Machine Learning

A Complete Course — Chapter 1: Introduction to Machine Learning

Pr. ABDELLAOUI ▪ www.abdellaoui.net
ENS-Meknes — Université Moulay Ismaïl ▪ 2026

Chapter outline

What is Machine Learning?

Types of Learning

The Machine Learning Pipeline

Formal Setting: Risk and Empirical Risk Minimization

Generalization, Overfitting and Underfitting

Data: Splitting, Leakage and Quality

Feature Engineering and Preprocessing

What is Machine Learning?

Section 1 — Chapter 1

Learning objectives of this chapter

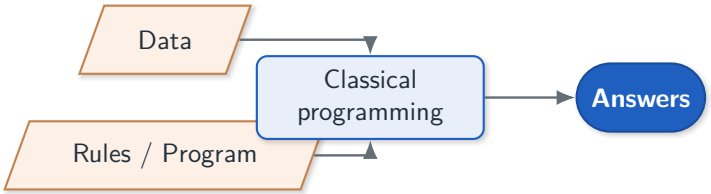
By the end of this chapter you will be able to:

- Define machine learning formally and intuitively
- Distinguish supervised, unsupervised, self-supervised and reinforcement learning
- Describe every stage of the ML pipeline
- Write the empirical risk minimization principle
- Explain overfitting, underfitting and the curse of dimensionality
- Split, clean and scale a dataset correctly

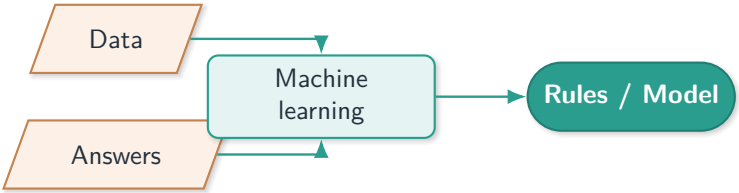
Prerequisites

Linear algebra (vectors, matrices), basic probability, calculus (derivatives), Python basics. Chapter 2 revisits all of them in depth.

Programming vs. learning



Human writes explicit rules (e.g. `if income` $>$ `50k`)



Algorithm *infers* the rules from examples

Tom Mitchell's definition (1997)

Definition

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Tom Mitchell's definition (1997)

Definition

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Example: spam filtering

- T: classify an e-mail as *spam* or *ham*
- P: fraction of e-mails correctly classified
- E: a corpus of e-mails labelled by users

Tom Mitchell's definition (1997)

Definition

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Example: spam filtering

- T: classify an e-mail as *spam* or *ham*
- P: fraction of e-mails correctly classified
- E: a corpus of e-mails labelled by users

The three ingredients are always present

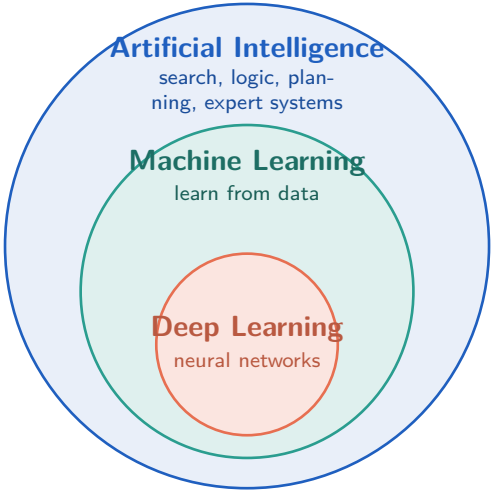
Whenever you read an ML problem, identify (T, P, E) first — it clarifies the data you need and the metric you optimize.

(T, P, E) for common applications

Application	Task T	Performance P	Experience E
Handwriting recognition	label digit images	accuracy	labelled images (MNIST)
Credit scoring	predict default	AUC, cost	past loans with outcomes
House pricing	predict price	RMSE	historical sales
Recommendation	rank items	click-through rate	user–item interactions
Autonomous driving	steer / brake	km without intervention	driving logs, simulation
Machine translation	FR $\rightarrow$ EN	BLEU score	parallel corpora
Game playing (Go)	choose a move	win rate	self-play games

Watch out

Choosing P badly is a classic failure: 99% accuracy on a dataset with 99% negatives is worthless (Chapter 5).

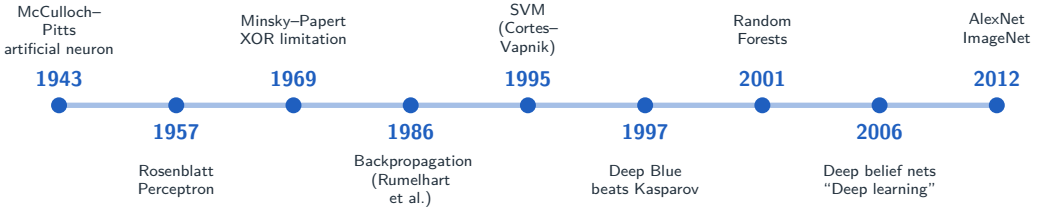


AI: any technique enabling machines to mimic intelligent behaviour.

ML: subset where behaviour is *learned* from data rather than hand-coded.

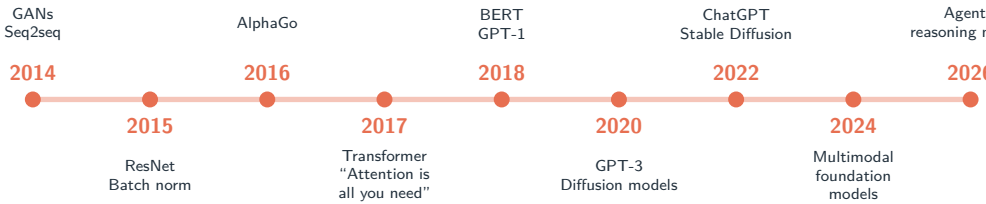
DL: subset of ML using multi-layer neural networks (Chapters 9–10).

A short history (1/2)



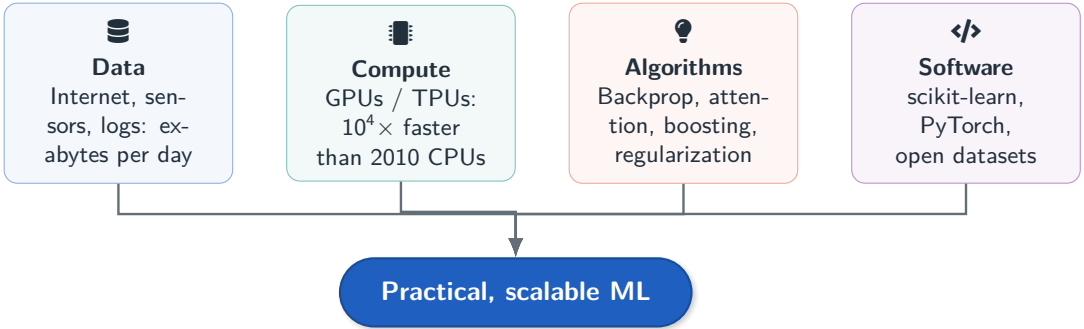
- 1950s–60s: *connectionism* and symbolic AI compete; perceptron enthusiasm then “AI winter”.
- 1980s–90s: backprop, statistical learning theory, kernel methods, boosting.

A short history (2/2)



- Three enablers of the deep learning era: big data, GPUs, algorithmic advances.
- Classical ML (this course's core) remains the workhorse for tabular data and small/medium datasets.

Why is ML so successful now?



When should you use ML?

Good candidates

- Rules are too complex to write by hand (vision, speech)
- The environment changes and the system must adapt (fraud, spam)
- Large amounts of relevant data exist
- Approximate answers are acceptable

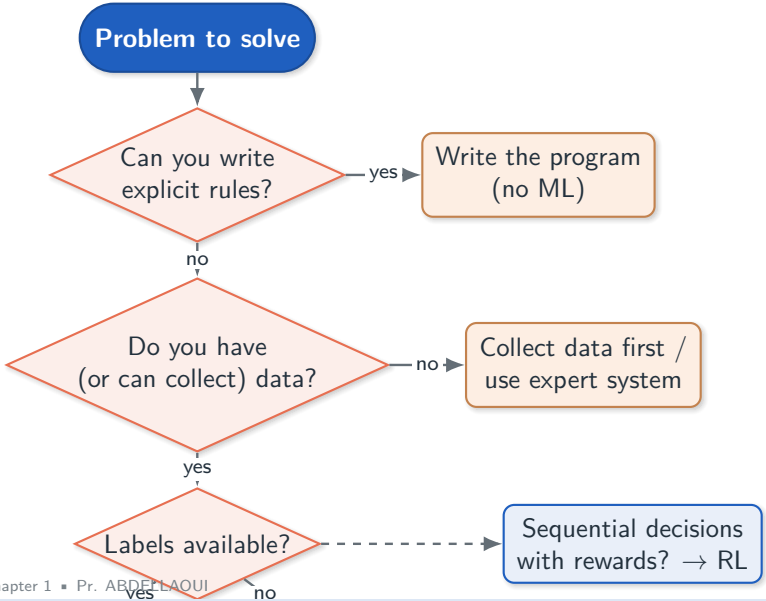
Poor candidates

- A simple deterministic rule exists (tax computation)
- No data, or data unrelated to the target
- Zero tolerance for error without human oversight
- Explanations legally required and model must be trivially auditable

Rule of thumb

Start with the simplest baseline (a heuristic, a linear model). Only add complexity when the baseline demonstrably fails.

Decision flowchart: is ML the right tool?



Vocabulary you must master

Term	Meaning
Instance / sample $\mathbf{x}_i$	one observation, described by d features: $\mathbf{x}_i \in \mathbb{R}^d$
Feature / attribute x_{ij}	the j-th measured quantity of sample i (age, pixel, word count)
Label / target y_i	the quantity we want to predict (class, price, ...)
Dataset $\mathcal{D}$	a collection $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$; design matrix $X \in \mathbb{R}^{n \times d}$
Model / hypothesis h	a function $h : \mathcal{X} \to \mathcal{Y}$ chosen from a family $\mathcal{H}$
Parameters $\boldsymbol{\theta}$	numbers learned from data (weights, thresholds)
Hyperparameters	settings fixed *before* training (depth, learning rate, k)
Training	finding $\boldsymbol{\theta}$ that minimizes a loss on $\mathcal{D}_{\text{train}}$
Inference / prediction	applying $h_{\boldsymbol{\theta}}$ to new inputs
Generalization	performance on unseen data

Notation used throughout the course

- Scalars: x, y, λ (italic)
- Vectors: $\mathbf{x}, \mathbf{w}, \boldsymbol{\theta}$ (bold, column vectors)
- Matrices: X, W, Σ (capital)
- $\mathbf{x}_i$: i-th sample; x_{ij}: its j-th feature
- n: number of samples; d: number of features
- $\hat{y}$: prediction; y: true value

$$X = \begin{pmatrix} — \ \mathbf{x}_1^\top \ — \\ \vdots \\ — \ \mathbf{x}_n^\top \ — \end{pmatrix} \in \mathbb{R}^{n \times d}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

- $\mathbb{E}[\cdot]$ expectation, $\mathbb{P}(\cdot)$ probability
- $\|\mathbf{x}\|_2 = \sqrt{\sum_j x_j^2}$ Euclidean norm
- $\mathbb{1}[A] = 1$ if A true, else 0
- $\arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$: minimizer of J

A first example: predicting house prices

Surface (m^2)	Rooms	Age	Price (k€)
80	3	20	180
120	4	5	310
60	2	35	120
150	5	10	390
⋮	⋮	⋮	⋮

Here $d = 3$, $\mathbf{x}_1 = (80, 3, 20)^\top$, $y_1 = 180$.

A linear model hypothesizes

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

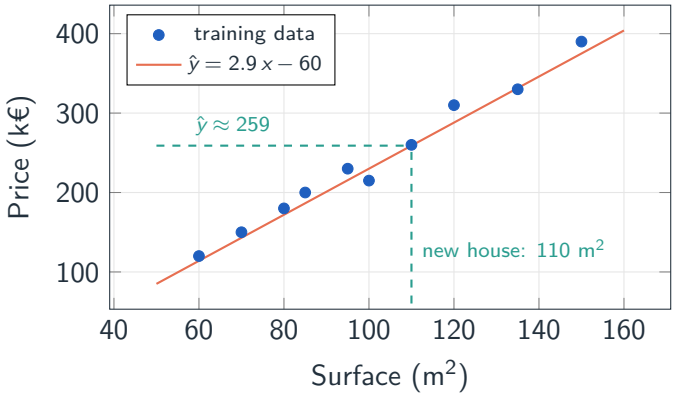
and training means choosing $\boldsymbol{\theta}$ to minimize

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \big(h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i \big)^2 .$$

Preview

Chapter 3 shows this has a closed-form solution $\hat{\boldsymbol{\theta}} = (X^\top X)^{-1} X^\top \mathbf{y}$ and can also be solved by gradient descent.

A first example: visualizing the fit



- The line is the *model*; its slope and intercept are the *parameters*.
- Prediction = plugging a new **x** into the fitted function.

Section recap

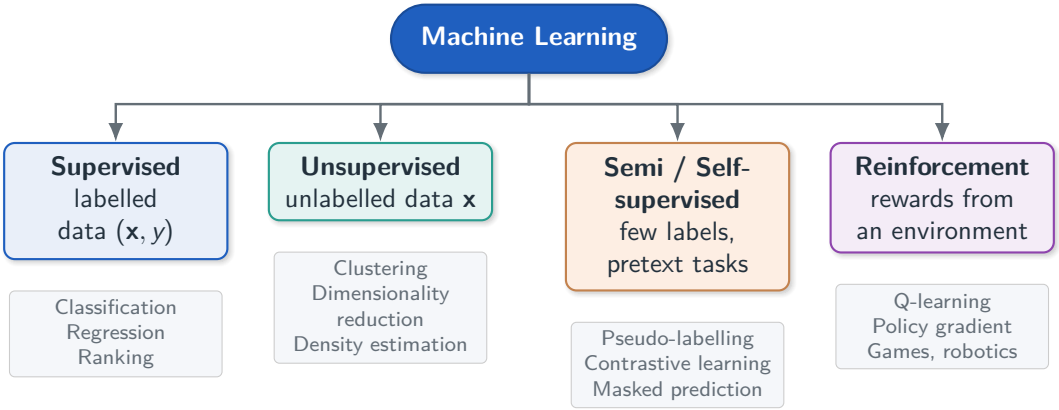
Key ideas

- ML = learning a function from data instead of programming it explicitly.
- Every ML problem is characterized by (T, P, E).
- AI $\supset$ ML $\supset$ DL; deep learning is one family of ML models.
- Use ML when rules are hard to write, data is abundant and errors are tolerable.
- Notation: $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, design matrix $X \in \mathbb{R}^{n \times d}$, model $h_{\boldsymbol{\theta}}$.

Types of Learning

Section 2 — Chapter 1

The learning paradigms at a glance



Supervised learning: definition

Supervised learning

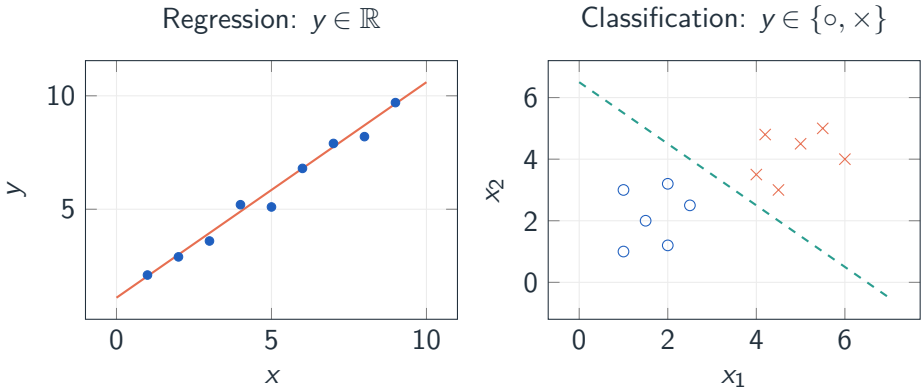
Given a training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ drawn i.i.d. from an unknown joint distribution $P(\mathbf{x}, y)$, learn a function $h : \mathcal{X} \to \mathcal{Y}$ such that $h(\mathbf{x}) \approx y$ for new pairs $(\mathbf{x}, y) \sim P$.

- The label y acts as a *teacher* (the "supervision").
- Two main flavours depending on the output space $\mathcal{Y}$:
  - $\mathcal{Y} = \mathbb{R}$ (or $\mathbb{R}^m$): regression
  - $\mathcal{Y} = \{1, \dots, K\}$ finite: classification ($K = 2$: binary; $K > 2$: multiclass)
- Also: multi-label ($\mathcal{Y} = \{0, 1\}^K$), ordinal regression, structured prediction (sequences, trees).

i.i.d. assumption

Samples are *independent and identically distributed*. Violations (time series, grouped data) require special care in evaluation (Chapter 5).

Regression vs. classification



- Regression: fit a curve; loss usually squared error $(y - \hat{y})^2$.
- Classification: find a *decision boundary*; loss usually 0–1 error $\mathbb{1}[y \neq \hat{y}]$ or cross-entropy.

Supervised learning: worked example (classification)

Iris dataset (Fisher, 1936): $n = 150$, $d = 4$, $K = 3$.

SL	SW	PL	PW	species
5.1	3.5	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.3	3.3	6.0	2.5	virginica

Encode labels: setosa $\to 0$, versicolor $\to 1$, virginica $\to 2$.

A simple hypothesis (a decision stump):

$$h(\mathbf{x}) = \begin{cases} 0 & \text{if PL} < 2.5 \\ 1 & \text{if } 2.5 \le \text{PL} < 4.9 \\ 2 & \text{otherwise} \end{cases}$$

Training error:

$$\hat{R}_n(h) = \frac{1}{150} \sum_{i=1}^{150} \mathbb{1}[h(\mathbf{x}_i) \neq y_i] \approx 0.07$$

Chapter 6 shows how decision trees learn such thresholds automatically.

Unsupervised learning: definition

Unsupervised learning

Given only inputs $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n$, discover *structure* in $P(\mathbf{x})$: groups, low-dimensional manifolds, densities, anomalies.

- **Clustering**: partition data into groups of similar points (k-means, DBSCAN, GMM).
- **Dimensionality reduction**: compress $\mathbf{x} \in \mathbb{R}^d$ to $\mathbf{z} \in \mathbb{R}^k$, $k \ll d$ (PCA, t-SNE, autoencoders).
- **Density estimation**: model $p(\mathbf{x})$ (KDE, mixture models).
- **Anomaly detection**: flag points with low $p(\mathbf{x})$.

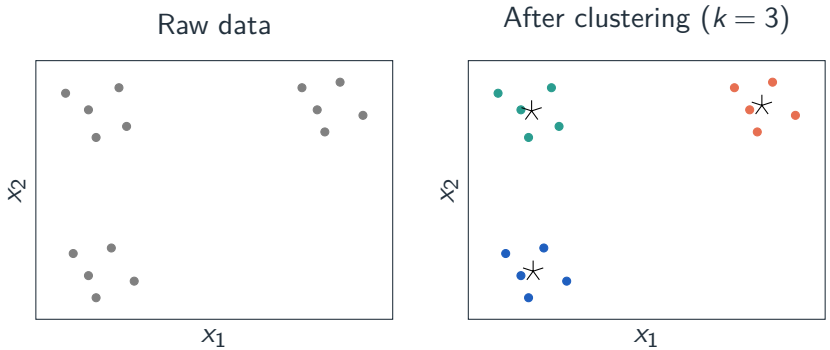
Harder to evaluate

No ground truth: quality is judged by internal criteria (silhouette, reconstruction error) or downstream usefulness.

Chapter 8

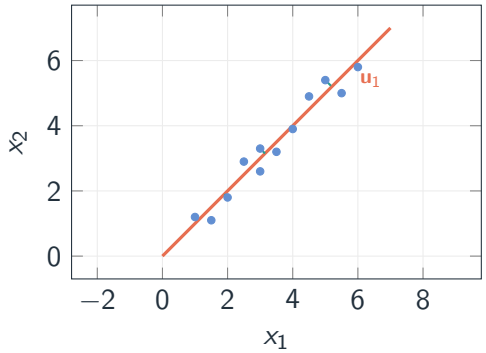
covers all unsupervised methods in depth.

Unsupervised learning: clustering illustrated



- Stars: cluster centroids $\boldsymbol{\mu}_k$. k-means minimizes $\sum_k \sum_{\mathbf{x} \in C_k} \|\mathbf{x} - \boldsymbol{\mu}_k\|^2$.
- The algorithm never saw a label — groups emerge from geometry alone.

Unsupervised learning: dimensionality reduction illustrated



PCA finds the direction $\mathbf{u}_1$ of maximal variance and projects each point:

$$z_i = \mathbf{u}_1^\top \mathbf{x}_i$$

Two numbers $\rightarrow$ one number, with little information lost.

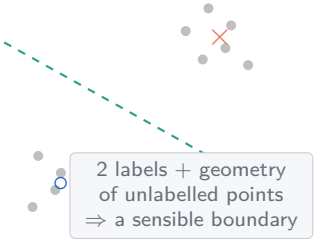
Setting

A small labelled set $\mathcal{D}_L = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_L}$ and a large unlabelled set $\mathcal{D}_U = \{\mathbf{x}_j\}_{j=1}^{n_U}$ with $n_U \gg n_L$.

Why it works — assumptions on $P(\mathbf{x})$:

- *Cluster assumption*: points in the same cluster share a label.
- *Manifold assumption*: data lie on a low-dimensional manifold; the boundary should cross low-density regions.

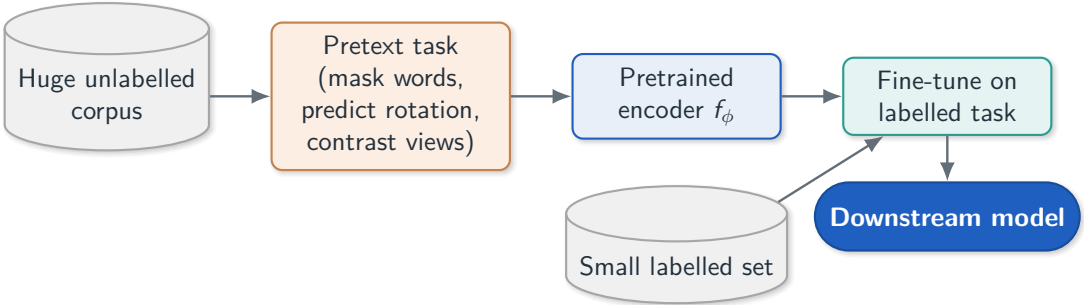
Techniques: self-training (pseudo-labels), co-training, label propagation, consistency regularization.



Self-supervised learning

Idea

Create labels *from the data itself* through a pretext task, learn a representation, then fine-tune on the real (small) task.



- Language: BERT masks 15% of tokens and predicts them; GPT predicts the next token.
- Vision: SimCLR / DINO pull two augmentations of the same image together in embedding space.

Reinforcement learning: the agent–environment loop



Objective

Find a policy π maximizing the expected discounted return

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \qquad \pi^* = \arg\max_{\pi} \mathbb{E}_\pi\big[G_t\big], \quad \gamma \in [0, 1).$$

- No labels: feedback is a *scalar reward*, often delayed and sparse.
- Trade-off between exploration (try new actions) and exploitation (use what works).

Reinforcement learning: key quantities

Markov Decision Process $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$

- State-value function

$$V^{\pi}(s) = \mathbb{E}_{\pi}\big[G_t \mid s_t = s\big]$$

- Action-value function

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\big[G_t \mid s_t = s, a_t = a\big]$$

- Bellman optimality equation

$$Q^{*}(s,a) = \mathbb{E}\big[r + \gamma \max_{a'} Q^{*}(s',a')\big]$$

Q-learning update (tabular):

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big]$$

Applications

Game playing (AlphaGo, Atari), robotics, recommender systems, RLHF for aligning language models, resource scheduling.

Scope

RL is beyond this course's core; we mention it for completeness.

Comparison of paradigms

	Supervised	Unsupervised	Self-supervised	Reinforcement
Input	$(\mathbf{x}, y)$	$\mathbf{x}$	$\mathbf{x}$ (+ pretext label)	(s, a, r, s')
Feedback	explicit label	none	derived from $\mathbf{x}$	scalar reward
Goal	predict y	find structure	learn representation	maximize return
Evaluation	accuracy, RMSE	silhouette, likelihood	downstream task	cumulative reward
Data cost	high (labelling)	low	low	simulation / interaction
Typical models	LR, SVM, trees, NN	k-means, PCA, GMM	BERT, SimCLR	DQN, PPO

Blurred boundaries

Modern systems combine them: pretrain self-supervised $\rightarrow$ fine-tune supervised $\rightarrow$ align with RL (RLHF).

Other learning settings you will meet

- **Online learning**: samples arrive one at a time; update θ after each.
- **Active learning**: the model *chooses* which samples to label.
- **Transfer learning**: reuse knowledge from a source task on a target task.
- **Multi-task learning**: one model, several outputs sharing representations.
- **Few-shot / zero-shot**: generalize from very few (or no) labelled examples.
- **Federated learning**: train across devices without centralizing data.
- **Continual learning**: learn new tasks without forgetting old ones.
- **Imitation learning**: learn a policy from expert demonstrations.

Batch (offline) learning

The default in this course: the whole training set is available at once; the model is trained, frozen, deployed.

Instance-based vs. model-based learning

Instance-based (lazy)
Memorize the training set; predict by comparing the query to stored examples.
e.g. k-NN: $\hat{y} = \text{majority}\{y_i : \mathbf{x}_i \in N_k(\mathbf{x})\}$

Training: $O(1)$ Prediction: $O(nd)$
Needs the data at inference time

Model-based (eager)
Compress the training set into parameters $\boldsymbol{\theta}$; discard data; predict with $h_{\boldsymbol{\theta}}$.
e.g. linear model $\hat{y} = \boldsymbol{\theta}^\top \mathbf{x}$

Training: costly Prediction: $O(d)$
Compact, fast to deploy

Parametric vs. non-parametric models

Parametric

The number of parameters is *fixed* regardless of n.
Linear regression ($d+1$ params), logistic regression, Naive Bayes, neural nets of fixed size.

- Strong assumptions $\Rightarrow$ data-efficient, may underfit.

Non-parametric

Model complexity *grows with* n.
k-NN, decision trees, kernel SVM, Gaussian processes, kernel density estimation.

- Flexible $\Rightarrow$ need more data, may overfit.

Misnomer

"Non-parametric" does not mean "no parameters" — it means the parameters are not fixed in number a priori.

Discriminative vs. generative models

Discriminative: model $P(y \mid \mathbf{x})$ directly (or just the boundary).

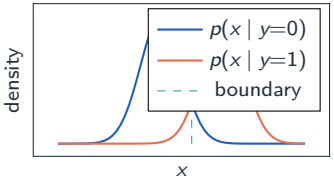
$$\hat{y} = \arg\max_y P(y \mid \mathbf{x}; \boldsymbol{\theta})$$

Logistic regression, SVM, trees, neural nets.

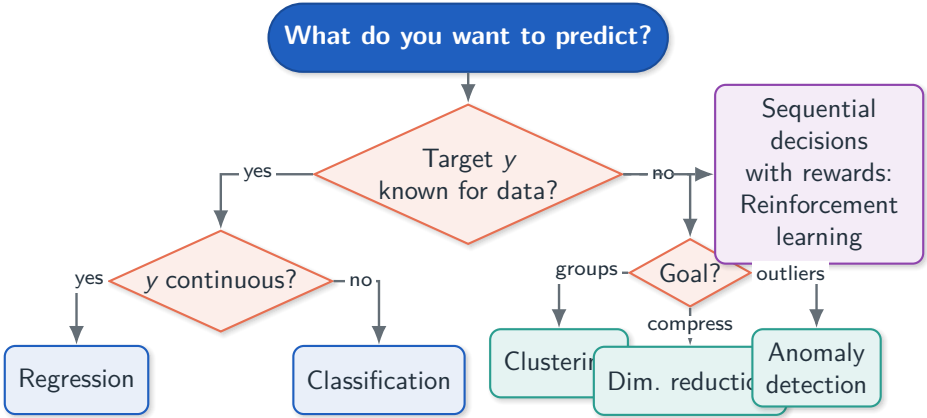
Generative: model the joint $P(\mathbf{x}, y) = P(\mathbf{x} \mid y)P(y)$, then use Bayes' rule:

$$P(y \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid y)P(y)}{\sum_{y'} P(\mathbf{x} \mid y')P(y')}$$

Naive Bayes, LDA, GMM, VAE, GAN, diffusion.



Choosing a paradigm: decision flowchart



Exercise: identify the paradigm

For each scenario, identify the paradigm (supervised / unsupervised / semi / self / RL) and the sub-task:

1. Predict tomorrow's electricity demand from the past 30 days.
2. Group customers by purchasing behaviour to design marketing campaigns.
3. Detect fraudulent credit card transactions given 10 000 labelled frauds among 10 million transactions.
4. Teach a drone to land using simulation feedback.
5. Compress 1000-dimensional gene expression profiles to 2-D for visualization.
6. Pretrain a text encoder by predicting masked words, then classify reviews with 500 labels.
7. Flag abnormal readings from a machine's vibration sensors (no failure history).

Exercise: identify the paradigm

For each scenario, identify the paradigm (supervised / unsupervised / semi / self / RL) and the sub-task:

1. Predict tomorrow's electricity demand from the past 30 days.
2. Group customers by purchasing behaviour to design marketing campaigns.
3. Detect fraudulent credit card transactions given 10 000 labelled frauds among 10 million transactions.
4. Teach a drone to land using simulation feedback.
5. Compress 1000-dimensional gene expression profiles to 2-D for visualization.
6. Pretrain a text encoder by predicting masked words, then classify reviews with 500 labels.
7. Flag abnormal readings from a machine's vibration sensors (no failure history).

Answers: (1) supervised regression / time series; (2) unsupervised clustering; (3) supervised binary classification, highly imbalanced; (4) reinforcement learning; (5) unsupervised dimensionality reduction; (6) self-supervised pretraining + supervised fine-tuning; (7) unsupervised anomaly detection.

Section recap

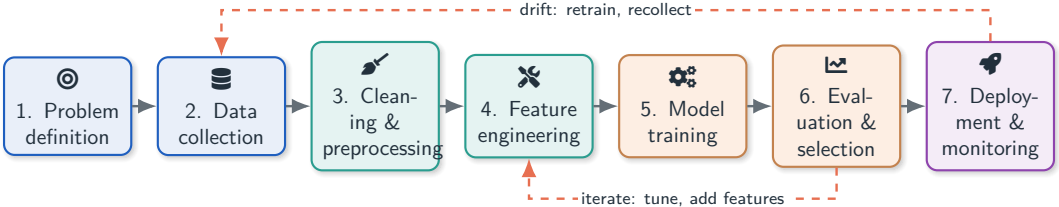
Key ideas

- **Supervised**: learn $\mathbf{x} \mapsto y$; regression ($y \in \mathbb{R}$) vs. classification (y discrete).
- **Unsupervised**: learn the structure of $P(\mathbf{x})$; clustering, dimensionality reduction, density, anomalies.
- **Semi/self-supervised**: exploit unlabelled data through assumptions or pretext tasks.
- **Reinforcement**: learn a policy from rewards; $Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a')]$.
- Orthogonal axes: instance vs. model-based, parametric vs. non-parametric, discriminative vs. generative.

The Machine Learning Pipeline

Section 3 — Chapter 1

The end-to-end ML pipeline



Reality check: practitioners spend 60–80% of their time on steps 2–4. Modelling (step 5) is often the smallest part.

Step 1 — Problem definition

Questions to answer *before* touching data:

- What decision will the prediction support? Who consumes it?
- What is the target y? Is it observable? When?
- What is the unit of analysis (a customer, a transaction, a day)?
- Which metric reflects business value (Chapter 5)?
- What baseline exists today (rule, human)?
- Constraints: latency, interpretability, fairness, budget.

Example: churn prediction

- Unit: one subscriber per month
- $y = 1$ if they cancel within 30 days
- Metric: recall at 10% contact budget
- Baseline: "call everyone who complained"
- Constraint: explain the top 3 reasons to the agent

Framing the target correctly

The target defines the problem

A different definition of y is a different ML problem, with different data, models and metrics.

Question	Target y	Task
"Will the customer churn?"	$y \in \{0, 1\}$	binary classification
"When will the customer churn?"	$y \in \mathbb{R}_+$ (months)	regression / survival analysis
"How likely is churn?"	$P(y = 1 \mid \mathbf{x}) \in [0, 1]$	probabilistic classification (calibrated)
"Which customers are most at risk?"	ordering	ranking
"Why do customers churn?"	segments	clustering + interpretation

Tip

Write the one-sentence problem statement: "Given inputs, predict target for unit in order to decision, measured by metric."

Sources

- Transactional databases, logs, sensors
- Public datasets (UCI, Kaggle, OpenML, HuggingFace)
- APIs, web scraping (check licences!)
- Surveys, annotation campaigns
- Simulation / synthetic data

Questions

- Is the sample representative of deployment conditions?
- Is n large enough? Rough rule: ≥ 10 samples per feature for linear models, far more for deep nets.
- How were labels obtained? Noise level? Inter-annotator agreement?
- Are there legal / privacy constraints (GDPR, consent)?

Selection bias

A credit model trained only on *accepted* applicants never sees rejected ones: the training distribution differs from the deployment distribution.

The data table: rows, columns, types

features **x** | target y

id	age	city	income	signup date	churn
1	34	Fès	4200	2023-01-12	0
2	51	Meknès	NA	2022-11-03	1
3	27	Rabat	3100	2023-05-30	0
4	45	Fès	6900	2021-07-19	1

numerical: age, income

categorical: city
datetime: signup date

missing: NA

- Each row = one sample $(\mathbf{x}_i, y_i)$; each column = one feature $x_{\cdot j}$. Design matrix $X \in \mathbb{R}^{n \times d}$ after encoding.

Feature types and what they imply

Type	Examples	Encoding for models	Typical issues
Numerical (continuous)	income, temperature	as is, scaled	outliers, skew
Numerical (discrete/count)	#purchases	as is, $\log(1+x)$	zero inflation
Binary	has_card	$\{0,1\}$	—
Categorical (nominal)	city, colour	one-hot, target encoding	high cardinality
Categorical (ordinal)	S < M < L	integer codes	respect order
Datetime	signup date	year, month, weekday, elapsed	leakage of future
Text	review	bag-of-words, TF-IDF, embeddings	vocabulary size
Image	photo	pixel tensor	need CNN (Ch. 10)

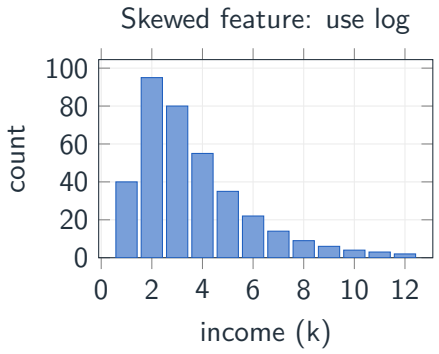
Section 7

details the preprocessing transforms for each type.

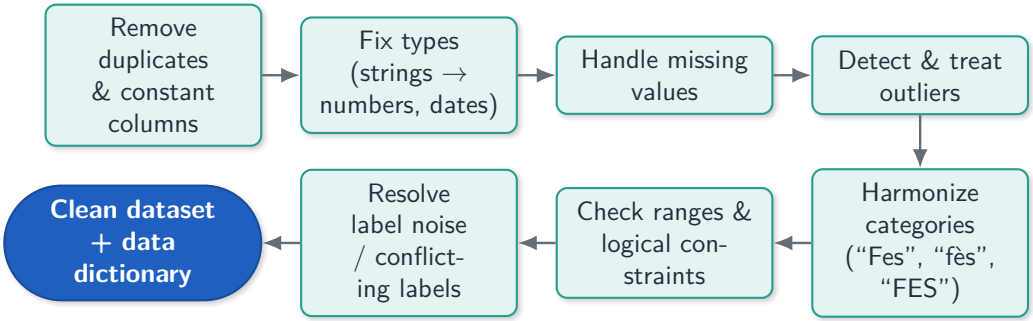
Step 3 — Exploratory Data Analysis (EDA)

Always look at your data before modelling:

- Shape: n, d, memory
- Per-feature summary: mean, median, std, min/max, % missing, # unique
- Distributions: histograms, box plots, log scale for skewed features
- Target distribution: class balance, range
- Relationships: correlation matrix, scatter plots, feature vs. target
- Duplicates, constant columns, obvious errors (age $= -1$)



Step 3 — Cleaning: a checklist



Golden rule

Every cleaning decision must be *scripted* and *reproducible*, never done by hand in a spreadsheet.

Feature engineering

Transforming raw variables into representations that make the underlying pattern easier for the model to learn.

Creating features

- Ratios: debt / income
- Aggregations: #purchases last 30 days
- Interactions: $x_1 x_2$, polynomial terms
- Date parts: weekday, hour, holiday flag
- Text: word counts, TF-IDF, embeddings
- Domain knowledge: BMI = weight / height^2

Transforming features

- Scaling: standardization, min-max
- Non-linear maps: log, $\sqrt{\cdot}$, Box–Cox
- Binning / discretization
- Encoding categoricals

Selecting features

- Filter (correlation), wrapper (RFE), embedded (Lasso, tree importance)

Feature engineering: a concrete example

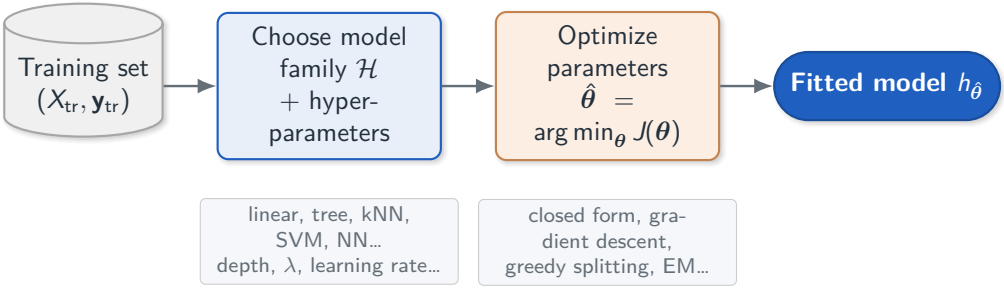
Raw log: one row per transaction (`user, timestamp, amount, merchant`)
Goal: predict fraud on the *current* transaction.

Engineered feature	Formula	Intuition
amount_z	$(a - \bar{a}_{\text{user}})/s_{\text{user}}$	unusual amount for this user
tx_last_1h	count of user's tx in previous hour	burst of activity
new_merchant	$\mathbb{1}$[merchant not seen before]	unfamiliar counterpart
hour_sin, hour_cos	$\sin(2\pi h/24)$, $\cos(2\pi h/24)$	cyclic time, 23h close to 0h
dist_home	haversine(location, home)	geographic anomaly
time_since_last	$t - t_{\text{prev}}$ (log-scaled)	velocity

Leakage alert

All aggregations must use only information available *before* the transaction's timestamp.

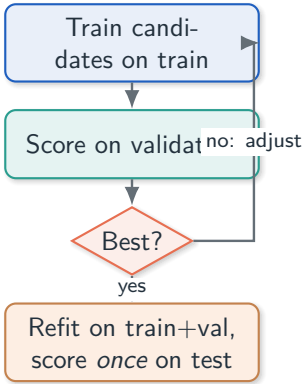
Step 5 — Model training



- Training = numerical optimization of an objective $J(\boldsymbol{\theta}) = \frac{1}{n}\sum_i \ell(y_i, h_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \lambda\,\Omega(\boldsymbol{\theta})$.
- Start simple (logistic regression / small tree), get a baseline number, then iterate.

Step 6 — Evaluation and model selection

- Never evaluate on training data: training error is optimistically biased.
- Hold out a validation set for choosing hyperparameters, a test set for the final estimate.
- Use cross-validation when data is scarce (Chapter 5).
- Compare against the baseline with confidence intervals.
- Inspect errors: which samples fail? why?



Serving patterns

- Batch scoring (nightly job writes predictions to a table)
- Online API (REST/gRPC endpoint, latency budget in ms)
- Embedded / edge (mobile, IoT: quantized models)
- Streaming (Kafka consumers)

Engineering concerns

- Package the *whole pipeline* (preprocessing + model) — training/serving skew
- Version data, code, model artifacts
- Reproducible environments (containers)
- Fallback to baseline on failure
- A/B test before full rollout

MLOps

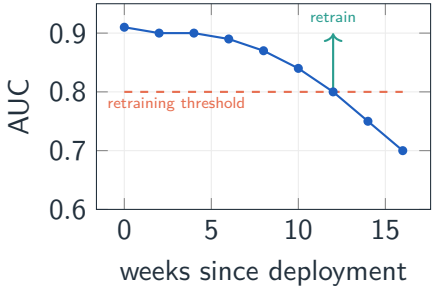
The discipline of reliably building, deploying and maintaining ML systems. Chapter 10 revisits it.

Data drift

$P_{\text{deploy}}(\mathbf{x}) \neq P_{\text{train}}(\mathbf{x})$: inputs change (new demographics, sensor recalibration).

Concept drift

$P_{\text{deploy}}(y \mid \mathbf{x}) \neq P_{\text{train}}(y \mid \mathbf{x})$: the relationship changes (new fraud tactics, pandemic).



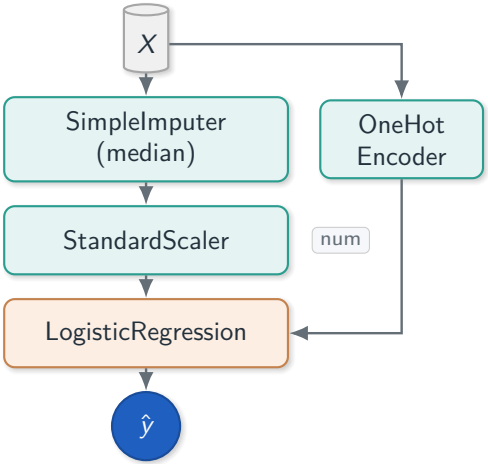
Monitor input statistics, prediction distribution, and (when labels arrive) the metric.

The pipeline as code: scikit-learn `Pipeline`

Python

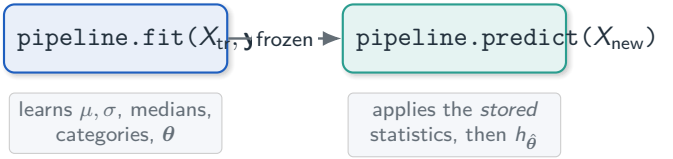
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import
StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import
LogisticRegression

num = Pipeline([("imp", SimpleImputer("median")),
("sc", StandardScaler())])
cat = OneHotEncoder(handle_unknown="ignore")
pre = ColumnTransformer([("num", num, num_cols),
("cat", cat, cat_cols)])
clf = Pipeline([("pre", pre),
("model", LogisticRegression())])
clf.fit(X_train, y_train)
clf.predict_proba(X_test)
```

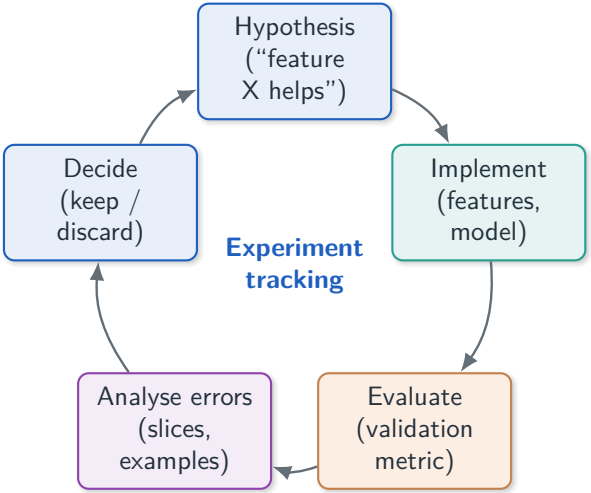


Why wrap everything in a pipeline?

- **No leakage**: the scaler's μ, σ and the imputer's median are computed on the training fold only, and re-used on validation/test.
- **Cross-validation friendly**: `cross_val_score(clf, X, y)` refits preprocessing in each fold.
- **Hyperparameter search** over preprocessing *and* model: `"model__C": [0.1, 1, 10]`.
- **Single artifact** to serialize and deploy: preprocessing cannot drift from the model.



Iterating: the ML development loop



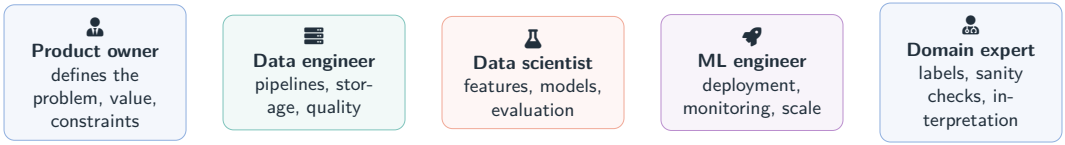
Log every run: data version, features, hyperparameters, seed, metric. Tools: MLflow, Weights & Biases, plain CSV.

Change *one thing at a time*; keep the validation set fixed so numbers are comparable.

Common pipeline failures

Symptom	Likely cause	Remedy
Validation 99%, production 70%	leakage, non-representative split	audit features, time-based split
Great offline, no business impact	wrong metric / target	revisit problem definition
Model degrades over months	drift	monitoring, scheduled retraining
Training error high	underfitting, bad features	richer model, feature engineering
Training $\ll$ validation error	overfitting	regularization, more data
Different results each run	unseeded randomness	fix seeds, log versions
Serving errors on new categories	unseen values	`handle_unknown`, fallback

Roles in an ML project



- In small teams one person wears several hats — this course trains the data-scientist role while giving the vocabulary to talk to the others.

Section recap

Key ideas

- Seven stages: problem $\rightarrow$ data $\rightarrow$ cleaning $\rightarrow$ features $\rightarrow$ training $\rightarrow$ evaluation $\rightarrow$ deployment; the loop iterates.
- The target definition and the metric are decided *before* modelling.
- Most of the effort is data work; leakage is the number-one silent failure.
- Encapsulate preprocessing + model in one `Pipeline` object.
- Deployment is not the end: monitor for data and concept drift.

Formal Setting: Risk and Empirical Risk Minimization

Section 4 — Chapter 1

Ingredients

- Input space $\mathcal{X}$ (e.g. $\mathbb{R}^d$), output space $\mathcal{Y}$ ($\mathbb{R}$ or $\{1, \dots, K\}$)
- An *unknown* joint distribution P on $\mathcal{X} \times \mathcal{Y}$
- A training sample $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \overset{\text{i.i.d.}}{\sim} P$
- A hypothesis class $\mathcal{H} \subseteq \{h : \mathcal{X} \to \mathcal{Y}\}$
- A loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ measuring the cost of predicting $\hat{y}$ when the truth is y

Goal

Use $\mathcal{D}$ to find $h \in \mathcal{H}$ that predicts well on *new* draws from P.

Regression

$$\text{squared: } \ell(y,\hat{y}) = (y-\hat{y})^2$$

$$\text{absolute: } \ell(y,\hat{y}) = |y-\hat{y}|$$

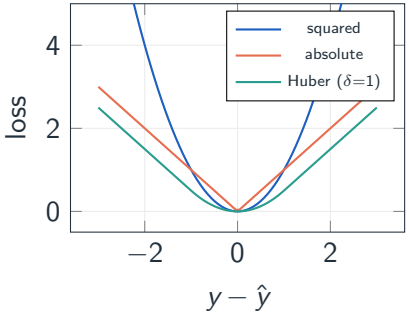
$$\text{Huber}_\delta : \begin{cases} \frac{1}{2}(y-\hat{y})^2 & |y-\hat{y}| \le \delta \\ \delta\,|y-\hat{y}| - \frac{\delta^2}{2} & \text{else} \end{cases}$$

Classification

$$\text{0–1: } \ell(y,\hat{y}) = \mathbb{1}[y \neq \hat{y}]$$

$$\text{log-loss: } \ell(y,\hat{p}) = -\log \hat{p}_y$$

$$\text{hinge: } \ell(y,f) = \max(0, 1-yf),\ y \in \{\pm 1\}$$



The loss encodes *what kind of error hurts*: squared loss punishes large errors heavily; absolute loss is robust to outliers.

True risk (expected loss)

Risk of a hypothesis

$$R(h) = \mathbb{E}_{(\mathbf{x},y)\sim P}\big[\ell(y, h(\mathbf{x}))\big] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(y, h(\mathbf{x}))\, dP(\mathbf{x}, y)$$

- $R(h)$ is the *average loss on future data* — exactly what we care about.
- It is not computable: P is unknown. We only have n samples.
- The **Bayes predictor** is the best possible function (not restricted to $\mathcal{H}$):

$$h^* = \underset{h \text{ measurable}}{\arg\min}\ R(h), \qquad R^* = R(h^*) \text{ (Bayes risk, irreducible)}$$

Bayes predictor for common losses

Squared loss: $h^*(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}]$. Absolute loss: $h^*(\mathbf{x}) = \text{median}(y \mid \mathbf{x})$. 0–1 loss: $h^*(\mathbf{x}) = \arg\max_k P(y = k \mid \mathbf{x})$.

Derivation: Bayes predictor under squared loss

Minimize $R(h) = \mathbb{E}[(y - h(\mathbf{x}))^2]$ pointwise. Fix $\mathbf{x}$ and let $c = h(\mathbf{x})$, $m = \mathbb{E}[y \mid \mathbf{x}]$:

$$\begin{aligned}\mathbb{E}\big[(y-c)^2 \mid \mathbf{x}\big] &= \mathbb{E}\big[(y-m+m-c)^2 \mid \mathbf{x}\big] \\ &= \underbrace{\mathbb{E}\big[(y-m)^2 \mid \mathbf{x}\big]}_{\mathrm{Var}(y|\mathbf{x})} + 2(m-c)\underbrace{\mathbb{E}[y-m \mid \mathbf{x}]}_{=0} + (m-c)^2 \\ &= \mathrm{Var}(y \mid \mathbf{x}) + (m-c)^2 \;\geq\; \mathrm{Var}(y \mid \mathbf{x}),\end{aligned}$$

with equality iff $c = m$. Hence

$$h^*(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}], \qquad R^* = \mathbb{E}_{\mathbf{x}}\big[\mathrm{Var}(y \mid \mathbf{x})\big] \text{ (noise floor).}$$

Interpretation

Regression = estimating the conditional mean. Even a perfect model has error $R^* > 0$ if y is noisy given $\mathbf{x}$.

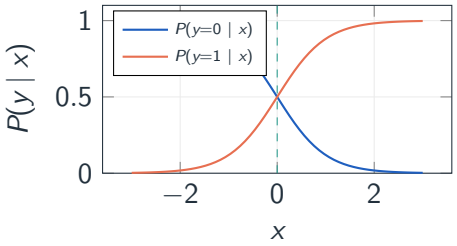
Derivation: Bayes classifier under 0–1 loss

For $\mathcal{Y} = \{1, \dots, K\}$ and $\ell = \mathbb{1}[y \neq \hat{y}]$, at a fixed $\mathbf{x}$ predicting class k costs

$$\mathbb{E}\big[\mathbb{1}[y \neq k] \mid \mathbf{x}\big] = \mathbb{P}(y \neq k \mid \mathbf{x}) = 1 - \mathbb{P}(y = k \mid \mathbf{x}).$$

Minimizing over k:

$$h^*(\mathbf{x}) = \arg\max_k \ \mathbb{P}(y = k \mid \mathbf{x}), \qquad R^* = \mathbb{E}_{\mathbf{x}}\big[1 - \max_k \mathbb{P}(y = k \mid \mathbf{x})\big].$$



Predict 1 when $P(y{=}1 \mid x) > \frac{1}{2}$, i.e. $x > 0$. Near the boundary both classes are plausible: this region contributes most of the Bayes error.

Empirical risk

Since P is unknown, replace the expectation by the sample average:

Empirical risk

$$\hat{R}_n(h) = \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, h(\mathbf{x}_i)\big)$$

- By the law of large numbers, for a *fixed* h: $\hat{R}_n(h) \xrightarrow[n\to\infty]{} R(h)$.
- $\hat{R}_n(h)$ is an *unbiased* estimate of $R(h)$ for fixed h: $\mathbb{E}_{\mathcal{D}}[\hat{R}_n(h)] = R(h)$.
- Called *training error* when computed on $\mathcal{D}_{\text{train}}$, *test error* on $\mathcal{D}_{\text{test}}$.

Subtlety

Once h is *chosen using* $\mathcal{D}$, $\hat{R}_n(h)$ is no longer unbiased — it is optimistic. This is the root of overfitting.

ERM principle (Vapnik)

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \ \hat{R}_n(h) = \arg\min_{h \in \mathcal{H}} \ \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, h(\mathbf{x}_i)\big)$$

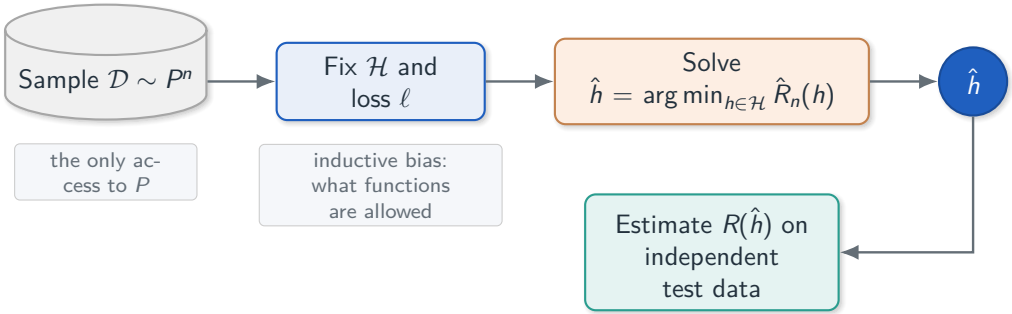
In parametric form, $\mathcal{H} = \{h_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta\}$ and ERM becomes an optimization over $\boldsymbol{\theta}$:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \ J(\boldsymbol{\theta}), \qquad J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, h_{\boldsymbol{\theta}}(\mathbf{x}_i)\big).$$

Instances of ERM

- Linear regression: $\mathcal{H} = \{\mathbf{x} \mapsto \boldsymbol{\theta}^\top \mathbf{x}\}$, squared loss $\Rightarrow$ least squares.
- Logistic regression: sigmoid model, log-loss $\Rightarrow$ maximum likelihood.
- Neural networks: $h_{\boldsymbol{\theta}}$ = composition of layers, any differentiable loss $\Rightarrow$ backprop + SGD.

ERM as a flowchart

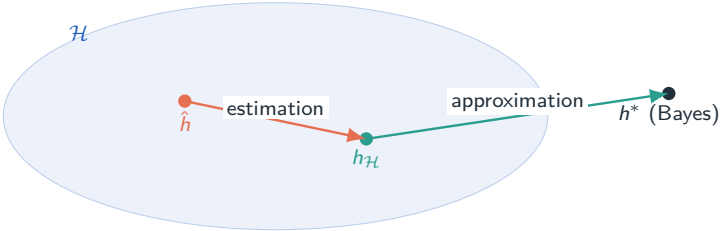


- Two sources of error: choosing $\mathcal{H}$ (approximation) and using finite n (estimation). Next slide.

Excess risk decomposition

Let h^* be the Bayes predictor, $h_{\mathcal{H}} = \arg\min_{h \in \mathcal{H}} R(h)$ the best in the class, $\hat{h}$ the ERM solution.

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



- Larger $\mathcal{H} \Rightarrow$ smaller approximation error but larger estimation error (harder to pick the right h from n samples).
- This trade-off is the theoretical face of *underfitting vs. overfitting*.

Generalization gap

$$R(\hat{h}) - \hat{R}_n(\hat{h})$$

the difference between true (test) risk and training risk of the learned model.

- For a fixed h, Hoeffding's inequality (bounded loss in $[0,1]$) gives

$$\mathbb{P}\Big(\Big|R(h) - \hat{R}_n(h)\Big| > \epsilon\Big) \leq 2e^{-2n\epsilon^2}.$$

- For the *selected* $\hat{h}$ we need a uniform bound over $\mathcal{H}$. For finite $\mathcal{H}$, with probability $\geq 1-\delta$:

$$R(\hat{h}) \leq \hat{R}_n(\hat{h}) + \sqrt{\frac{\log|\mathcal{H}| + \log(2/\delta)}{2n}}.$$

- Message: the gap shrinks with n and grows with the *capacity* of $\mathcal{H}$. Chapter 5 extends this to infinite classes (VC dimension).

Structural risk minimization / regularization

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \; \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, h_{\boldsymbol{\theta}}(\mathbf{x}_i)\big) + \lambda\, \Omega(\boldsymbol{\theta})$$

Ω penalizes complexity; $\lambda \geq 0$ controls the trade-off.

Common penalties:

- $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$ (ridge / weight decay)
- $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$ (lasso, sparsity)
- Tree depth, number of leaves
- Early stopping (implicit)

Why it helps

Restricting to "simple" $\boldsymbol{\theta}$ effectively shrinks $\mathcal{H}$, reducing estimation error at a small cost in approximation error.

Bayesian view

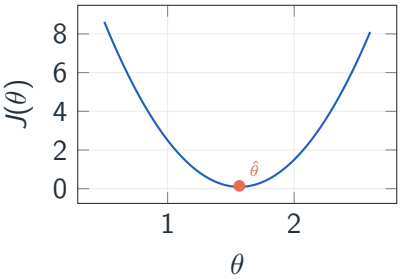
$\lambda\Omega(\boldsymbol{\theta})$ = negative log-prior; regularized ERM = MAP estimation (Chapter 2).

Worked example: ERM for a 1-D linear model

Data: $(1,2),(2,3),(3,5),(4,6)$. Class $\mathcal{H} = \{h(x) = \theta x\}$, squared loss.

$$J(\theta) = \frac{1}{4}\sum_{i=1}^{4}(y_i - \theta x_i)^2$$

$$\frac{dJ}{d\theta} = -\frac{2}{4}\sum_i x_i(y_i - \theta x_i) = 0 \;\Rightarrow\; \hat{\theta} = \frac{\sum_i x_i y_i}{\sum_i x_i^2}$$



$$\sum x_i y_i = 2 + 6 + 15 + 24 = 47$$
$$\sum x_i^2 = 1 + 4 + 9 + 16 = 30$$

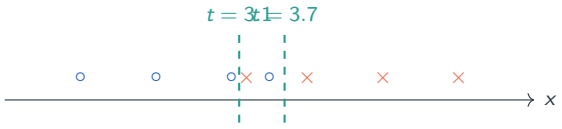
$$\hat{\theta} = \tfrac{47}{30} \approx 1.567, \quad \hat{R}_n(\hat{h}) \approx 0.147$$

ERM = find the bottom of the empirical loss curve. Chapter 3 generalizes to $\boldsymbol{\theta} \in \mathbb{R}^d$.

Worked example: ERM for a threshold classifier

Data (1-D): negatives at $x = 1, 2, 3, 3.5$, positives at $x = 3.2, 4, 5, 6$. Class $\mathcal{H} = \{h_t(x) = \mathbb{1}[x > t]\}$, 0–1 loss.

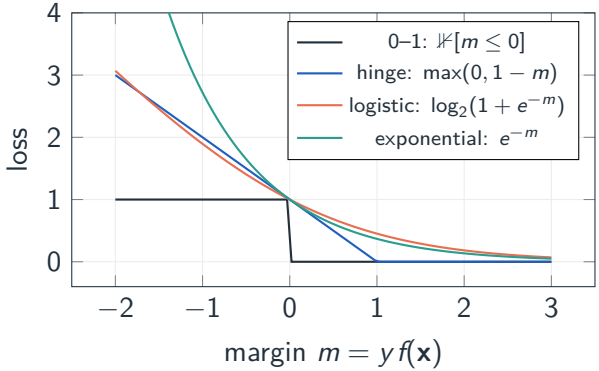
threshold t	$\hat{R}_n(h_t)$
2.5	2/8
3.1	1/8
3.3	2/8
3.7	1/8
4.5	3/8



Two ERM solutions with error 1/8; the data cannot be separated perfectly by any threshold — irreducible error for this $\mathcal{H}$.

Observation

0–1 loss is piecewise constant $\Rightarrow$ no gradient. Practical algorithms minimize a *surrogate* (log-loss, hinge), Chapter 4.



- All are convex upper bounds of the 0–1 loss $\Rightarrow$ tractable optimization; minimizing them also drives 0–1 error down (*classification calibration*).

Inductive bias

The set of assumptions a learner uses to generalize beyond the training data: the choice of $\mathcal{H}$, the loss, the regularizer, the optimizer.

Examples:

- Linear models: the relationship is (approximately) linear.
- k-NN: nearby points have similar labels (smoothness).
- Trees: axis-aligned rectangular regions.
- CNNs: translation invariance, locality.
- Regularization: small weights are more plausible.

No Free Lunch theorem (Wolpert, 1996)

Averaged over *all* possible problems, every learning algorithm has the same performance. There is no universally best model; bias must match the problem.

Consequence

Model selection is an empirical question — hence the central role of validation (Chapter 5)

Learning as optimization vs. learning as inference

Optimization view

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \; \hat{R}_n(\boldsymbol{\theta}) + \lambda \Omega(\boldsymbol{\theta})$$

- Point estimate
- Tools: gradient descent, convexity (Chapter 2)
- Covers most of Chapters 3–10

Probabilistic view

$$p(\boldsymbol{\theta} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})$$

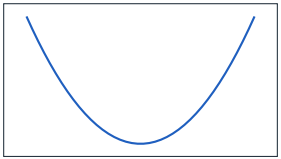
- MLE: $\arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D} \mid \boldsymbol{\theta})$
- MAP: $\arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$
- Full Bayesian: integrate over $\boldsymbol{\theta}$ (uncertainty)

Bridge

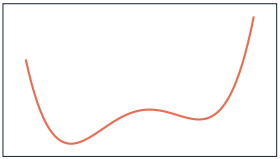
With Gaussian noise, MLE $\Leftrightarrow$ least squares; with a Gaussian prior on $\boldsymbol{\theta}$, MAP $\Leftrightarrow$ ridge regression. Same algorithms, two vocabularies.

- ERM is only useful if the arg min can be computed.
- **Convex** problems (linear regression, logistic regression, SVM): global optimum reachable; efficient solvers.
- **Non-convex** problems (neural nets, k-means, trees): local optima; heuristics (SGD, greedy splits, restarts).
- Complexity matters: normal equations cost $O(nd^2 + d^3)$; one epoch of SGD costs $O(nd)$.

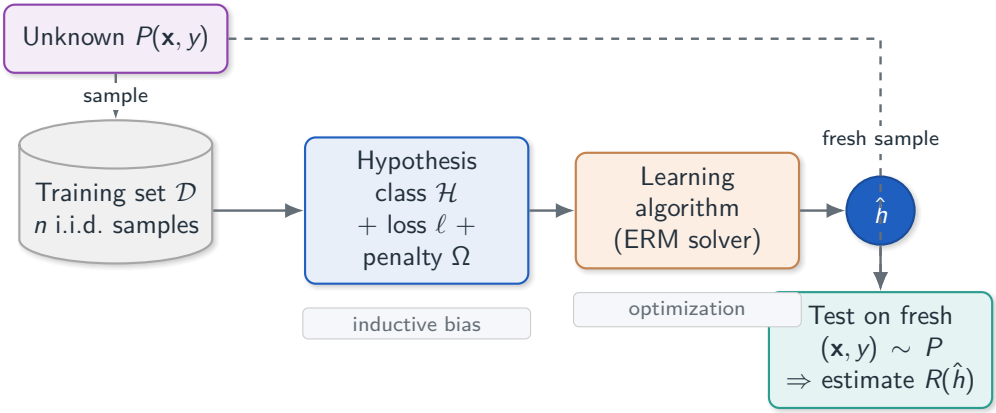
convex J



non-convex J



Putting it together: the learning problem in one picture



- Everything in this course is a specific choice of the blue, orange and green boxes.

Key formulas

- True risk $R(h) = \mathbb{E}[\ell(y, h(\mathbf{x}))]$; Bayes predictor $h^* = \arg\min R$; Bayes risk R^* is irreducible.
- Squared loss $\Rightarrow h^*(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}]$; 0–1 loss $\Rightarrow h^*(\mathbf{x}) = \arg\max_k P(y = k \mid \mathbf{x})$.
- Empirical risk $\hat{R}_n(h) = \frac{1}{n}\sum_i \ell(y_i, h(\mathbf{x}_i))$; ERM $\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{R}_n(h)$.
- Excess risk = estimation + approximation error; regularization $+\lambda\Omega(\boldsymbol{\theta})$ trades them off.
- Generalization gap bounded by $\sqrt{(\log|\mathcal{H}| + \log(2/\delta))/2n}$ for finite classes.

Generalization, Overfitting and Underfitting

Section 5 — Chapter 1

Generalization

A model is useful only if it performs well on data it has *never seen*. Memorizing the training set is trivial; generalizing is the whole difficulty.

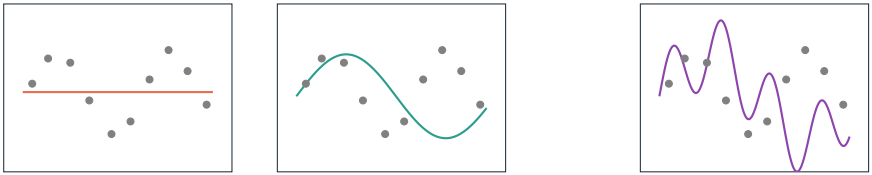
- Training error: $\hat{R}_n(\hat{h})$ on $\mathcal{D}_{\text{train}}$
- Test error: $\hat{R}_{\text{test}}(\hat{h})$ on $\mathcal{D}_{\text{test}}$, an unbiased proxy for $R(\hat{h})$
- Generalization gap: test $-$ train

Low train error + low test error	✔ good fit
High train error + high test error	underfitting
Low train error + high test error	overfitting

Capacity (informal)

The ability of $\mathcal{H}$ to fit a wide variety of functions. Grows with the number of parameters, polynomial degree, tree depth, smaller k in k-NN, smaller regularization λ, more training epochs.

degree 1 degree 3 degree 9



- Left: too rigid (underfits). Middle: about right. Right: passes through every point, wiggles between them (overfits).

Underfitting (high bias)

The model is too simple to capture the structure in the data: *both* training and test errors are high.

Symptoms

- Training error close to a trivial baseline (predict the mean / majority class)
- Systematic residual patterns (e.g. curved residuals with a linear fit)
- Adding data does not help

Remedies

- Increase capacity: more features, polynomial terms, deeper trees, bigger networks
- Decrease regularization λ
- Train longer (if using iterative optimization)
- Better features (feature engineering)

Overfitting (high variance)

The model fits the noise in the training set rather than the signal: training error is low, test error is high.

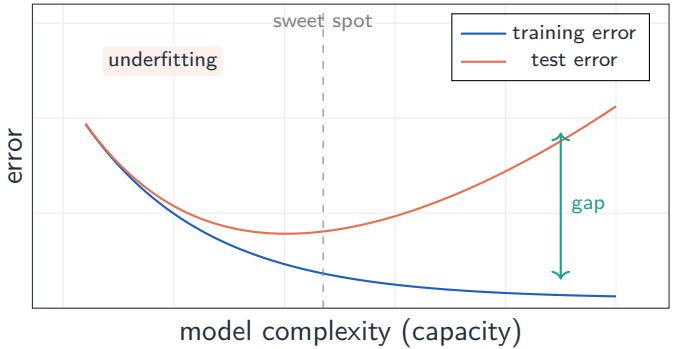
Symptoms

- Large gap between training and validation error
- Performance very sensitive to the particular training sample (retrain on a different split $\Rightarrow$ different model)
- Very large or erratic parameter values

Remedies

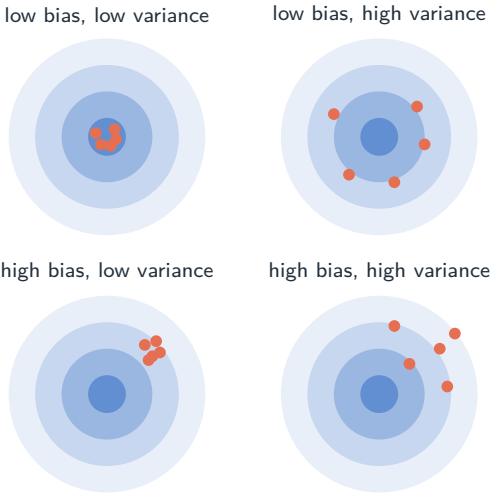
- More training data
- Regularization (ℓ_2, ℓ_1, dropout, pruning)
- Reduce capacity: fewer features, shallower trees
- Early stopping
- Ensembles (bagging averages out variance, Chapter 6)
- Data augmentation

Error vs. model complexity: the fundamental curve



- Training error decreases monotonically with capacity; test error is U-shaped. Model selection = finding the minimum of the red curve.

Bias and variance: intuition



Centre = truth. Each dot = the model obtained from a *different training sample*.

Bias: average distance of the dots' centre to the target.

Variance: spread of the dots.

Bias–variance decomposition (preview of Chapter 5)

For squared loss, with $y = f(\mathbf{x}) + \varepsilon$, $\mathbb{E}[\varepsilon] = 0$, $\mathrm{Var}(\varepsilon) = \sigma^2$, and $\hat{f}$ trained on a random sample $\mathcal{D}$:

Expected test error at a point x

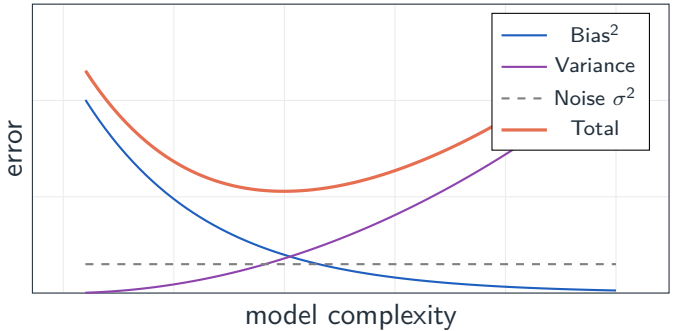
$$\mathbb{E}_{\mathcal{D},\varepsilon}\left[\left(y - \hat{f}(\mathbf{x})\right)^2\right] = \underbrace{\left(f(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\mathbf{x})]\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(\mathbf{x})]\right)^2\right]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Noise}}$$

- **Bias**: error from wrong assumptions (too simple a model). Decreases with capacity.
- **Variance**: sensitivity to the specific sample. Increases with capacity, decreases with n.
- **Noise**: irreducible, equals the Bayes risk.

The trade-off

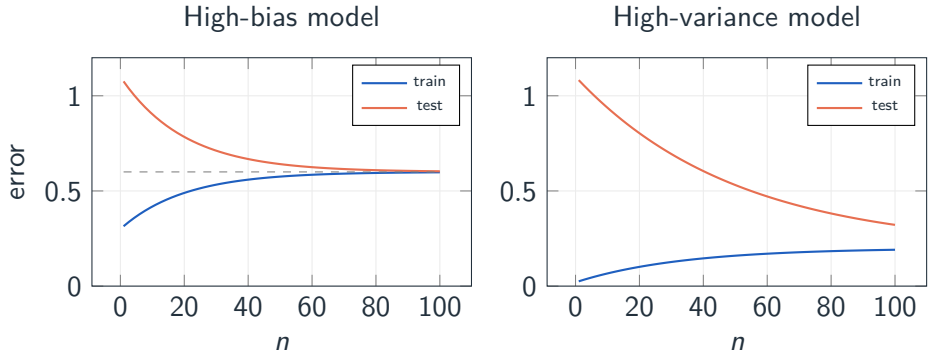
Total error is minimized at an intermediate capacity where Bias^2+Variance is smallest — the U-shape of the previous slide.

Bias–variance curves



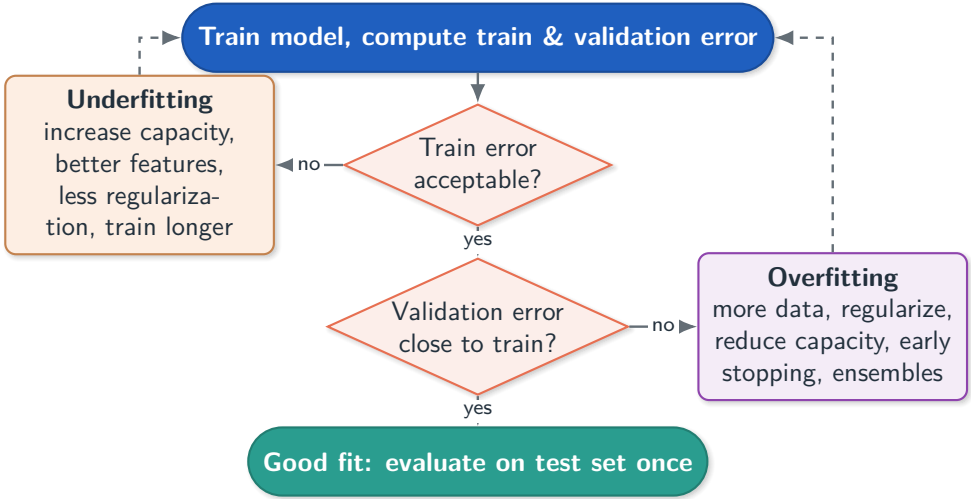
- Simple models: dominated by bias. Complex models: dominated by variance.
- More data lowers the variance curve $\Rightarrow$ the optimum moves to more complex models.

Learning curves: error vs. training set size



- **High bias**: curves converge quickly to a *high* plateau ⇒ more data will not help; need a richer model.
- **High variance**: large persistent gap ⇒ more data helps; so does regularization.
- Plot learning curves before deciding whether to collect more data.

Diagnosing fit: decision flowchart



Quick check: diagnose each situation

#	Train error	Validation error	Diagnosis
1	0.02	0.31	
2	0.28	0.30	
3	0.10	0.11	
4	0.25	0.60	
5	0.00	0.00	

Quick check: diagnose each situation

#	Train error	Validation error	Diagnosis
1	0.02	0.31	overfitting (high variance)
2	0.28	0.30	
3	0.10	0.11	
4	0.25	0.60	
5	0.00	0.00	

Quick check: diagnose each situation

#	Train error	Validation error	Diagnosis
1	0.02	0.31	overfitting (high variance)
2	0.28	0.30	underfitting (high bias) — if the baseline is much worse; else good fit near the noise floor
3	0.10	0.11	
4	0.25	0.60	
5	0.00	0.00	

Quick check: diagnose each situation

#	Train error	Validation error	Diagnosis
1	0.02	0.31	overfitting (high variance)
2	0.28	0.30	underfitting (high bias) — if the baseline is much worse; else good fit near the noise floor
3	0.10	0.11	good fit
4	0.25	0.60	
5	0.00	0.00	

Quick check: diagnose each situation

#	Train error	Validation error	Diagnosis
1	0.02	0.31	overfitting (high variance)
2	0.28	0.30	underfitting (high bias) — if the baseline is much worse; else good fit near the noise floor
3	0.10	0.11	good fit
4	0.25	0.60	both: too simple *and* unstable — check for a bug or leakage/shift between splits
5	0.00	0.00	

Quick check: diagnose each situation

#	Train error	Validation error	Diagnosis
1	0.02	0.31	overfitting (high variance)
2	0.28	0.30	underfitting (high bias) — if the baseline is much worse; else good fit near the noise floor
3	0.10	0.11	good fit
4	0.25	0.60	both: too simple *and* unstable — check for a bug or leakage/shift between splits
5	0.00	0.00	suspicious: probable leakage or trivial task

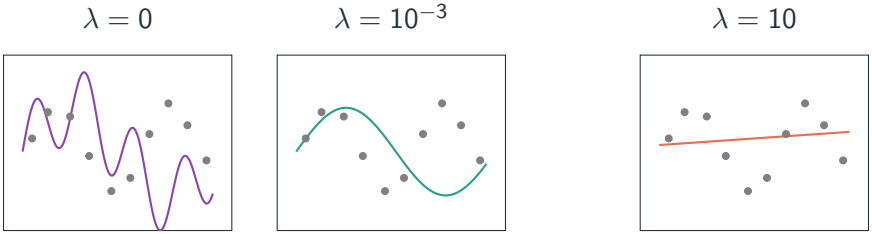
Quick check: diagnose each situation

#	Train error	Validation error	Diagnosis
1	0.02	0.31	overfitting (high variance)
2	0.28	0.30	underfitting (high bias) — if the baseline is much worse; else good fit near the noise floor
3	0.10	0.11	good fit
4	0.25	0.60	both: too simple *and* unstable — check for a bug or leakage/shift between splits
5	0.00	0.00	suspicious: probable leakage or trivial task

Always compare with a baseline

Errors are only meaningful relative to the trivial predictor and to the irreducible noise.

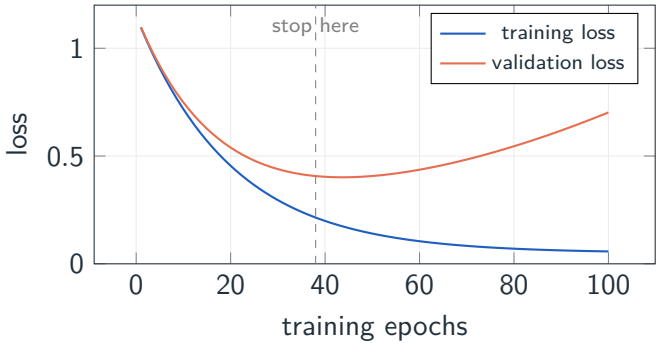
Regularization in action: ridge regression on a degree-9 polynomial



$$\hat{\boldsymbol{\theta}}_\lambda = \arg\min_{\boldsymbol{\theta}} \; \frac{1}{n} \sum_i \left(y_i - \boldsymbol{\theta}^\top \phi(x_i)\right)^2 + \lambda \left\|\boldsymbol{\theta}\right\|_2^2$$

- Same 10 parameters in all three panels; λ alone moves the model from overfitting to underfitting. λ is chosen by validation.

Early stopping as implicit regularization



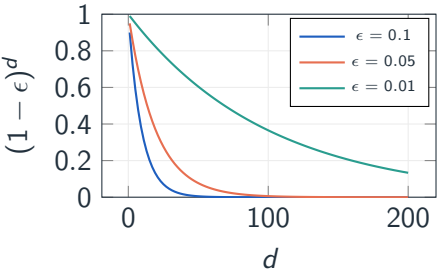
- For iterative learners (gradient descent, boosting), the number of iterations acts as a capacity knob.
- Stop when validation loss has not improved for p epochs (*patience*); keep the best checkpoint.

The curse of dimensionality (1/3): volume concentrates near the boundary

In d dimensions, consider the unit ball and the inner ball of radius $1-\epsilon$:

$$\frac{\mathrm{Vol}\big(B_{1-\epsilon}\big)}{\mathrm{Vol}\big(B_1\big)} = (1-\epsilon)^d \xrightarrow[d\to\infty]{} 0.$$

d	$\epsilon = 0.1$	$\epsilon = 0.05$	$\epsilon = 0.01$
2	0.81	0.90	0.98
10	0.35	0.60	0.90
100	$2.7 \cdot 10^{-5}$	0.006	0.37
1000	≈ 0	≈ 0	$4.3 \cdot 10^{-5}$

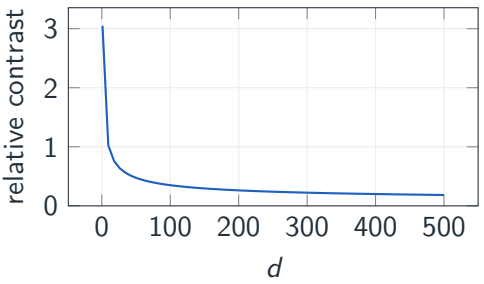


In high dimension, almost all the volume lies in a thin shell: "interior" points essentially do not exist.

The curse of dimensionality (2/3): distances become uninformative

For n points drawn uniformly in $[0,1]^d$, let $d_{\min}$ and $d_{\max}$ be the nearest and farthest distances from a query. Then (Beyer et al., 1999)

$$\frac{d_{\max} - d_{\min}}{d_{\min}} \xrightarrow[d\to\infty]{} 0 \quad \text{in probability.}$$



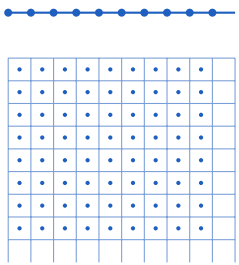
Consequences:

- "Nearest" neighbours are barely nearer than random points $\Rightarrow$ k-NN and kernel methods degrade.
- Euclidean distance loses meaning; need dimensionality reduction or learned metrics.

The curse of dimensionality (3/3): data requirements explode

To cover $[0,1]^d$ with a grid of cells of side ϵ, you need $(1/\epsilon)^d$ cells — and roughly that many samples to have one per cell.

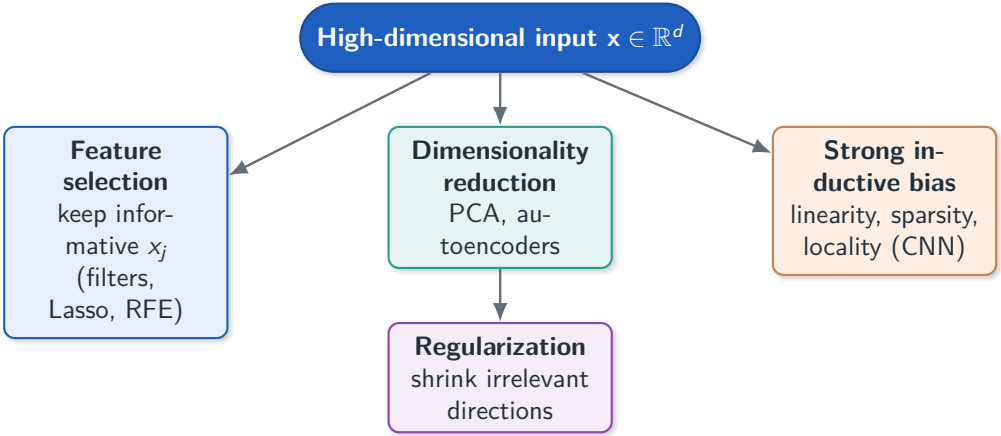
d	samples needed ($\epsilon = 0.1$)
1	10
2	100
5	10^5
10	10^{10}
100	10^{100}



Blessing of structure

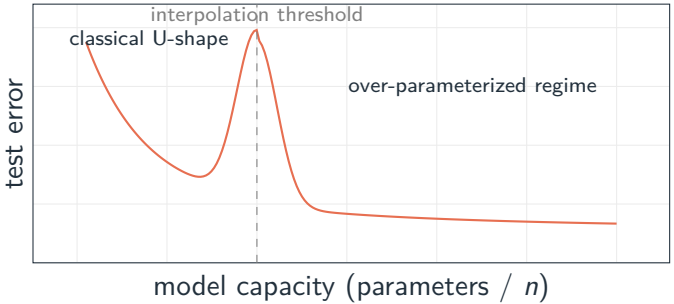
Real data are not uniform: they lie near low-dimensional manifolds and have sparse, redundant features. Good models exploit this (Chapter 8).

Fighting the curse of dimensionality



- Also: more data (when possible), and metric learning to make distances meaningful.

Double descent: a modern twist



- Belkin et al. (2019): for very large models (deep nets, kernel ridge with $\lambda \to 0$) test error can *decrease again* past the interpolation threshold.
- Does not invalidate the bias–variance intuition for classical models; it reveals implicit regularization by the optimizer.

Occam's razor

Among hypotheses that explain the data equally well, prefer the simplest.

- **MDL principle**: choose h minimizing $L(h) + L(\mathcal{D} \mid h)$, the bits to describe the model plus the bits to describe the data given the model.
- Regularized ERM is an instance: $\lambda\Omega(\boldsymbol{\theta})$ plays the role of $L(h)$, the loss the role of $L(\mathcal{D} \mid h)$.
- Information criteria (Chapter 5): $\text{AIC} = 2k - 2\log\hat{L}, \quad \text{BIC} = k\log n - 2\log\hat{L}$, where k = number of parameters.

Not a law of nature

Simplicity is a useful prior, not a guarantee. Some problems genuinely need complex models — validation decides.

Key ideas

- Underfitting: high bias, high train & test error. Overfitting: high variance, low train, high test error.
- Test error is U-shaped in capacity; training error decreases monotonically.
- $\mathbb{E}[(y - \hat{f})^2] = \text{Bias}^2 + \text{Variance} + \sigma^2$.
- Learning curves tell you whether more data will help.
- Regularization, early stopping, more data, ensembles: the anti-overfitting toolbox.
- Curse of dimensionality: $(1 - \epsilon)^d \to 0$, distances concentrate, $(1/\epsilon)^d$ samples to cover the space.

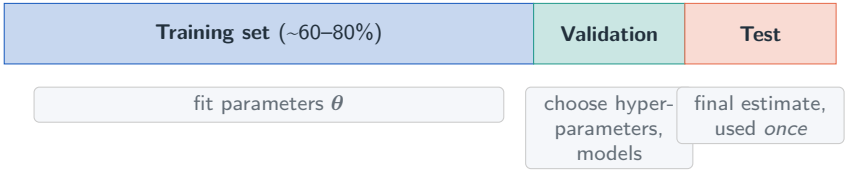
Data: Splitting, Leakage and Quality

Section 6 — Chapter 1

Why split the data?

Principle

Any data used to make a decision about the model (fitting parameters, choosing hyperparameters, selecting features) can no longer give an unbiased estimate of generalization.



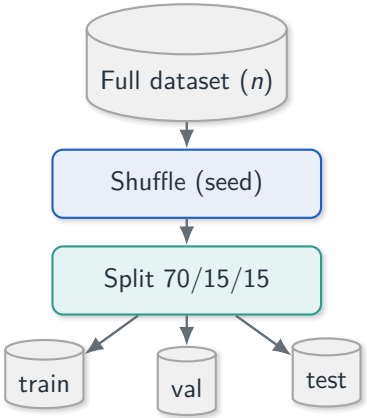
- Validation set $\neq$ test set: the validation set is used repeatedly and becomes optimistically biased too.
- Small datasets: replace the fixed validation set by cross-validation (next slides).

Hold-out split

- Shuffle, then cut: e.g. 70 / 15 / 15.
- Use a fixed random seed for reproducibility.
- Test set size: enough for a precise estimate. Standard error of an accuracy estimate:

$$\mathrm{SE} \approx \sqrt{\frac{\hat{p}(1-\hat{p})}{n_{\text{test}}}}$$

e.g. $\hat{p} = 0.9$, $n_{\text{test}} = 100 \Rightarrow \mathrm{SE} = 3\%$; $n_{\text{test}} = 10\,000 \Rightarrow 0.3\%$.

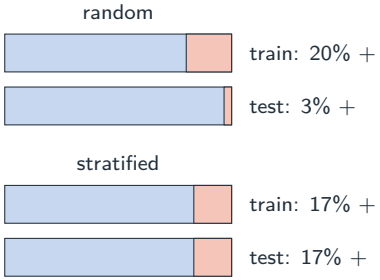


Stratification

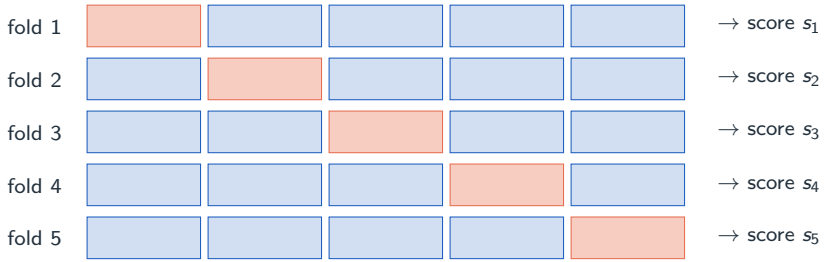
Split so that each subset has the same class proportions (or the same distribution of a key variable) as the full dataset.

Why: with a 2% positive class and $n_{\text{test}} = 200$, a random split may contain 0–8 positives $\Rightarrow$ wildly unstable recall estimates.

```
train_test_split(X, y,
test_size=0.2, stratify=y,
random_state=42)
```



k-fold cross-validation



- Every sample is used for validation exactly once. Typical $k = 5$ or 10. Cost: k trainings.
- Leave-one-out (LOO) is $k = n$: nearly unbiased but expensive and high-variance.

Cross-validation for model selection: algorithm

Input: data $\mathcal{D}$, candidate hyperparameters $\Lambda = \{\lambda_1, \dots, \lambda_m\}$, number of folds k

Partition $\mathcal{D}$ into folds $F_1, \dots, F_k$ (stratified if classification);

foreach $\lambda \in \Lambda$ **do**

  for $j = 1$ **to** k **do**

    Train $h_\lambda^{(-j)}$ on $\mathcal{D} \setminus F_j$;

    $s_j(\lambda) \leftarrow$ score of $h_\lambda^{(-j)}$ on F_j;

  $\bar{s}(\lambda) \leftarrow \frac{1}{k} \sum_j s_j(\lambda)$;

$\lambda^* \leftarrow \arg\max_\lambda \bar{s}(\lambda)$;

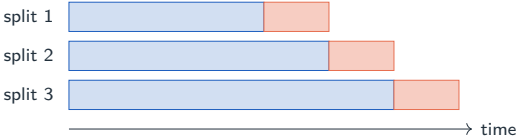
Retrain h_{λ^*} on all of $\mathcal{D}$;

Output: h_{λ^*} (evaluate once on a separate test set)

The test set must not participate in any of these steps; otherwise use *nested* CV (Chapter 5).

Splitting non-i.i.d. data

Time series: never train on the future.



Expanding-window (shown) or sliding-window CV.

Grouped data: samples from the same patient / user / device must stay in the same fold.



`GroupKFold`: patients A, B in train; C, D in validation. Otherwise the model memorizes patient identity.

Random splits on such data give absurdly optimistic scores

Leakage

Information from outside the training data — typically from the target or from the future — is used to build the model, producing performance that will not materialize in deployment.

Type	Example
Target leakage	feature "account_closed_date" when predicting churn
Train–test contamination	scaling with μ, σ computed on the full dataset before splitting
Temporal leakage	using next month's sales to predict this month's
Duplicate leakage	near-identical images in train and test
Group leakage	same patient in both sets
Preprocessing leakage	feature selection on all data, then CV

Leakage: the classic scaling mistake

Wrong

```
X_scaled = StandardScaler().fit_transform(X)
X_tr, X_te = train_test_split(X_scaled)
model.fit(X_tr, y_tr)
```

Test statistics leaked into μ, σ used on train. Mild here, but the same pattern with target encoding, imputation by target, or feature selection is severe.

Right

```
X_tr, X_te = train_test_split(X)
sc = StandardScaler().fit(X_tr)
model.fit(sc.transform(X_tr), y_tr)
model.predict(sc.transform(X_te))
```

Or simply: put the scaler inside a `Pipeline` and let CV handle it.

Rule

Fit every transform on training data only; *apply* it to validation/test data.

Detecting leakage

- Suspiciously high scores (much better than the literature or the baseline) — be paranoid, not proud.
- A single feature with near-perfect importance: check how and *when* it is generated.
- Ask for every feature: "Is this value known at prediction time, before the outcome?"
- Simulate deployment: build the features from a snapshot of the database as of date t, evaluate on outcomes after t.
- Hash / deduplicate samples across splits.

Real case

A pneumonia-detection model achieved excellent results because X-rays from the ICU (sicker patients) came from a portable machine with a visible marker. The model learned the marker, not the pneumonia.

Quick check: spot the leakage

For each feature used to predict whether a loan (granted at date t) will default within 12 months, decide: **OK** or **leak**?

1. Applicant's income at application time
2. Number of late payments in the 12 months after t
3. Credit score computed by the bank the day before t
4. "Account status" field, whose values include "written off"
5. Average default rate of the applicant's zip code, computed on the whole dataset
6. Loan amount requested

Quick check: spot the leakage

For each feature used to predict whether a loan (granted at date t) will default within 12 months, decide: **OK** or **leak**?

1. Applicant's income at application time — OK
2. Number of late payments in the 12 months after t
3. Credit score computed by the bank the day before t
4. "Account status" field, whose values include "written off"
5. Average default rate of the applicant's zip code, computed on the whole dataset
6. Loan amount requested

Quick check: spot the leakage

For each feature used to predict whether a loan (granted at date t) will default within 12 months, decide: **OK** or **leak**?

1. Applicant's income at application time — OK
2. Number of late payments in the 12 months after t — leak (future)
3. Credit score computed by the bank the day before t
4. "Account status" field, whose values include "written off"
5. Average default rate of the applicant's zip code, computed on the whole dataset
6. Loan amount requested

Quick check: spot the leakage

For each feature used to predict whether a loan (granted at date t) will default within 12 months, decide: **OK** or **leak**?

1. Applicant's income at application time — OK
2. Number of late payments in the 12 months after t — leak (future)
3. Credit score computed by the bank the day before t — OK
4. "Account status" field, whose values include "written off"
5. Average default rate of the applicant's zip code, computed on the whole dataset
6. Loan amount requested

Quick check: spot the leakage

For each feature used to predict whether a loan (granted at date t) will default within 12 months, decide: **OK** or **leak**?

1. Applicant's income at application time — OK
2. Number of late payments in the 12 months after t — leak (future)
3. Credit score computed by the bank the day before t — OK
4. "Account status" field, whose values include "written off" — leak (target proxy)
5. Average default rate of the applicant's zip code, computed on the whole dataset
6. Loan amount requested

Quick check: spot the leakage

For each feature used to predict whether a loan (granted at date t) will default within 12 months, decide: **OK** or **leak**?

1. Applicant's income at application time — OK
2. Number of late payments in the 12 months after t — leak (future)
3. Credit score computed by the bank the day before t — OK
4. "Account status" field, whose values include "written off" — leak (target proxy)
5. Average default rate of the applicant's zip code, computed on the whole dataset — leak (target encoding without out-of-fold)
6. Loan amount requested

Quick check: spot the leakage

For each feature used to predict whether a loan (granted at date t) will default within 12 months, decide: **OK** or **leak**?

1. Applicant's income at application time — OK
2. Number of late payments in the 12 months after t — leak (future)
3. Credit score computed by the bank the day before t — OK
4. "Account status" field, whose values include "written off" — leak (target proxy)
5. Average default rate of the applicant's zip code, computed on the whole dataset — leak (target encoding without out-of-fold)
6. Loan amount requested — OK

Missing values

Mechanisms (Rubin)

- **MCAR**: missing completely at random (sensor glitch)
- **MAR**: missingness depends on observed variables (older people skip the income field)
- **MNAR**: depends on the missing value itself (high earners hide income)

Strategies

- Drop rows (only if few and MCAR)
- Drop column (if mostly missing)
- Impute: mean/median/mode, k-NN, iterative (MICE)
- Add a missingness indicator $\mathbb{1}[x_j \text{ missing}]$ — often informative!
- Models that handle NA natively (XGBoost, LightGBM)

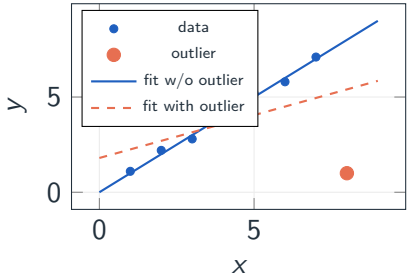
Imputation is a model

Fit it on the training set; applying the test median is leakage.

Detection

- z-score: $|x - \mu| / \sigma > 3$
- IQR rule: outside $[Q_1 - 1.5\,\mathrm{IQR},\ Q_3 + 1.5\,\mathrm{IQR}]$
- Multivariate: Mahalanobis distance $\sqrt{(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})}$, isolation forest

Treatment: remove (if error), winsorize (clip), transform (log), or use robust models / losses.



A single point can drag a least-squares fit — squared loss is not robust.

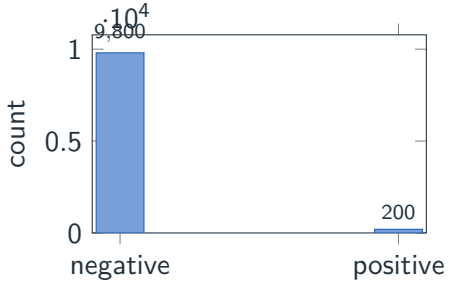
Class imbalance

Fraud, rare diseases, defects: positives are 0.1–5% of the data.

- Accuracy is misleading: predicting "negative" always gives 99%.
- Use precision, recall, F1, PR-AUC (Chapter 5).
- Stratify splits.
- Resampling: undersample majority, oversample minority (SMOTE), or reweight the loss:

$$J = \frac{1}{n} \sum_i w_{y_i} \, \ell(y_i, h(\mathbf{x}_i)), \qquad w_k \propto \frac{1}{n_k}$$

- Tune the decision threshold instead of using 0.5.



Label quality

- Labels are produced by humans or by proxies — they contain noise.
- Measure inter-annotator agreement (Cohen's κ); it upper-bounds achievable accuracy.
- Random label noise mostly increases variance; *systematic* noise (biased annotators) shifts the learned boundary.
- Strategies: multiple annotators + majority vote, confident-learning to find likely mislabels, robust losses, cleaning the most suspicious samples.

Rule of thumb

An hour spent fixing labels often beats a day spent tuning models.

Dataset documentation

Data dictionary: for each column

- name, type, unit, allowed range
- how it is generated and *when* it becomes available
- missing-value convention
- known issues

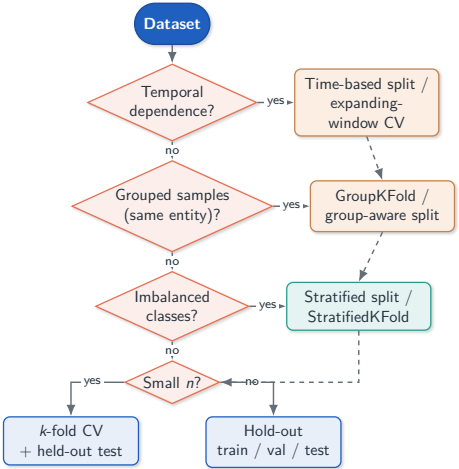
Datasheets for datasets (Gebru et al., 2021)

- Motivation, composition, collection process
- Preprocessing, uses, distribution, maintenance
- Ethical and legal considerations

Why bother

Six months later nobody remembers that `flag_3` was computed after the outcome. Documentation is leakage prevention.

Splitting strategy: decision flowchart



Section recap

Key ideas

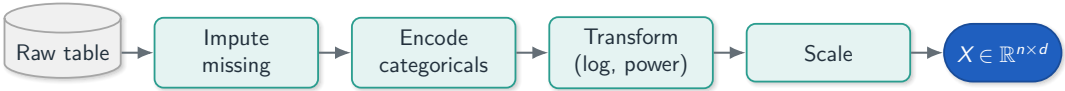
- Train / validation / test: fit, select, report — the test set is touched once.
- k-fold CV: $\frac{1}{k}\sum_j s_j$ with standard error $\mathrm{sd}(s_j)/\sqrt{k}$.
- Respect time and groups when splitting; stratify for imbalance.
- Leakage = using information unavailable at prediction time; fit all transforms on training data only.
- Missing values, outliers, imbalance and label noise are modelling decisions, not chores.

Feature Engineering and Preprocessing

Section 7 — Chapter 1

Why preprocess?

- Most algorithms assume **numerical**, **complete**, **comparably scaled** inputs.
- Distance-based (k-NN, SVM, k-means) and gradient-based (linear models, neural nets) methods are sensitive to feature scale.
- Tree-based models are scale-invariant but still need encoded categoricals (in most libraries) and benefit from good features.



Standardization

For feature j with training mean μ_j and standard deviation σ_j:

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}, \qquad \mu_j = \frac{1}{n}\sum_i x_{ij}, \quad \sigma_j = \sqrt{\frac{1}{n}\sum_i (x_{ij} - \mu_j)^2}$$

After transformation each feature has mean 0 and variance 1.

- Default choice for linear models, SVM, neural nets, PCA.
- Not bounded; outliers remain outliers (but shrunk).
- `StandardScaler`

Example

ages $\{20, 30, 40, 50, 60\}$: $\mu = 40$, $\sigma = 14.1$
$z = \{-1.41, -0.71, 0, 0.71, 1.41\}$

Min–max scaling and robust scaling

Min–max

$$x'_{ij} = \frac{x_{ij} - \min_j}{\max_j - \min_j} \in [0, 1]$$

- Bounded output: good for neural nets with bounded activations, image pixels.
- A single outlier squashes everything else.
- `MinMaxScaler`

Robust scaling

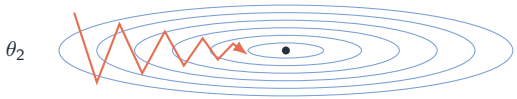
$$x'_{ij} = \frac{x_{ij} - \text{median}_j}{\text{IQR}_j}, \quad \text{IQR} = Q_3 - Q_1$$

- Uses quantiles: insensitive to outliers.
- `RobustScaler`

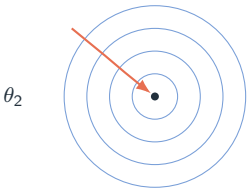


Why scaling matters: gradient descent

unscaled: $J = \theta_1^2 + 25\,\theta_2^2$



scaled: $J = \theta_1^2 + \theta_2^2$



- Elongated contours $\Rightarrow$ the gradient points away from the minimum, the step size must be tiny, the path zig-zags.
- Circular contours $\Rightarrow$ the gradient points straight at the minimum: fast convergence. Chapter 2 quantifies this via the condition number.

Two customers, features (age, income in €):

$$\mathbf{x}_a = (25,\ 30\,000), \quad \mathbf{x}_b = (60,\ 31\,000)$$

$$\|\mathbf{x}_a - \mathbf{x}_b\|_2 = \sqrt{35^2 + 1000^2} \approx 1000.6$$

The 35-year age difference is invisible; income dominates purely because of its unit.

After standardization (say $\sigma_{\text{age}} = 12$, $\sigma_{\text{inc}} = 15\,000$):

$$\|\mathbf{z}_a - \mathbf{z}_b\|_2 = \sqrt{\left(\tfrac{35}{12}\right)^2 + \left(\tfrac{1000}{15000}\right)^2} \approx 2.92$$

Now age drives the distance, as it should.

Affected methods

k-NN, k-means, SVM (RBF), PCA, any ℓ_2-regularized model (the penalty $\|\boldsymbol{\theta}\|^2$ treats all coefficients equally).

Non-linear transforms for skewed features

- $\log(1+x)$ for counts, incomes, durations (right-skewed, positive).
- $\sqrt{x}$: milder.
- **Box–Cox** ($x>0$): $x^{(\lambda)}=\dfrac{x^{\lambda}-1}{\lambda}$ ($\lambda\neq 0$), $\log x$ ($\lambda=0$); λ fitted by maximum likelihood.
- **Yeo–Johnson**: Box–Cox extended to $x\leq 0$.
- **Quantile transform**: map to a uniform or Gaussian via the empirical CDF.



One-hot encoding

A feature with m categories becomes m binary columns (or $m-1$ to avoid collinearity in linear models):

$$\text{city} = \text{Meknès} \longmapsto (\underbrace{0}_{\text{Fès}}, \underbrace{1}_{\text{Meknès}}, \underbrace{0}_{\text{Rabat}})$$

- No spurious order between categories.
- Works with every model.
- Explodes for high cardinality (10 000 zip codes $\rightarrow$ 10 000 columns), sparse matrices help.
- Unseen category at test time: `handle_unknown="ignore"` $\rightarrow$ all zeros.

city	Fès	Meknès	Rabat
Fès	1	0	0
Meknès	0	1	0
Rabat	0	0	1
Fès	1	0	0

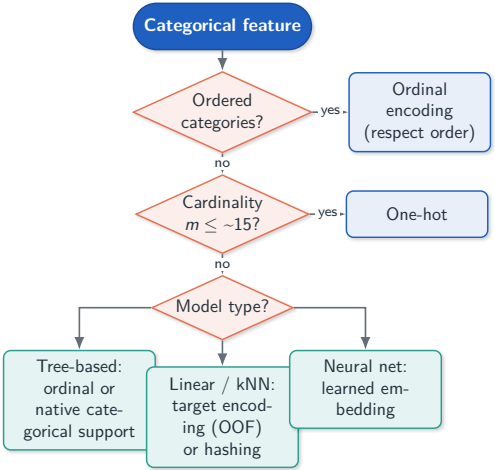
Encoding categorical features: ordinal, target, hashing, embeddings

Method	Idea	When
Ordinal / label	map to integers $0, 1, \ldots, m-1$	truly ordered categories (S<M<L); trees tolerate it for nominal too
Target (mean) encoding	replace category c by $\dfrac{\sum_{i:x_i=c} y_i + \alpha\,\bar{y}}{n_c + \alpha}$ (smoothed)	high cardinality; must be computed out-of-fold to avoid leakage
Frequency / count	replace by n_c or n_c/n	quick, captures rarity
Hashing	hash category to one of B buckets	huge vocabularies, streaming
Embeddings	learn a dense vector $\mathbf{e}_c \in \mathbb{R}^k$ jointly with the model	neural nets, very high cardinality (words, product ids)

Target encoding leakage

Using the training target to encode a training row gives the model a peek at its own label. Use k-fold out-of-fold statistics.

Encoding decision flowchart



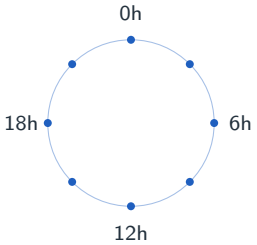
Handling datetime features

A timestamp is rarely useful raw. Extract:

- year, month, day, weekday, hour
- is_weekend, is_holiday, quarter
- elapsed time since a reference (account age)
- time since last event, time to next known event

Cyclic encoding: hour 23 and hour 0 are neighbours, integers don't know that.

$$h \mapsto \left(\sin \frac{2\pi h}{24},\ \cos \frac{2\pi h}{24}\right)$$



Text features (preview)

Bag of words: document $\to$ vector of word counts over a vocabulary of size V.

$$\text{TF-IDF}(t, d) = \text{tf}(t, d) \cdot \log \frac{N}{1 + \text{df}(t)}$$

- tf: frequency of term t in document d
- df: number of documents containing t; N total documents
- Rare, document-specific words get high weight.

Documents: d_1 = "good movie", d_2 = "bad movie", d_3 = "good good film"

	good	movie	bad	film
d_1	1	1	0	0
d_2	0	1	1	0
d_3	2	0	0	1

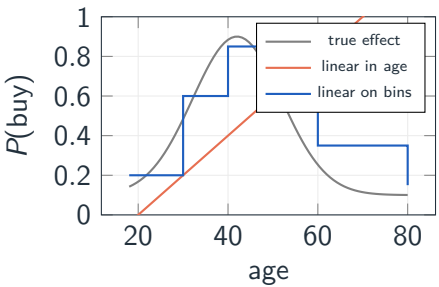
Pipeline: tokenize $\to$ lowercase $\to$ remove stop-words $\to$ (stem) $\to$ count $\to$ TF-IDF $\to$ linear model. Dense embeddings in Chapter 10.

Discretization (binning)

Replace a continuous feature by the interval it falls in:

- Equal-width bins, equal-frequency (quantile) bins, or domain thresholds (age groups)
- Then one-hot encode the bins

Pros: lets linear models capture non-monotonic effects; robust to outliers.
Cons: loses information within bins; sharp edges.



Feature map

$\phi : \mathbb{R}^d \to \mathbb{R}^D$ with $D > d$, e.g. degree-2 polynomial for $d = 2$:

$$\phi(x_1, x_2) = (1,\ x_1,\ x_2,\ x_1^2,\ x_1 x_2,\ x_2^2)$$

A linear model in $\phi(\mathbf{x})$ is a non-linear model in $\mathbf{x}$.

- Number of degree-p monomials in d variables: $\binom{d+p}{p}$ — grows fast ($d = 100$, $p = 3$: 176 851).
- Always combine with regularization and scaling.
- Kernel methods (Chapter 7) compute inner products $\phi(\mathbf{x})^\top \phi(\mathbf{x}')$ without building ϕ explicitly.

Interaction

$x_1 x_2$ lets the effect of x_1 depend on x_2: e.g. discount $\times$ is_weekend.

Feature selection: three families

Filter
Score each feature independently of the model: correlation with y, mutual information, χ^2, variance threshold.
fast, ignores interactions

Wrapper
Search subsets using model performance: forward selection, backward elimination, RFE.
accurate, expensive

Embedded
Selection happens during training: Lasso (ℓ_1 zeros out weights), tree feature importance.
cheap, model-specific

- Fewer features $\Rightarrow$ less variance, faster, more interpretable.
- Select on training folds only (feature selection is part of the model).

Mutual information

$$I(X;Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} = H(Y) - H(Y \mid X) \geq 0$$

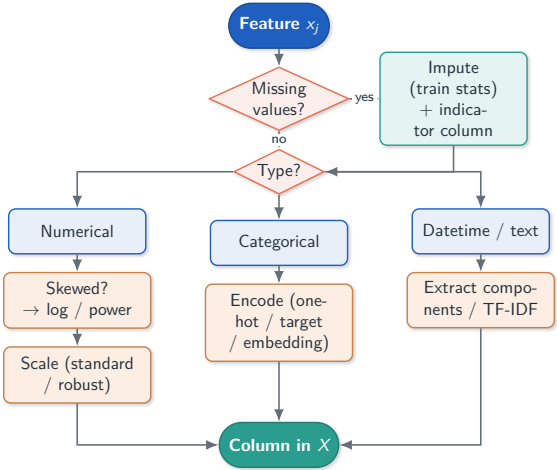
The reduction in uncertainty about Y from knowing X. Zero iff independent; captures non-linear dependence, unlike correlation.

Toy: Y = buys, X = has_coupon.

	$Y=0$	$Y=1$
$X=0$	0.40	0.10
$X=1$	0.15	0.35

$p(X{=}1) = 0.5$, $p(Y{=}1) = 0.45$
$H(Y) = -0.55 \log_2 0.55 - 0.45 \log_2 0.45 = 0.993$ bits
$H(Y \mid X) = 0.5\, H(0.2) + 0.5\, H(0.7) = 0.5(0.722) + 0.5(0.881) = 0.802$
$I(X;Y) = 0.993 - 0.802 = \mathbf{0.191}$ bits

Preprocessing decision flowchart (per feature)



Which models need which preprocessing?

Model	Scaling	One-hot	Missing values	Outlier sensitivity
Linear / logistic regression	required	required	impute	high (squared loss)
k-NN, k-means	required	required	impute	high
SVM (RBF)	required	required	impute	medium
Neural networks	required	or embeddings	impute	medium
Decision trees	no	ordinal OK	some handle natively	low
Random forest / boosting	no	ordinal OK	XGBoost/LightGBM native	low
Naive Bayes	no	counts	impute	low

Habit

Build a `ColumnTransformer` with a numerical branch (impute $\rightarrow$ scale) and a categorical branch (impute $\rightarrow$ encode); reuse it for every model.

Worked example: preprocessing a small table by hand

Training rows: (age, city, income) = (30, Fès, 4000), (50, Rabat, NA), (40, Fès, 6000). Test row: (35, Meknès, 5000).

1. **Impute** income with the training median: median$\{4000, 6000\} = 5000 \Rightarrow$ row 2 income $= 5000$.
2. **Scale** age: $\mu = 40$, $\sigma = \sqrt{(100 + 100 + 0)/3} = 8.16$; income: $\mu = 5000$, $\sigma = 816$.
3. **One-hot** city over training categories $\{$Fès, Rabat$\}$.

row	age$_z$	income$_z$	Fès	Rabat
1	-1.22	-1.22	1	0
2	1.22	0.00	0	1
3	0.00	1.22	1	0
test	-0.61	0.00	0	0

The test row uses *training* statistics; "Meknès" is unseen $\Rightarrow$ all-zero encoding.

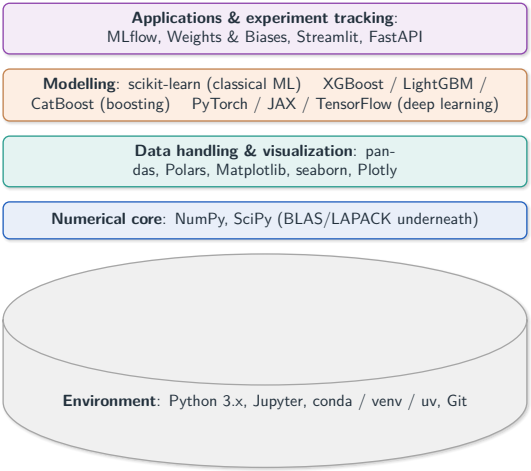
Key formulas and rules

- Standardization $z = (x - \mu)/\sigma$; min–max $(x - \min)/(\max - \min)$; robust $(x - \text{median})/\text{IQR}$.
- Scale for distance- and gradient-based models; trees don't care.
- One-hot for low cardinality; target encoding (out-of-fold) / embeddings for high cardinality.
- $\log(1 + x)$, Box–Cox for skew; cyclic $(\sin, \cos)$ for periodic features; TF-IDF for text.
- Feature selection: filter, wrapper, embedded; mutual information $I(X; Y) = H(Y) - H(Y \mid X)$.
- Every transform is fitted on training data only and lives inside the pipeline.

Tools and Ecosystem

Section 8 — Chapter 1

The Python ML stack



- This course: NumPy for understanding, scikit-learn for practice, PyTorch from Chapter 9.

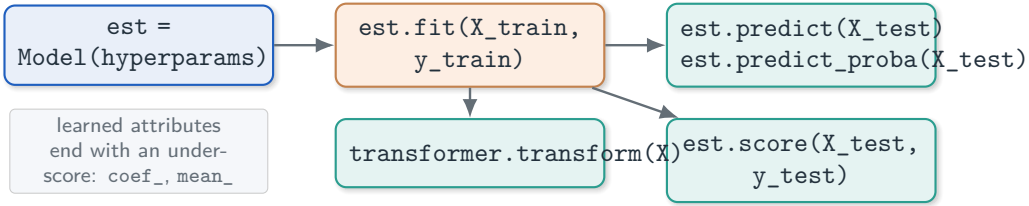
NumPy in five lines

Design matrix and least squares

```
import numpy as np
X = np.array([[1, 80], [1, 120], [1, 60]])  # n
x (d+1)
y = np.array([180, 310, 120])
theta = np.linalg.solve(X.T @ X, X.T @ y)
y_hat = X @ theta
```

- Arrays are the lingua franca; `@` is matrix multiplication.
- `np.linalg.solve` rather than explicit inversion (numerically stabler).
- Broadcasting: `(X - X.mean(0)) / X.std(0)` standardizes all columns at once.
- Vectorize: avoid Python loops over samples.

scikit-learn: the estimator API



- Same interface for every model ⇒ swap `LogisticRegression` for `RandomForestClassifier` in one line.
- `Pipeline`, `ColumnTransformer`, `GridSearchCV`, `cross_val_score` compose estimators.

A complete minimal script

Iris classification end to end

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

X, y = load_iris(return_X_y=True)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.2, stratify=y, random_state=0)

clf = make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000))
print("CV accuracy:", cross_val_score(clf, X_tr, y_tr, cv=5).mean())
clf.fit(X_tr, y_tr)
print("Test accuracy:", clf.score(X_te, y_te))
```

- Stratified split, CV on the training part only, one final test score. Every principle of this chapter in 12 lines.

Where to find data

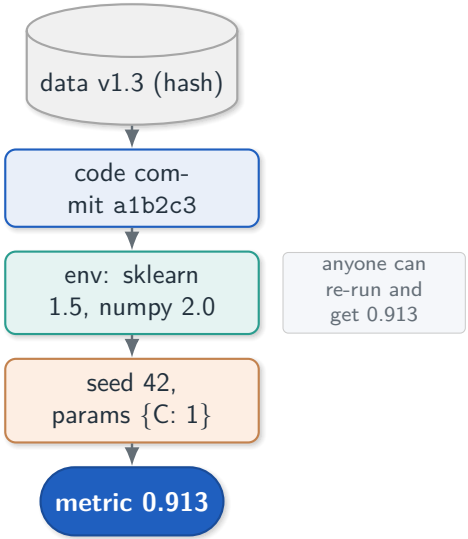
Source	Content	Notes
UCI ML Repository	classic tabular datasets	small, well documented
OpenML	thousands of datasets + benchmark results	`sklearn.datasets.fetch_openml`
Kaggle	competitions, community datasets	real-world messiness
HuggingFace Datasets	NLP, vision, audio	one-line loading
scikit-learn built-ins	iris, digits, wine, California housing	for teaching
Government open data	census, transport, health	licences vary
torchvision / TFDS	MNIST, CIFAR, ImageNet	deep learning

Check the licence and the collection process

Public $\neq$ free to use for any purpose; and famous datasets have known biases and label errors.

Reproducibility checklist

- Fix random seeds (`random_state`, `np.random.seed`, `torch.manual_seed`)
- Pin library versions (`requirements.txt`, `environment.yml`, lock files)
- Version code (Git) and data (DVC, hashes)
- Script everything: no manual spreadsheet edits
- Log hyperparameters and metrics for every run



Hardware and scale

Regime	Data size	Tools	Notes
Laptop	$< 10^6$ rows, $< 10^3$ features	pandas + scikit-learn	most classical ML lives here
Single GPU	images, text, 10^6–10^8 samples	PyTorch	this course's deep learning chapters
Multi-GPU / cluster	foundation models	PyTorch DDP, JAX, Spark	beyond scope

- Rule: prototype on a subsample; scale only once the pipeline works.
- Complexity awareness: $O(n^2)$ memory (kernel matrices, pairwise distances) breaks at $n \approx 10^5$.
- Free GPU notebooks: Google Colab, Kaggle kernels.

Notebooks vs. scripts

Jupyter notebooks

- Exploration, plots, teaching
- Iterative, visual
- Hidden state, hard to test and version

Python modules / scripts

- Reusable pipelines, training jobs
- Testable, versionable, runnable in CI
- Less interactive

Workflow

Explore in a notebook → move stable code into `src/features.py`, `src/train.py` → import them back into the notebook.

Course logistics and resources

Textbooks

- Hastie, Tibshirani, Friedman — *The Elements of Statistical Learning*
- Bishop — *Pattern Recognition and Machine Learning*
- Murphy — *Probabilistic Machine Learning*
- Géron — *Hands-On ML with Scikit-Learn, Keras & TensorFlow*
- Goodfellow, Bengio, Courville — *Deep Learning*

Practice

- Lab 1: EDA + preprocessing pipeline on a tabular dataset
- Each chapter: notebook re-implementing the algorithm in NumPy, then scikit-learn
- Project: full pipeline on a dataset of your choice, with a report

Online: scikit-learn user guide, *Distill.pub*, Stanford CS229 notes.

Section recap

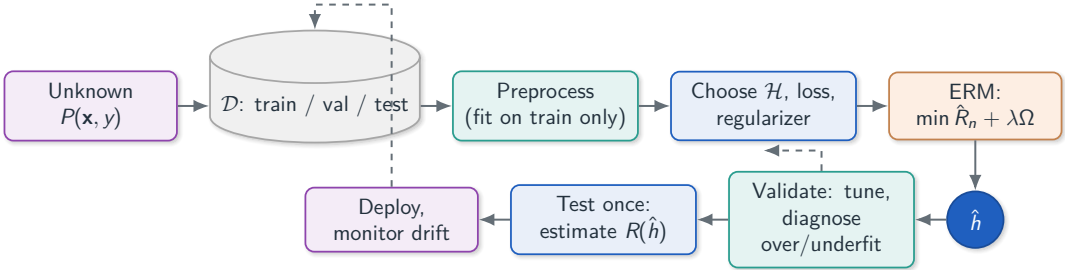
Key ideas

- Stack: NumPy $\rightarrow$ pandas $\rightarrow$ scikit-learn $\rightarrow$ PyTorch; Jupyter for exploration, scripts for pipelines.
- scikit-learn: `fit` / `predict` / `transform` everywhere; compose with `Pipeline`.
- Reproducibility: seeds, pinned versions, versioned data and code, logged experiments.
- Know the scale regime you are in and the complexity of your algorithms.

Chapter Summary and Exercises

Section 9 — Chapter 1

Chapter 1 in one picture



Key equations of Chapter 1

True risk	$R(h) = \mathbb{E}_{(\mathbf{x},y)\sim P}[\ell(y, h(\mathbf{x}))]$
Empirical risk	$\hat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, h(\mathbf{x}_i))$
ERM	$\hat{h} = \arg\min_{h\in\mathcal{H}} \hat{R}_n(h)$
Regularized ERM	$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \hat{R}_n(\boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\theta})$
Bayes predictors	$\mathbb{E}[y \mid \mathbf{x}]$ (squared), $\arg\max_k P(y = k \mid \mathbf{x})$ (0–1)
Excess risk	estimation error + approximation error
Bias–variance	$\mathbb{E}[(y - \hat{f})^2] = \text{Bias}^2 + \text{Var} + \sigma^2$
Hoeffding	$\mathbb{P}\left(\left\lvert R - \hat{R}_n \right\rvert > \epsilon\right) \le 2e^{-2n\epsilon^2}$
Curse of dimensionality	$(1-\epsilon)^d \to 0$; $(1/\epsilon)^d$ samples to cover $[0,1]^d$
Standardization	$z = (x - \mu)/\sigma$
Mutual information	$I(X;Y) = H(Y) - H(Y \mid X)$
CV estimate	$\frac{1}{k}\sum_j s_j$, $\text{SE} = \text{sd}(s_j)/\sqrt{k}$

Key vocabulary

- Task, performance, experience
- Supervised / unsupervised / RL
- Regression / classification
- Feature, label, sample
- Hypothesis class, parameters
- Hyperparameters
- Loss, risk, empirical risk
- ERM, regularization
- Bayes predictor, Bayes risk
- Inductive bias
- Generalization gap
- Bias, variance
- Overfitting, underfitting
- Capacity, learning curve
- Curse of dimensionality
- Train / validation / test
- Cross-validation, stratification
- Leakage, drift

- "More complex models are always better." — Test error is U-shaped; complexity must match data size and noise.
- "High training accuracy means a good model." — Training error is optimistically biased; only held-out data counts.
- "Deep learning replaces feature engineering." — For tabular data, boosted trees with good features still dominate.
- "The test set can be used a few times." — Every look leaks information; use validation/CV for iteration.
- "Scaling is optional." — Not for distance- or gradient-based methods.
- "Correlation with the target means the feature is useful." — Check whether it is available at prediction time (leakage).

Exercises (1/4): concepts

1. Give (T, P, E) for: (a) an e-mail auto-complete system, (b) a system that detects defective parts on a production line, (c) a chess engine trained by self-play.
2. For each, state whether it is supervised, unsupervised, self-supervised or reinforcement learning, and what the input/output spaces are.
3. Explain in your own words why the empirical risk of the *selected* hypothesis $\hat{h}$ is an optimistic estimate of its true risk, while it is unbiased for a fixed h.
4. A colleague reports 99.5% accuracy on a fraud dataset with 0.3% fraud. Why is this not impressive? Which quantities would you ask for?
5. Give one example each of a parametric and a non-parametric model, and explain the trade-off.

Exercises (2/4): calculations

6. Data (x, y): $(0,1), (1,3), (2,4), (3,4)$. For $\mathcal{H} = \{h(x) = \theta x + 1\}$ with squared loss, derive $J(\theta)$, compute $\hat{\theta}$ and $\hat{R}_n(\hat{h})$.
7. Show that for the absolute loss, the Bayes predictor is the conditional median. (Hint: differentiate $\mathbb{E}[|y - c| \mid \mathbf{x}]$ w.r.t. c.)
8. Compute $(1 - \epsilon)^d$ for $\epsilon = 0.02$ and $d \in \{10, 100, 1000\}$. In which case does more than half the volume of the unit ball lie within ϵ of its surface?
9. Standardize the feature $\{2, 4, 4, 4, 5, 5, 7, 9\}$. Then min–max scale it. Which value is affected most if we append an outlier 100?
10. Compute the mutual information between X and Y for the joint table $p(0,0) = 0.3$, $p(0,1) = 0.2$, $p(1,0) = 0.1$, $p(1,1) = 0.4$.

Exercises (3/4): diagnosis

11. A model has training RMSE 2.1 and validation RMSE 2.3; the baseline (predict the mean) has RMSE 9.0. Diagnose. What would you try next?
12. A model has training accuracy 100% and validation accuracy 71%. Diagnose and list four remedies in order of preference.
13. You observe a learning curve where training and validation errors converge to 0.35 by $n = 2000$ and stay flat until $n = 20\,000$. Will collecting 100 000 samples help? Why?
14. A churn model uses the feature "number of support calls in the 30 days after the prediction date". What is wrong? How would you detect it if the feature name were not explicit?
15. You split 10 000 medical images randomly into train/test and get 97% accuracy. Each patient contributed about 10 images. Explain why the score is unreliable and how to fix the split.

Exercises (4/4): programming

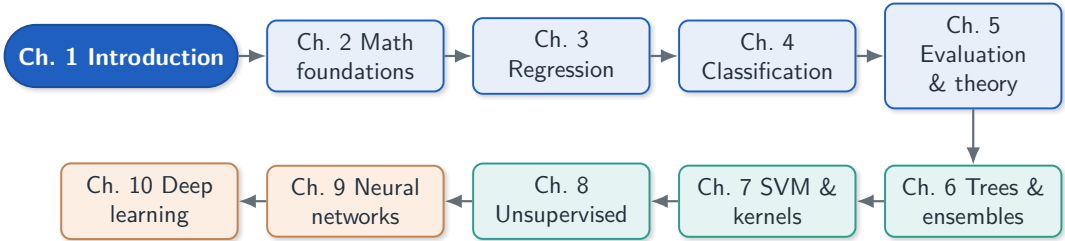
16. Load the California housing dataset (`fetch_california_housing`). Perform EDA: distributions, missing values, correlations with the target.
17. Build a `ColumnTransformer` + `LinearRegression` pipeline. Compare validation RMSE with and without standardization, and with log-transformed skewed features.
18. Implement 5-fold cross-validation *by hand* with NumPy (no `cross_val_score`), and check that it matches scikit-learn's result.
19. Plot learning curves (training and validation error vs. n) for a linear model and for a decision tree of depth 20. Interpret.
20. Introduce deliberate leakage (scale on the full dataset before splitting; then add a feature derived from y). Measure how much each inflates the validation score.

Deliverable

A notebook + a one-page report answering exercises 16–20, with a reproducible seed.

Ex. 6. $J(\theta) = \frac{1}{4}\sum_i (y_i - 1 - \theta x_i)^2$ with residual targets $y_i - 1 = \{0, 2, 3, 3\}$:
$\hat{\theta} = \frac{\sum x_i(y_i-1)}{\sum x_i^2} = \frac{0+2+6+9}{0+1+4+9} = \frac{17}{14} \approx 1.214$;
$\hat{R}_n = \frac{1}{4}\left[0^2 + (2-1.214)^2 + (3-2.429)^2 + (3-3.643)^2\right] \approx 0.339$.
Ex. 8. $0.98^{10} = 0.817$, $0.98^{100} = 0.133$, $0.98^{1000} \approx 1.7 \cdot 10^{-9}$. For $d = 100$ and $d = 1000$, more than half the volume is within ϵ of the surface.
Ex. 10. $p(X{=}0) = 0.5$, $p(Y{=}0) = 0.4$. $I = \sum p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)} =$
$0.3 \log_2 \frac{0.3}{0.2} + 0.2 \log_2 \frac{0.2}{0.3} + 0.1 \log_2 \frac{0.1}{0.2} + 0.4 \log_2 \frac{0.4}{0.3} = 0.175 - 0.117 - 0.1 + 0.166 = \mathbf{0.125}$ bits.
Ex. 11. Small gap, both far below baseline: reasonable fit; slight room to reduce bias (richer features / model) — check residual plots first.
Ex. 12. Overfitting: (1) more data / augmentation, (2) regularization, (3) reduce capacity, (4) ensembles.

What comes next



- Chapter 2 builds the mathematical toolkit: linear algebra, calculus, probability, information theory and convex optimization — everything used to derive the algorithms of Chapters 3–10.