AI Research Labs

Section 2 of 6 · 8 lessons

Statistical Learning and Model Evaluation

How to tell whether a model is learning something real, why a low training error is not necessarily good news, and how to avoid misleading evaluation results.

2.1 Loss Functions and Risk

2.2 Empirical Risk Minimisation (ERM)

2.3 Underfitting and Overfitting

2.4 Generalisation and the Generalisation Gap

2.5 The Bias–Variance Trade-off

2.6 Training, Validation and Test Sets

2.7 Cross-validation

2.8 Data Leakage and Good ML Practices



Free course with quizzes, progress tracking and Scoopy, an AI tutor:
airesearchlabs.si/learn/ml

Machine Learning Foundations · Edition November 2026 · © 2026 AI Research Labs. Free for personal study.

Where this section fits

Every lesson is a step in building a model. These are the steps this section teaches; the Big picture PDF and the online map show how they connect to the rest.

2

SUPERVISED LEARNING · STEP 2

Get, split and prepare the data

How do I keep part of the data honest for the final check?

Lessons here: 2.6, 2.8 · feeds into Evaluate honestly, Control complexity, Turn predictions into decisions, Measure similarity

4

SUPERVISED LEARNING · STEP 4

Evaluate honestly

Will it work on customers it has never seen?

Lessons here: 2.3, 2.4, 2.7 · feeds into Control complexity

◆

UNDERNEATH BOTH MAPS · FOUNDATION

Loss functions

Lessons here: 2.1 · feeds into Fit it, Fit it

◆

UNDERNEATH BOTH MAPS · FOUNDATION

Empirical risk minimisation

Lessons here: 2.2 · feeds into Fit it, Fit it, Control complexity

◆

UNDERNEATH BOTH MAPS · FOUNDATION

Bias–variance trade-off

Lessons here: 2.5 · feeds into Evaluate honestly, Control complexity

Loss Functions and Risk

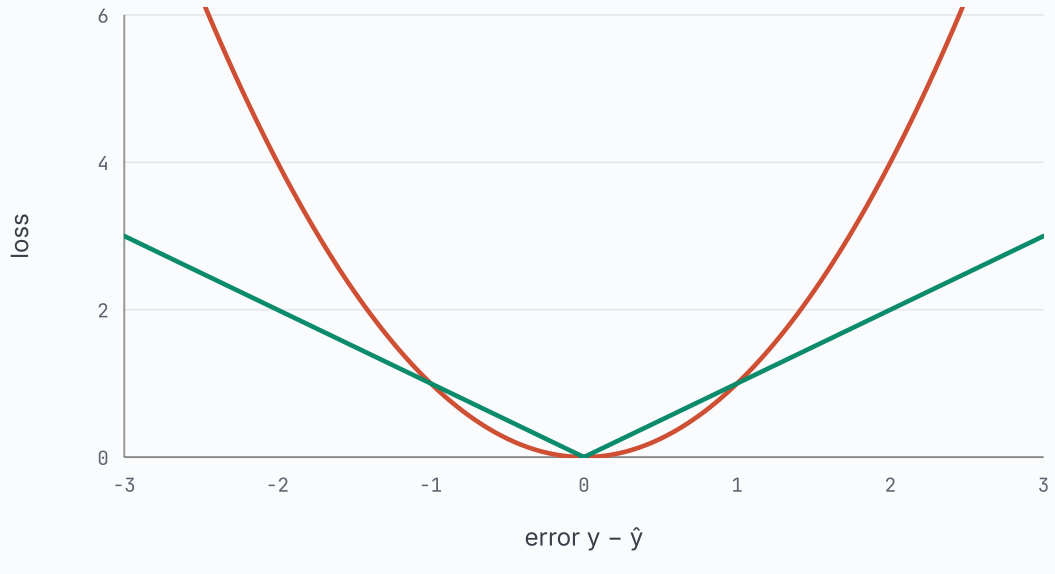
A loss function scores one prediction; risk averages it. Squared, absolute, percentage, 0–1, cross-entropy, cost-weighted and hinge loss: what each punishes, where each is used, and how to compute them by hand.

What a loss function does

A **loss function** $L(y, \hat{y})$ gives a number for how bad one prediction is: zero for perfect, larger for worse. **Risk** is the average loss (lesson 1.5). Choosing the loss is a modelling decision: it should reflect the task, the data and what errors actually cost.

Regression losses

LOSS	ONE OBSERVATION	BEHAVIOUR
Squared (L2)	$(y - \hat{y})^2$	An error of 4 costs 16, an error of 2 costs 4. Large errors dominate. Smooth and easy to minimise, so linear models get a closed-form solution. Sensitive to outliers. Used in OLS, ridge and LASSO.
Absolute (L1)	$\lvert y - \hat{y} \rvert$	An error of 4 costs 4. Less affected by outliers. Not smooth at zero. Averaged, it is the MAE.
Percentage	$\lvert y - \hat{y} \rvert / \lvert y \rvert$	Relative error (MAPE). Breaks down when y is near zero.



Squared loss grows quadratically, so one large error can outweigh many small ones. Absolute loss grows linearly.

Classification losses

LOSS	FORMULA	BEHAVIOUR
0–1 (misclassification)	$1[y \neq \hat{y}]$	1 if wrong, 0 if right. Averages to the error rate. Not smooth, so hard to optimise directly.
Cross-entropy (log loss)	$-[y \ln p + (1 - y) \ln(1 - p)]$	Uses the predicted probability $p = P(Y = 1)$. Confident and right: near 0. Confident and wrong: huge. Fits logistic regression (4.5).
Cost-weighted	$c_{FP} \cdot FP + c_{FN} \cdot FN$	Mistakes priced differently. Used to choose thresholds (4.9).
Hinge	$\max\{0, 1 - yf(x)\}$	Used by SVMs (6.6). No loss once a point is correct and outside the margin.

WORKED EXAMPLE: CROSS-ENTROPY BY HAND

True label $y = 1$. Model A says $p = 0.8$; model B says $p = 0.2$.

$L_A = -\ln 0.8 = 0.223$. $L_B = -\ln 0.2 = 1.609$. B is punished about 7 times more because it gave low probability to what actually happened.

If instead $y = 0$ and $p = 0.8$: $L = -\ln(1 - 0.8) = 1.609$. Being confidently wrong is costly in either direction.

Loss vs evaluation metric

The **loss** is what the algorithm minimises while fitting; it has to be mathematically convenient. The **metric** is how you judge the final model for the real task. They can differ: a logistic regression is fitted with cross-entropy but may be judged by AUC, or by expected cost at a chosen threshold.

TYPICAL QUESTIONS

"Why is squared loss a poor choice when the data contain a few extreme outliers?" It squares large errors, so a single outlier can dominate the fit; absolute loss grows only linearly.

"Why not fit logistic regression by minimising squared error on 0/1 labels?" Cross-entropy is the maximum-likelihood loss for a yes/no outcome; it punishes confident wrong probabilities heavily and gives a well-behaved optimisation problem.

Common doubts

FIRST PASS **What is the difference between a loss, the risk and the objective?**

- **Loss** $L\big(y_i, f(x_i)\big)$: the error on **one** example.
- **Empirical risk** $\hat{R}_n(f) = \frac{1}{n}\sum_i L\big(y_i, f(x_i)\big)$: the average loss over the training data.
- **Objective**: whatever training actually minimises. That is the empirical risk, plus a penalty if you regularise: $\hat{R}_n(f) + \lambda\,\Omega(f)$.

Books and software often use "loss" for all three. In an exam, say which one you mean.

GOING DEEPER **Does the objective represent the error?**

Mostly. With no penalty, the objective is the average error on the **training** data. Two caveats:

- With a penalty (ridge, LASSO, the SVM's C), it is error plus a cost for complexity, so a lower objective doesn't always mean a lower training error.
- It is training error, not the error on new data. That is the population risk, which you estimate on held-out data (lesson 1.5).

Cheatsheet

Squared loss (L2)

$(y-\hat{y})^2$. Punishes big errors; outlier-sensitive; OLS, ridge, LASSO.

Absolute loss (L1)

$|y-\hat{y}|$. Linear; robust to outliers; MAE.

Cross-entropy / log loss

$-[y\ln p + (1-y)\ln(1-p)]$. Logistic regression. Confidently wrong = huge loss.

0–1 loss

$1[y \neq \hat{y}]$. Averages to the error rate; not smooth.

Loss vs metric

Loss: minimised during fitting (must be convenient). **Metric**: judges the final model for the task (AUC, cost, RMSE).

Online version with the interactive quiz: airesearchlabs.si/learn/ml/statistical-learning/loss-functions-and-risk

Empirical Risk Minimisation (ERM)

ERM chooses the model in a family with the lowest training loss. OLS and logistic regression are both ERM. Why minimising training loss is necessary but not sufficient.

The recipe

We can't minimise population risk because we can't compute it. So we do the next best thing: pick a **family** of candidate functions $\mathcal{F}$ (for example "all straight lines") and choose the member with the smallest average training loss.

EMPIRICAL RISK MINIMISATION

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \hat{R}_n(f) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L\big(y_i, f(x_i)\big)$$

"$\arg\min$" means "the f that achieves the minimum", not the minimum value itself.

Familiar methods are all ERM

METHOD	FAMILY $\mathcal{F}$	LOSS L
Ordinary least squares	Linear functions $\beta_0 + \beta^\top x$	Squared loss
Logistic regression	Sigmoid of a linear score	Cross-entropy
Linear SVM	Linear scores $w^\top x + b$	Hinge loss (+ penalty)
Ridge / LASSO	Linear functions	Squared loss + penalty on β

The last two rows add a **penalty** for complexity: that is *regularised* ERM, the subject of Section 5.

Two choices control what ERM finds

- **The family $\mathcal{F}$:** too narrow, and even the best member is poor (underfitting). Too rich, and some member can fit the training noise exactly (overfitting).
- **The loss L:** determines what "best" means.

MINIMISING TRAINING LOSS ≠ MINIMISING TRUE LOSS

$\hat{f}$ minimises $\hat{R}_n$, not R. A very flexible family can drive training loss to zero while population risk stays large. That's why ERM is always paired with an honest estimate of performance on data not used for fitting.

HOW IT'S ASKED

"OLS is empirical risk minimisation with squared loss. Explain in plain words." Among all straight lines (or planes), OLS picks the one whose average squared vertical distance to the training points is smallest.

Common doubts

FIRST PASS **Is minimising the objective the same thing as ERM?**

Without a penalty, yes: the objective **is** the empirical risk, and ERM is the principle of choosing the model that minimises it. With a penalty it is **regularised ERM**: minimise $\hat{R}_n(f) + \lambda\,\Omega(f)$. Ridge, LASSO and the soft-margin SVM are all regularised ERM.

FIRST PASS **What is the difference between ordinary ERM and regularised ERM?**

Ordinary ERM trusts the training data completely: it picks whatever fits it best. With a flexible model, that includes fitting the noise. **Regularised ERM** adds a penalty on complexity, $\lambda\,\Omega(f)$, and gives up a little training error for a simpler model that generalises better. λ sets the trade-off and is chosen by cross-validation (Section 5).

GOING DEEPER **How do loss, empirical risk, ERM and optimisation fit together?**

1. Choose a **loss**: how one wrong prediction is scored.
2. Average it over the training set: the **empirical risk**.
3. **ERM** says: pick the model in your family $\mathcal{F}$ with the lowest empirical risk (plus a penalty, if regularised).
4. **Optimisation** is how you actually find it. OLS has a formula, the normal equations. Logistic regression has none, so it is solved step by step with gradient descent or Newton's method.
5. Then check the result on held-out data, because a low empirical risk doesn't guarantee a low population risk.

Cheatsheet

ERM

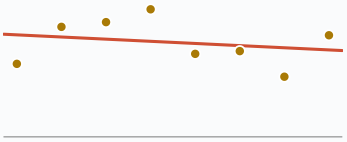
$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_i L(y_i, f(x_i))$. OLS = ERM + squared loss; logistic = ERM + cross-entropy.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/statistical-learning/empirical-risk-minimisation

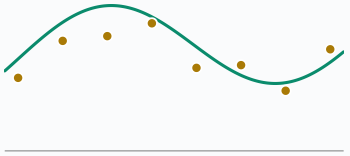
Underfitting and Overfitting

Too simple and a model misses the pattern; too flexible and it memorises noise. How to recognise each from training and validation errors, and what to do about it.

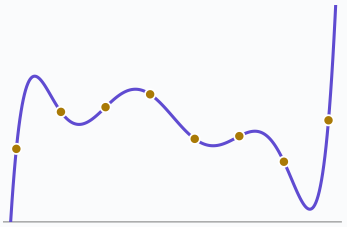
Three fits to the same data



Underfit: a straight line misses the curve.



Good fit: follows the real pattern, ignores the noise.



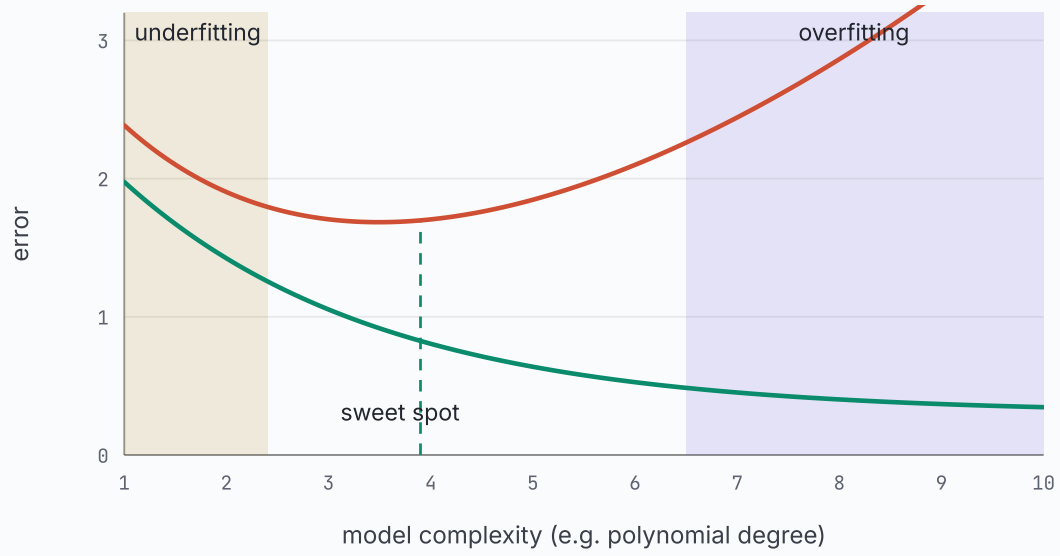
Overfit: a degree-7 polynomial hits every point and wiggles wildly between them.

The overfit curve has **zero training error**: it passes through every point. Yet between and beyond the points it is wild, so its predictions for new data are poor.

The diagnosis table

	TRAINING ERROR	ERROR ON UNSEEN DATA	STORY
Underfitting	High	High	Model too simple to capture the real pattern (high bias)
Good fit	Low-ish	Similar, low	Captures repeatable structure
Overfitting	Very low	Much higher	Fits noise and quirks of this sample (high variance)

The U-shaped curve



As complexity grows, training error only goes down. Error on new data first falls, then rises. Choosing complexity, or later the penalty λ, means choosing a point on this axis.

What to do

IF YOU SEE...	TRY
Underfitting	A more flexible model, better features (e.g. X^2, interactions), less regularisation
Overfitting	More training data, a simpler model, fewer features, regularisation (Section 5), cross-validation to tune complexity

TRAINING ERROR CAN'T CHOOSE COMPLEXITY

Training error always prefers the most complex model: a degree-10 polynomial always fits the training data at least as well as a degree-1 line. Use validation data or cross-validation instead.

Common doubts

GOING DEEPER **How do I read training, validation and test results together?**

- **Training high, validation low** → possible overfitting.
- **Training and validation high, test low** → possible distribution shift, a preprocessing inconsistency, or another evaluation problem (see lesson 2.4).
- **All three low** → possible underfitting.

"High" and "low" here refer to performance (accuracy), not error.

Cheatsheet

Diagnose from errors

Train low, val high → **overfit**. Both high → **underfit**. Train & val good, test bad → shift / leakage / evaluation issue.

Training error picks complexity

Wrong: choose the model with the lowest training error. Right: choose on validation data or by CV.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/statistical-learning/underfitting-and-overfitting

Generalisation and the Generalisation Gap

Generalisation is performing well on new data. The generalisation gap is how much worse a model does there than on its training data. What makes the gap grow or shrink, and what to do when test scores disappoint.

Definition

Generalisation is the ability of a fitted model to perform well on new observations from the same population. The **generalisation gap** measures how far training performance overstates real performance:

$$\text{gap} = R(f) - \hat{R}_n(f) = (\text{true risk}) - (\text{training risk})$$

The true gap can't be computed, because $R(f)$ needs the unknown population. We estimate it by comparing training error with error on held-out data.

What affects the gap

FACTOR	EFFECT ON THE GAP
Model complexity / number of parameters	More flexible → **bigger** gap
Training-set size n	More representative data → **smaller** gap (it tends to 0 as $n \to \infty$)
Regularisation / algorithmic stability	Constraining the model → **smaller** gap
Noise in the data	Noisier → easier to "fit the noise" → **bigger** gap



A learning curve: with more data, training error rises slightly (harder to memorise) and validation error falls. The gap closes.

When the test score disappoints

SCENARIO

Validation accuracy is 96%, but the final test accuracy is only 79%. What now?

Possible causes: overfitting to the validation set through many rounds of selection; **distribution shift** (the test data differ from training and validation, e.g. collected later or elsewhere); preprocessing applied inconsistently to the test set; or an evaluation bug.

Actions: investigate the cause; improve the model using only training and validation data; then get a **fresh, independent test set** for a final honest estimate. Don't keep tuning against the same test set: that would leak it (lesson 2.8).

BIG IDEA

A small gap is good news only if both errors are low. A model can have a tiny gap and still be useless (underfitting: both errors high).

Common doubts

GOING DEEPER **Validation accuracy is 96% but test accuracy is 79%. What should I do?**

First, investigate rather than tweak. Likely causes are **overfitting** (often to the validation set itself, after trying many models), **distribution shift** between the validation and test data, **preprocessing inconsistencies** (e.g. the test set was scaled differently), or an **evaluation problem**.

Then improve the model using only the training and validation data, and evaluate the final version on a **fresh independent test set**. If you keep adjusting until the old test score goes up, that test set has become a validation set and its score is no longer an honest estimate.

Cheatsheet

Generalisation gap drivers

↑ complexity → ↑ gap. ↑ data → ↓ gap. ↑ regularisation → ↓ gap. ↑ noise → ↑ gap.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/statistical-learning/generalisation-and-generalisation-gap

The Bias–Variance Trade-off

Expected prediction error splits into bias², variance and irreducible noise. Why flexible models trade bias for variance, illustrated with the thought experiment of many training sets.

The thought experiment

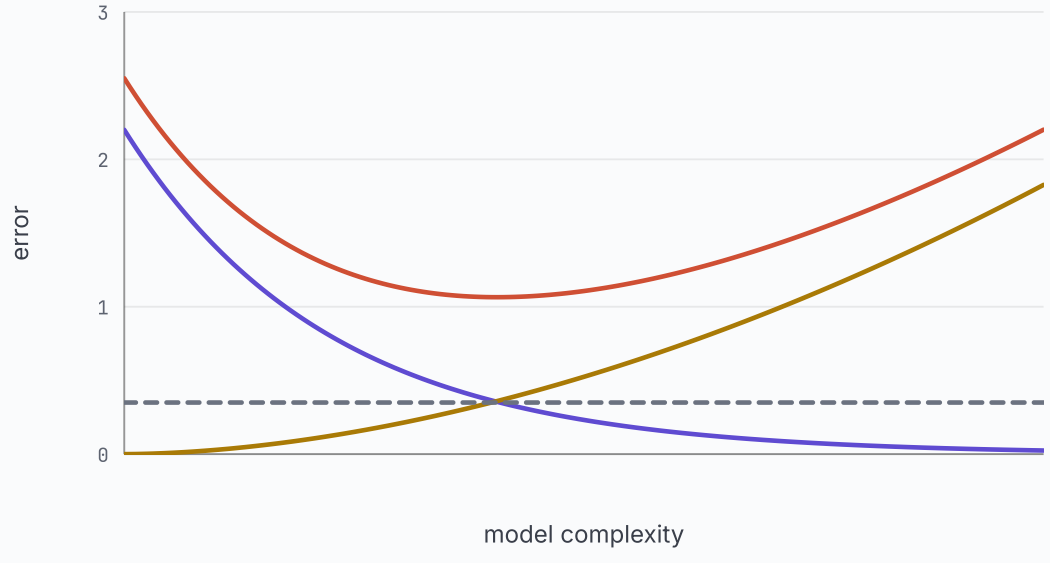
Imagine drawing many different training sets from the same population and fitting the same type of model to each. At one input x_0 you'd get a spread of predictions $\hat{f}(x_0)$. Three things determine how wrong they are:

- **Bias:** how far the *average* prediction (over all those training sets) is from the truth. Systematic error from an overly simple model.
- **Variance:** how much the prediction jumps around from one training set to another.
- **Irreducible noise:** randomness in Y itself (ε, with variance σ^2) that no model can predict.

SQUARED-ERROR DECOMPOSITION AT A FIXED INPUT X_0

$$\mathbb{E}\big[(Y-\hat{f}(x_0))^2\big] = \underbrace{\big(\mathbb{E}[\hat{f}(x_0)]-f(x_0)\big)^2}_{\text{bias}^2} + \underbrace{\operatorname{Var}\big(\hat{f}(x_0)\big)}_{\text{variance}} + \underbrace{\sigma^2}_{\text{noise}}$$

Bias² and variance are the **reducible** error. σ^2 can't be reduced by any choice of model.



As complexity rises, bias² falls and variance rises. Total error is U-shaped; the noise floor never moves.

Darts

	LOW VARIANCE	HIGH VARIANCE
Low bias	Tight cluster on the bullseye (ideal)	Scattered around the bullseye
High bias	Tight cluster, consistently off-centre	Scattered and off-centre (worst)

Simple vs flexible models

	SIMPLE (STRAIGHT LINE)	FLEXIBLE (DEGREE-11 POLYNOMIAL)
Bias	High: can't represent curvature	Low
Variance	Low: stable across samples	High: chases noise
Typical failure	Underfitting	Overfitting

Practical implications

- With limited or noisy data, a simpler model is often better.
- A flexible model can capture a genuinely complex relationship, but needs enough data and careful validation.
- Regularisation (Section 5) adds a little bias to remove a lot of variance. More data reduces variance; it does nothing to σ^2.

COMMON SLIPS

"Variance" here means variance of the *fitted model* across training sets, not the variance of the data. And the goal is the lowest *total* error, not minimising bias or variance separately.

Cheatsheet

Bias–variance decomposition

$\mathbb{E}[(Y - \hat{f})^2] = \text{bias}^2 + \text{Var}(\hat{f}) + \sigma^2$.

Simple vs flexible

Simple: high bias, low variance, underfits. Flexible: low bias, high variance, overfits.

"Variance" means...

Variance of the **fitted model** across training sets, not the variance of the data.

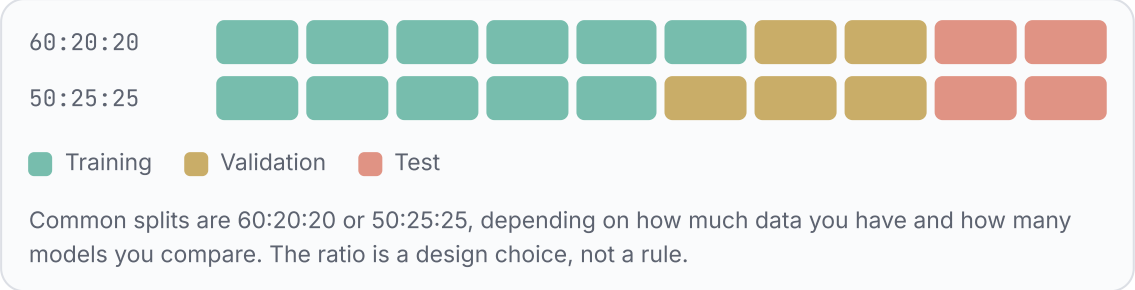
Online version with the interactive quiz: airesearchlabs.si/learn/ml/statistical-learning/bias-variance-trade-off

Training, Validation and Test Sets

What each data split is for, typical split sizes, and why repeatedly checking the test set and picking the best model quietly ruins your performance estimate.

Three sets, three jobs

SET	USED FOR	RULE
Training	Fitting the parameters (e.g. $\hat{\beta}$)	Never the sole evidence of quality
Validation	Comparing models, tuning hyperparameters (degree, λ, K, threshold)	Can be reused, but each reuse slightly overfits it
Test	One final, honest estimate of performance on new data	Touch it **once**, after all decisions are made



Common splits are 60:20:20 or 50:25:25, depending on how much data you have and how many models you compare. The ratio is a design choice, not a rule.

With $n = 1{,}000$ and a 60:20:20 split you get 600 training, 200 validation and 200 test observations. After choosing, the final model is often re-trained on training + validation data.

> THE SINGLE MOST IMPORTANT RULE OF ML EVALUATION
>
> You only get an unbiased estimate of performance by evaluating on data that played no role whatsoever in fitting *or choosing* the model.

Why picking the best test score is a leak

SCENARIO: 40 MODELS

A bank has 10,000 records: 7,000 training, 1,500 validation, 1,500 test. A data scientist trains 40 models and checks each one's **test** accuracy: 85%, 87%, 89%, ..., up to 96% for model 40. They select model 40 and report 96%.

No model was trained on test rows, so where's the problem? The **choice** used the test set. Among 40 candidates, the winner partly won by luck on those particular 1,500 rows. That's selection bias: the reported 96% is too optimistic.

Correct: train on the training set, compare on the validation set (or by cross-validation), select the best, then evaluate that one model on the test set once.

Stratified splitting

When one class is rare (say 2% fraud), a random split can leave very few positives in a small validation or test set. A **stratified** split keeps the class proportions the same in every subset.

Common doubts

GOING DEEPER **Who actually "learns" the test set: the data scientist or the model?**

The data scientist, or any automated selection procedure. Each of the 40 models is trained from scratch on the training data; none of them sees test rows or remembers earlier models.

But the **person** looks at test results and decides what to try next and which model to keep. Through those decisions, the overall development process adapts to the test set. Leakage happens through human (or automated) model-selection decisions, not through the training of any single model.

GOING DEEPER **Why is choosing model 40 because it has the highest test accuracy a problem?**

Repeatedly checking test results influences every later development decision, so the selection process indirectly fits the test set. With 40 tries, some models score well on that particular test set by chance. Picking the maximum keeps that luck, so the final reported test accuracy is **overly optimistic**: expect less on genuinely new data.

Cheatsheet

Train / validation / test

Train: fit parameters. **Validation**: choose models & hyperparameters. **Test**: final estimate, used **once**.

Typical splits

60:20:20 or 50:25:25 (train:val:test); train:val 50–80% / 20–50%. A design choice.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/statistical-learning/training-validation-and-test-sets

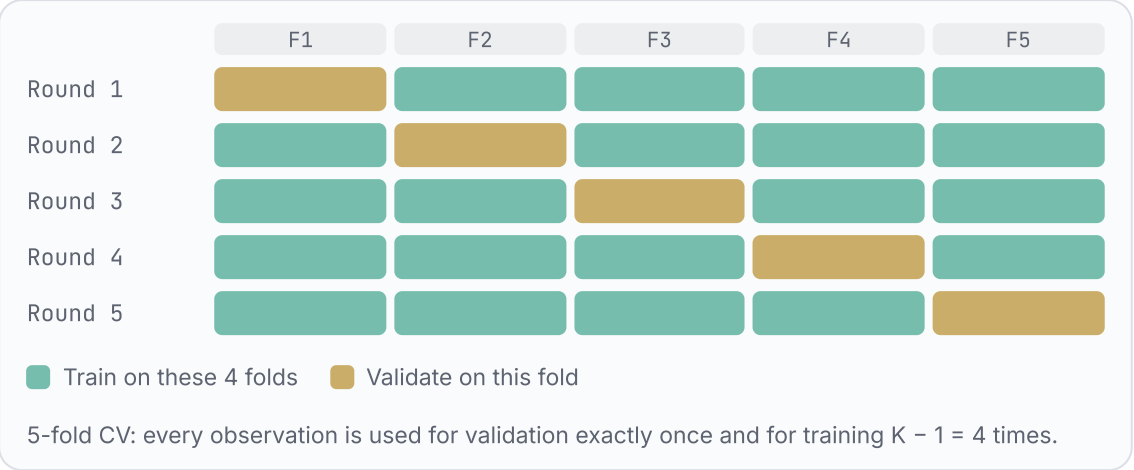
Cross-validation

K-fold cross-validation step by step, a worked comparison of two models, why each fold is an independent experiment, and when ordinary random folds are the wrong tool.

Why not just one validation split?

A single split has two weaknesses: unless data are plentiful you end up with too few training points *or* too few validation points, and the result depends on which rows happened to land where. Cross-validation reuses the data through several train/validation partitions.

K-fold cross-validation



5-fold CV: every observation is used for validation exactly once and for training K − 1 = 4 times.

1. Randomly split the data into K roughly equal folds (typically $K = 5$ or 10).
2. For each fold $i = 1, \dots, K$: train on the other $K - 1$ folds and compute the error on fold i.
3. Average the K errors: $\mathrm{CV} = \frac{1}{K} \sum_{i=1}^{K} \mathrm{err}_i$.
4. Do this for each candidate model (or each value of a hyperparameter) and pick the best average. Re-train the chosen model on all the training data.

Cost: you fit the model K times per candidate. Training on $\frac{K-1}{K}$ of the data is nearly as good as training on all of it.

WORKED EXAMPLE: CHOOSING BETWEEN TWO MODELS WITH 5-FOLD CV

VALIDATION RMSE	F1	F2	F3	F4	F5	MEAN
Model A (simple)	2.10	2.40	2.20	2.30	2.10	**2.22**
Model B (flexible)	1.50	3.20	1.60	2.90	1.80	**2.20**

B is marginally better on average, but its fold errors range from 1.5 to 3.2: a sign of instability (high variance). A sensible analyst might prefer A. **Look at the spread, not just the mean.**

Each fold is an independent experiment

A patient can be in the validation fold in round 1 and in the training folds in round 2. That's fine: each round is a fresh fit from scratch, and nothing learned in round 1 is carried into round 2. Anything that learns from data (scaling, feature selection, oversampling) must be redone **inside each round**, using that round's training folds only.

CV is not the final test

CV chooses models and hyperparameters using the training data. A separate untouched test set still gives the final honest number. And folds must respect the structure of the data:

- **Time series:** random folds train on the future. Use forward-chaining folds instead (2.8).
- **Repeated measurements on a person:** keep all of a person's rows in the same fold (group K-fold).

Common doubts

GOING DEEPER **When moving to the second fold, the previous validation fold becomes training data. Aren't we influencing the outcome?**

No. Each fold is an **independent** training–validation experiment. The model is re-fitted from scratch each round, and feature selection (or any other learned preprocessing) is repeated using only that round's training folds. Being validation data in one round and training data in another is exactly how CV is designed, and it isn't leakage by itself.

What *can* bias the result is repeatedly changing your whole methodology based on CV scores. Then the CV estimate itself becomes optimistic, which is why a final untouched test set is still valuable.

GOING DEEPER **If feature selection happens inside each fold, why would the selected features differ between folds?**

Because each round sees a different training subset, so the "best 20 genes" are ranked on slightly different data. In the wrong approach, genes are selected once using all the data, so every fold uses the same genes. In the correct approach they're selected independently within each fold, so they *can* differ, though they may also turn out identical.

Cheatsheet

K-fold CV

1) Split into K folds. 2) For each fold: train on $K-1$, validate on it. 3) Average the K errors. 4) Pick the best candidate; re-fit on all training data.

CV with preprocessing

Scaling, imputation, feature selection and oversampling happen **inside each fold**, on that fold's training part only.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/statistical-learning/cross-validation

Data Leakage and Good ML Practices

The six common ways information leaks into model evaluation: test-set reuse, preprocessing before splitting, feature selection before CV, related records across splits, time leakage and target leakage. Each with a scenario, why it is wrong and the fix.

THE ONE PRINCIPLE

A model may only learn from information that would genuinely be available when it makes real predictions. Anything else makes measured performance look better than real performance.

Data leakage is when information that shouldn't be available during model development or prediction influences the model-building process. The model isn't necessarily trained on test rows. The leak can be a statistic, a choice or a feature.

The six leaks at a glance

#	LEAK	WHAT GOES WRONG	FIX
1	Test set reused for selection	Picking the model with the best test score	Select with validation data or CV; test once
2	Preprocessing before splitting	Scaling/imputation statistics computed on all data	Split first; fit preprocessing on training data only
3	Feature selection before CV	Features chosen using every row's label	Select features inside each training fold
4	Related records across splits	Same patient or customer in train and test	Group-based splitting
5	Time leakage	The future influences predictions about the past	Chronological split; forward-chaining validation
6	Target leakage	A feature reveals the outcome, or exists only after it	Use only features available at prediction time

Leak 1: reusing the test set for model selection

Covered in lesson 2.6: trying 40 models and keeping the best *test* score makes the reported score optimistic, because the selection adapted to that test set. **Takeaway:** the test set must stay independent of every development decision.

Leak 2: preprocessing before splitting

Preprocessing turns raw data into model-ready form: standardisation, normalisation, missing-value imputation, category encoding, PCA. The leak appears when its parameters (means, medians, category lists, components) are learned from the whole dataset before splitting.

Why standardise at all?

Features on very different scales (house size in sq ft, number of bedrooms) can distort distance-based methods and penalised models. **Standardisation** puts each feature on a comparable scale:

$$z = \frac{x - \mu}{\sigma}$$

μ (the mean) is the centre of the distribution; σ (the standard deviation) measures how spread out values are. Two neighbourhoods can have the same average house size, 1,500 sq ft, but in one every house is 1,400–1,600 (small σ) while the other ranges from 600 to 3,000 (large σ). A z of +1 means "one typical spread above average".

SCENARIO: HOUSE PRICES

1,000 houses. Wrong: compute $\mu = 1{,}500$, $\sigma = 500$ on all 1,000, then split. The validation and test rows have shaped μ and σ.

Right: split first (700 / 150 / 150). Compute the statistics from the 700 training rows only, say $\mu = 1{,}400$, $\sigma = 400$. Apply **those same numbers** to training, validation and test. Don't recompute statistics on the validation or test sets.

WHY NOT SEPARATE STATISTICS FOR THE TEST SET?

Take a 1,800 sq ft house. With the training statistics: $z = (1800 - 1400)/400 = +1.0$. If the test set were standardised with its own statistics, say $\mu = 2{,}000$, $\sigma = 500$: $z = (1800 - 2000)/500 = -0.4$.

The same house now looks *below* average instead of above. The model learned what $z = +1$ means on the training scale; changing the ruler for the test set breaks that, and it hides genuine differences between the test and training distributions.

The same applies to imputing missing values with a mean or median, min–max scaling, PCA and learning category mappings: **fit on training data, apply everywhere**.

Leak 3: feature selection before cross-validation

SCENARIO: CANCER FROM GENE MEASUREMENTS

200 patients, 100 gene measurements, outcome cancer (1) or not (0). The data scientist uses all 200 patients to find the 20 genes most associated with cancer, keeps those, then runs 10-fold CV and reports 3% error.

Why it's wrong: the labels of patients who later sit in validation folds already helped choose the genes. With many candidate features, some genes correlate with the label purely by chance; picking them using all labels produces artificially strong CV results. Even if every gene were pure noise, this procedure could report a low error.

Right: for every fold, take the 9 training folds, select the 20 best genes using **only those**, train, and evaluate on the held-out fold. Each round may pick different genes. In practice use a pipeline that does selection and training together inside each fold.

A tiny version you can check by hand

Six patients, three genes; we keep the single gene that agrees with the label most often. Patient P6 is the held-out validation patient.

PATIENT	G1	G2	G3	LABEL Y
P1	1	1	0	1
P2	1	0	1	1
P3	0	1	1	1
P4	0	0	1	0
P5	0	0	0	0
P6 (held out)	1	0	1	0

- **Correct** (P1–P5 only): G1 agrees 4/5, G2 agrees 4/5, G3 agrees 3/5. Tie; the rule takes the first, G1. G1 predicts P6 = 1, which is **wrong**.
- **Wrong** (all six): G1 4/6, G2 **5/6**, G3 3/6. G2 wins and predicts P6 = 0, which is **right**.

The wrong approach "did better", but only because P6's own label broke the tie in G2's favour. The validation result was decided by the answer it was supposed to be predicting. Across many folds and genes, that inflates the score.

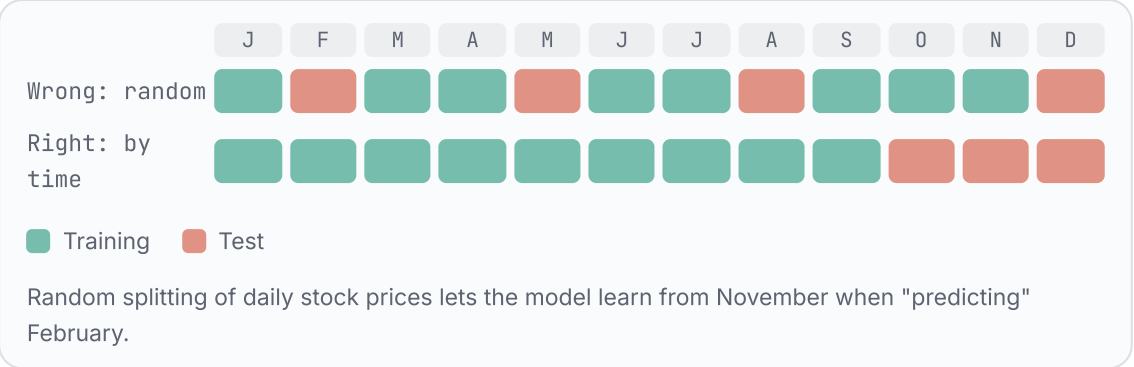
Leak 4: related observations across splits

100 patients each have 5 blood-test records (500 rows). Splitting the rows at random puts records of the same patient in both training and test. Patients have individual patterns (their typical blood pressure, history), so the model can learn *who the patient is* rather than a general medical pattern, and

score unusually well on another record of someone it has already seen. On completely new patients it does worse.

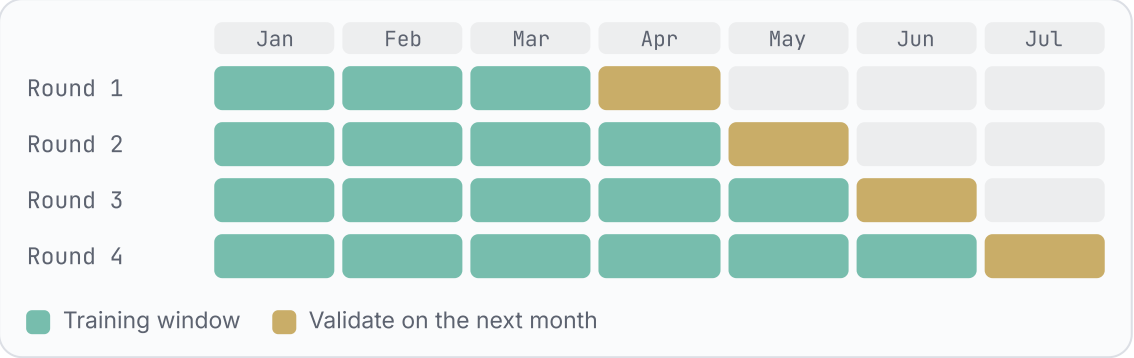
Fix: split by patient ID so all of a patient's records stay together (group split; in scikit-learn, `GroupKFold` and `GroupShuffleSplit`). The same applies to many transactions per customer, purchases per shopper, or measurements per machine.

Leak 5: time leakage



Random splitting of daily stock prices lets the model learn from November when "predicting" February.

Evaluated on 10 February, a randomly-split model has already trained on November and December data, which wouldn't exist yet in a real forecast. **Fix:** split chronologically, and validate with **forward chaining**, always predicting the period just after the training window:



Leak 6: target leakage

A feature directly or indirectly reveals the outcome, often because it only becomes known *after* the event being predicted. The model looks superb in testing and is useless in real use.

> SCENARIO: LOAN DEFAULTS WITH 99% ACCURACY
>
> Features: age, salary, credit score, loan amount, employment status, and **collections status** (whether the loan was sent to debt collection). Follow the timeline: (1) customer applies; (2) bank records salary, age, credit score; (3) bank wants to predict default; (4) customer defaults; (5) loan goes to collections. Collections status only exists after step 4, but the prediction must be made at step 3. Drop it.

Similarly, when predicting 30-day hospital readmission, age, history and blood pressure at discharge are fine; a feature recording "readmitted within 30 days" is the outcome itself.

Answering "is this approach reasonable?"

1. **Identify the problem:** "Mean and SD are computed on the entire dataset before splitting."
2. **Name the leak:** "Test-set information has influenced preprocessing."
3. **Explain the consequence:** "The evaluation is over-optimistic."
4. **Recommend the fix:** "Split first, fit preprocessing on training data only, apply the fitted transformation to validation and test."

> ONE TEST FOR EVERY LEAK
>
> Ask: did anything from the held-out data, from the future, or from the outcome influence a calculation or a decision before the final score? If yes, there is leakage.

Spot the leak: practice

SCENARIO	VERDICT
Impute missing blood-test values with the whole dataset's median, then split 80/20	Leak 2. Compute the median on training data only.
Predicting "will cancel next month" using "calls to the cancellation hotline this month"	Leak 6. A consequence of the intent to cancel; not available in time.
Keep the 50 features most correlated with the label (all rows), then report 5-fold CV	Leak 3. Select inside each fold.
Weekly sales for 3 years, random 80/20 split, to forecast next quarter	Leak 5. Split chronologically.
Standardise on training only, choose among 12 models by CV on training, test once	**No leak.**
Three photos per patient, photos split randomly	Leak 4. Split by patient.

Common doubts

FIRST PASS **Why is repeatedly checking test scores and choosing the best model a form of leakage?**

Because the test set influenced a decision. Even though no model trained on test rows, the choice of model adapted to that particular test set, and the winner partly won by chance. Its test score is therefore optimistic. Select on validation data or CV and use the test set once at the end.

FIRST PASS **Why is calculating the mean and standard deviation on the entire dataset a problem?**

Those statistics then include the validation and test rows, so information about the held-out distribution enters the pipeline before evaluation. Split first, compute the statistics on the training set only, and reuse them to transform validation and test data.

FIRST PASS **Why does selecting features using the full dataset leak into the validation folds?**

Choosing features "most associated with the label" uses every row's label, including rows that later serve as validation. So the validation labels helped decide which features the model uses, and the CV score is inflated, especially when there are many candidate features that correlate with the label by chance.

FIRST PASS **Why must all records of the same patient or customer stay in the same split?**

Records of one person are similar to each other. If they appear in both training and test, the model can recognise the person instead of learning a general pattern, which inflates the test score. Real deployment involves new people, so split by person (group split).

FIRST PASS **Why can randomly splitting time-series data give unrealistic results?**

A random split lets the model train on later dates and be tested on earlier ones, i.e. learn from the future. In real forecasting the future isn't available. Use a chronological split and forward-chaining validation.

FIRST PASS **Why can't we use features that become available after the prediction time?**

At the moment the real prediction is made, those features don't exist yet. A model that relies on them looks excellent in testing (they effectively reveal the answer) and fails in practice. Use only information available at the intended prediction time.

GOING DEEPER **If we follow the train–validation–test workflow correctly, is all leakage eliminated?**

No. The workflow prevents test-set reuse, but leakage has several forms:

TYPE	MAIN PROBLEM
Model-selection leakage	Test results influence model selection
Preprocessing leakage	Validation/test information influences preprocessing
Feature-selection leakage	Validation labels influence feature selection
Target leakage	Future information reveals the target
Duplicate leakage	Related records appear across datasets
Temporal leakage	Future information influences past predictions

A correct split is necessary, not sufficient.

GOING DEEPER **Should training, validation and test sets each get their own mean and standard deviation?**

No. Calculate the mean and SD on the **training data only**, and reuse those exact values for validation and test. Recomputing them per set changes the reference scale: the same 1,800 sq ft house would be $z = +1$ with training statistics but $z = -0.4$ with test statistics, and genuine differences between the sets get hidden.

GOING DEEPER **With correct feature selection, do we still use all 100 genes in the validation fold?**

All 100 measurements are *available*, but the selection uses only the training fold: pick the best 20 genes from the training fold, then use **those same 20 genes** for that fold's validation data. Validation labels never influence which genes are chosen.

GOING DEEPER **If both approaches end up picking the same genes, how does it matter whether selection happens before or after CV?**

Sometimes they do pick the same genes, and then the scores coincide. The problem is the cases where they don't: in the six-patient example, the held-out patient's label was the only reason G2 beat G1. The wrong approach can even produce a better individual prediction, but that doesn't make the method valid. The goal of evaluation is a **trustworthy performance estimate**, not making each prediction right by peeking.

GOING DEEPER **In loan-default prediction, is the problem that a new customer has never taken a loan, so we can't predict their default?**

That's a different problem, usually called limited or thin credit history. **Target leakage** is about using information that only becomes available after the prediction point. Collections status is known too late to predict default at application time. The model must use only information available when the prediction is made.

Cheatsheet

The six leaks

1 Test reuse · 2 Preprocess before split · 3 Feature selection before CV · 4 Related records across splits · 5 Time leakage · 6 Target leakage.

Six golden rules

Split before preprocessing. Never tune on test. Select features inside folds. Keep related rows together. Respect time order. Use only prediction-time features.

One test for every leak

Did anything from held-out data, the future, or the outcome influence a calculation or decision before the final score? If yes → leak.

Standardisation

$z = \frac{x-\mu}{\sigma}$ with μ, σ from the **training set only**, reused for validation and test.

"Is this approach reasonable?"

1) Identify the problem. 2) Name the leak. 3) Explain the consequence (optimistic estimate). 4) Recommend the fix.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/statistical-learning/data-leakage-and-good-ml-practices