple, flexible	scaling, slow, dimension
tree	readable rules, no scaling	unstable, overfits
SVM (kernel)	accurate, few hyper-param.	scales poorly with m , opaque

→ Start with a simple baseline, validate by cross-validation, then try richer models. Combining many trees (forests, boosting) gives the best tabular models.

Check your understanding: algorithms

1. Fit a least-squares line to $(1, 2)$, $(2, 3)$, $(3, 5)$ and predict at $x = 4$.
2. With k -NN, which of $k = 1$ and $k = 100$ overfits? Why must the features be standardised?
3. A node contains 80 % on-time and 20 % late shipments: compute its Gini impurity. In an SVM, what does increasing C do?

Check your understanding: algorithms, answers

1. $\hat{w}_1 = \frac{3}{2}$, $\hat{w}_0 = \frac{1}{3}$: prediction 6.33 at $x = 4$.
2. $k = 1$ overfits (training error 0, boundary follows each point); $k = 100$ is smoother. Standardise because distances are dominated by large-scale features.
3. $G = 1 - 0.8^2 - 0.2^2 = 0.32$ (a pure node has $G = 0$). A larger C penalises margin violations more: narrower margin, less regularisation, more risk of overfitting.

Generalisation bounds

Can we guarantee that it will work?

Why bounds?

Cross-validation is **empirical**: it gives an estimate, not a guarantee.

Definition 2.19: the question

Can we bound the gap between what we **observe** ($\mathcal{R}_S$) and what we **want** ($\mathcal{R}$)?

These studies are old; they stem from **PAC** learning (*Probably Approximately Correct*, Valiant, 1984).

$$\mathbb{P}(|\mathcal{R}(h) - \mathcal{R}_S(h)| \geq \varepsilon) \leq \delta, \quad \varepsilon > 0, \delta \in [0, 1].$$

Form of a bound

Bounds are usually written

$$|\mathcal{R}(h) - \mathcal{R}_S(h)| \leq \varepsilon(\delta, m) \quad \text{with probability at least } 1 - \delta,$$

where $1 - \delta$ is a **confidence level**. The function ε **decreases** with the number of examples m and **increases** when δ decreases (more confidence). Typical rate: $\mathcal{O}(1/\sqrt{m})$ (sometimes $\mathcal{O}(\ln m/\sqrt{m})$).

Example: $\varepsilon = 0.03$, $\delta = 0.05$: “95 % sure that the true error is within 3 points of the observed one”.

One fixed hypothesis: Hoeffding

Theorem 2.1: Hoeffding's inequality

Let $Z_1, \dots, Z_m$ be i.i.d. in $[0, 1]$ with mean μ . Then for all $\varepsilon > 0$,

$$\mathbb{P}\left(\left|\frac{1}{m} \sum_{i=1}^m Z_i - \mu\right| \geq \varepsilon\right) \leq 2e^{-2m\varepsilon^2}.$$

Apply it to $Z_i = \mathbb{1}\{h(\mathbf{x}_i) \neq y_i\}$ for a **fixed** h :

$$\mathbb{P}(|\mathcal{R}_S(h) - \mathcal{R}(h)| \geq \varepsilon) \leq 2e^{-2m\varepsilon^2}.$$

Example: $m = 1000$, $\varepsilon = 0.05$: $2e^{-5} \approx 0.013$.

→ Law of large numbers with an explicit rate: the gap shrinks like $1/\sqrt{m}$.

Uniform deviation

A bound for one fixed h is not enough: the algorithm picks $\hat{h}$ **after** seeing S , so it may pick the h for which $\mathcal{R}_S$ is luckily small.

We need bounds that are **uniform**, valid for all $h \in \mathcal{H}$ simultaneously:

$$\left\{ \sup_{h \in \mathcal{H}} |\mathcal{R}^\ell(h) - \mathcal{R}_S^\ell(h)| \geq \varepsilon \right\}.$$

They rely on convergence of $\mathcal{R}_S$ to $\mathcal{R}$ (strong law of large numbers) and **concentration inequalities**; some depend on the **complexity** of $\mathcal{H}$.

Finite hypothesis space

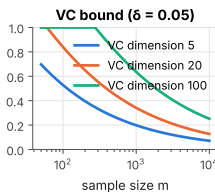
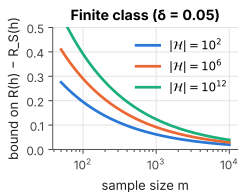
Theorem 2.2: uniform generalisation bound

Let $\mathcal{H}$ be finite, S an i.i.d. sample of size m , $\ell \in [0, 1]$, $\delta > 0$. Then with probability at least $1 - \delta$, **for all** $h \in \mathcal{H}$:

$$\mathcal{R}^\ell(h) \leq \mathcal{R}_S^\ell(h) + \sqrt{\frac{\ln |\mathcal{H}| + \ln(2/\delta)}{2m}}.$$

Proof idea: union bound over the $|\mathcal{H}|$ hypotheses,
 $\mathbb{P}(\exists h : \text{gap} \geq \varepsilon) \leq 2|\mathcal{H}|e^{-2m\varepsilon^2}$; set this equal to δ and solve for ε .

Reading the bound



$|\mathcal{H}| = 10^6$, $\delta = 0.05$, $m = 10\,000$:

$$\varepsilon = \sqrt{\frac{13.8 + 3.7}{20\,000}} \approx 0.03.$$

Conversely, for $\varepsilon = 0.05$:

$$m \geq \frac{\ln |\mathcal{H}| + \ln(2/\delta)}{2\varepsilon^2} \approx 3\,500.$$

→ Richer $\mathcal{H}$ ($\ln |\mathcal{H}| \uparrow$) $\Rightarrow$ looser bound; more data $\Rightarrow$ tighter, in $1/\sqrt{m}$.

Infinite hypothesis spaces

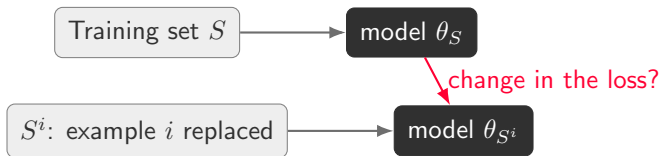
$\mathcal{H}$ is rarely finite: think of all lines in the plane, or all lines separating two classes of points.

- We measure the complexity of $\mathcal{H}$ differently: the **VC dimension** (Vapnik–Chervonenkis, 1971) or the **Rademacher complexity**.
- d_{VC} = the largest number of points that $\mathcal{H}$ can **shatter** (realise all 2^n labellings). Lines in $\mathbb{R}^2$: $d_{VC} = 3$.
- Bound (Vapnik): $\mathcal{R}(h) \leq \mathcal{R}_S(h) + \sqrt{\frac{d_{VC} \left(\ln \frac{2m}{d_{VC}} + 1 \right) + \ln \frac{4}{\delta}}{m}}$.

These notions are rather difficult to handle: we now follow **another approach**.

Uniform stability I: the idea

A more recent approach: it does **not** evaluate the complexity of $\mathcal{H}$ but the **hyper-parameter** controlling it (λ). Disadvantage: **only for convex problems**.



An algorithm is **stable** if changing **one** example barely changes its predictions.

Uniform stability II: definition

Definition 2.20: uniform stability

An algorithm $\mathcal{A}$ has **uniform stability** β/m with respect to the loss ℓ if

$$\forall S, \forall i \in \{1, \dots, m\}, \quad \sup_{\mathbf{x}} |\ell(\theta_S, \mathbf{x}) - \ell(\theta_{S^i}, \mathbf{x})| \leq \frac{\beta}{m},$$

where θ_S are the parameters learned on S and S^i is S with the i -th example replaced by another one drawn from $\mathcal{D}$.

The smaller β , the more stable the algorithm.

Uniform stability III: the bound

Theorem 2.3: bound via uniform stability

Let $\delta > 0$, $m > 1$; an algorithm with uniform stability β/m for a loss bounded by K . With probability at least $1 - \delta$ over the draw of S :

$$\mathcal{R}^\ell(\theta_S) \leq \mathcal{R}_S^\ell(\theta_S) + \frac{2\beta}{m} + (4\beta + K) \sqrt{\frac{\ln 1/\delta}{2m}}.$$

→ Stable algorithm + enough data $\Rightarrow$ the empirical risk is a reliable guide.

Stability and regularisation

For a regularised problem $\min \mathcal{R}_S(h) + \lambda \|h\|^2$ with a Lipschitz convex loss, one shows $\beta \propto \frac{1}{\lambda}$ (Bousquet & Elisseeff, 2002).

- **larger** $\lambda \Rightarrow$ more stable $\Rightarrow$ smaller gap $\mathcal{R} - \mathcal{R}_S$;
- but also a **larger** $\mathcal{R}_S$ (underfitting): the bias–variance trade-off again.

This is the theoretical justification of choosing λ by cross-validation: [the bound trades the two terms](#).

Check your understanding: part 3

1. $|\mathcal{H}| = 1000$ hypotheses, $m = 5000$ examples, $\delta = 0.05$. Compute the gap ε guaranteed by the finite-class bound.
2. By which factor does ε change if we *quadruple* m ? And if $\mathcal{H}$ grows from 10^3 to 10^6 hypotheses (qualitatively)?
3. What does uniform stability measure, and how does λ influence it?

Check your understanding: part 3, answers

1. $\varepsilon = \sqrt{\frac{\ln 1000 + \ln 40}{2 \times 5000}} = \sqrt{\frac{6.91 + 3.69}{10\,000}} \approx 0.033.$
2. $\varepsilon \propto 1/\sqrt{m}$: quadrupling m **halves** ε . A larger $\mathcal{H}$ increases $\ln |\mathcal{H}|$ ($6.9 \rightarrow 13.8$): the bound **loosens**, but only logarithmically.
3. How much the **loss changes when one training example is replaced** (constant β/m). Larger λ gives smaller β (more stable), at the price of a larger empirical risk.

Take-home messages



Learn from data

what counts is
performance on **new**
data



Align

loss and threshold
follow the business cost



Validate

normalise on train, test
once, regularise, choose
 λ by CV

Further reading

- James, Witten, Hastie, Tibshirani. *An Introduction to Statistical Learning*.
- Hastie, Tibshirani, Friedman. *The Elements of Statistical Learning*.
- Shalev-Shwartz, Ben-David. *Understanding Machine Learning*.
- Valiant (1984); Vapnik, Chervonenkis (1971); Koltchinskii, Panchenko (2002).
- Bousquet, Elisseeff. *Stability and Generalization*, JMLR 2002.