AI Research Labs

Complete course · 49 lessons

Machine learning, *properly.*

From what a model is actually minimising to ridge, LASSO, K-means and support vector machines. Each lesson builds the intuition first, then the maths, then a worked example you can do with a pen and a calculator.

1 Machine Learning Foundations
2 Statistical Learning and Model Evaluation
3 Linear Regression
4 Logistic Regression and Classification
5 Regularisation and Model Selection
6 Clustering and Support Vector Machines



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.

Contents

Section 1 · Machine Learning Foundations

1.1 What is Machine Learning?
1.2 Types of Machine Learning
1.3 The ML Modelling Process
1.4 Fundamental Limits of Machine Learning
1.5 Empirical Risk and Population Risk

Section 2 · Statistical Learning and Model Evaluation

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

Section 3 · Linear Regression

3.1 Regression Fundamentals
3.2 Simple and Multiple Linear Regression
3.3 Ordinary Least Squares (OLS)
3.4 Calculating Regression Coefficients by Hand
3.5 Interpreting Coefficients
3.6 Residuals and Regression Assumptions
3.7 R^2, Adjusted R^2, RMSE and Other Metrics
3.8 Standard Errors, Confidence Intervals and p-values
3.9 Multicollinearity and VIF

Section 4 · Logistic Regression and Classification

4.1 Regression versus Classification
4.2 Odds, Log-odds and the Sigmoid Function
4.3 The Logistic Regression Model
4.4 Maximum Likelihood Estimation
4.5 Cross-entropy / Log-loss
4.6 The Confusion Matrix
4.7 Accuracy, Precision, Recall and Specificity
4.8 Sensitivity and Specificity Trade-offs
4.9 Threshold Selection and Classification Costs
4.10 ROC Curve and AUC
4.11 Class Imbalance

Section 5 · Regularisation and Model Selection

5.1 Why Regularisation is Needed
5.2 Model Complexity and Feature Selection
5.3 Best Subset and Stepwise Selection
5.4 Ridge Regression (L2)
5.5 LASSO Regression (L1)
5.6 Understanding λ (Lambda)
5.7 Ridge versus LASSO
5.8 Mathematical Objectives and Worked Calculations
5.9 Selecting λ Using Cross-validation

Section 6 · Clustering and Support Vector Machines

6.1 K-means Clustering
6.2 Distance Calculations and Cluster Assignment
6.3 Updating Centroids and Convergence
6.4 SVM Fundamentals
6.5 Margins and Support Vectors
6.6 Hard-margin and Soft-margin SVM
6.7 The C Parameter and Kernels

Section 1

Machine Learning Foundations

What machine learning does, how it differs from traditional programming, how models learn from data, and why learning from a finite dataset has limits.

1.1 What is Machine Learning? · 6 min

1.2 Types of Machine Learning · 7 min

1.3 The ML Modelling Process · 7 min

1.4 Fundamental Limits of Machine Learning · 6 min

1.5 Empirical Risk and Population Risk · 8 min

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.

1

SUPERVISED LEARNING · STEP 1

Frame the problem

What exactly are we predicting, and what decision will it drive?

Lessons here: 1.1, 1.2, 1.3 · feeds into Get, split and prepare the data

◆

UNDERNEATH BOTH MAPS · FOUNDATION

No free lunch

Lessons here: 1.4 · feeds into Choose the model, Choose the model, Evaluate honestly

◆

UNDERNEATH BOTH MAPS · FOUNDATION

Empirical vs population risk

Lessons here: 1.5 · feeds into Evaluate honestly

What is Machine Learning?

Machine learning is learning a prediction rule from data instead of writing it by hand. What that means, how it differs from traditional programming, and why the decision matters more than the prediction.

The one-sentence version

Machine learning (ML) is about making predictions and decisions from data. Instead of a programmer writing the rule, an algorithm *learns* the rule from examples. The term was coined by Arthur Samuel in 1959, whose checkers program learned to play better than he did by adjusting its own scoring rule from experience.

Traditional programming vs machine learning

In traditional programming you write the rules and the computer applies them. In machine learning you supply examples of inputs and the right outputs, and the algorithm finds a rule that maps one to the other.

	TRADITIONAL PROGRAMMING	MACHINE LEARNING
You provide	Rules + data	Data + answers (examples)
Computer produces	Answers	Rules (a fitted model)
Spam filter example	"If the subject contains FREE!!! then spam"	Learn which word patterns separate 100,000 labelled spam and non-spam emails
Breaks when	The world changes faster than you can update the rules	New data stops looking like the training data

ML shines when the rule is too complicated to write down (recognising a face), changes over time (fraud patterns), or must be personalised (recommendations).

A model is a function

Whatever the method, the output of learning is a function f that takes inputs X (also called **features** or **predictors**) and returns a prediction $\hat{y} = f(X)$ of the **target** Y (also called the response, outcome or label). The hat on $\hat{y}$ always means "estimated" or "predicted".

$$\underbrace{X}_{\text{inputs}} \longrightarrow \boxed{f} \longrightarrow \underbrace{\hat{y}}_{\text{prediction}}$$

ML is not one technique. It is a toolbox of models (linear regression, logistic regression, SVMs, K-means, trees, neural networks...) and of methods for fitting and checking them.

Predictions exist to support decisions

A prediction is rarely the end goal. A hospital predicts readmission risk so it can decide who gets a follow-up call; a bank predicts default so it can decide whether to lend. That has a big consequence that runs through this whole course: **the costs of different mistakes matter**. Missing a sick patient is not the same as calling a healthy one, so "most accurate" and "most useful" are not always the same model.

BIG IDEA

Machine learning is not about fitting the data you have. It is about predicting data you haven't seen yet, well enough to make good decisions.

Where ML sits

ML is part of the broader field of artificial intelligence, and it grew out of statistics and pattern recognition. Statistics and ML share most of their goals but historically differ in emphasis: statistics focuses more on inference (is this effect real?) and ML more on prediction at scale. The two are converging, and this course uses both perspectives: you will compute p-values *and* worry about test-set performance.

HOW IT'S ASKED

"Define machine learning and explain how it differs from traditional programming." A full answer mentions: learning a rule from data rather than coding it, the model as a function from inputs to outputs, and generalisation to new data as the goal.

Cheatsheet

Machine learning

Learning a prediction rule f from data so that $\hat{y} = f(X)$ predicts Y well on **new** data.

Features vs target

Features / predictors X: the inputs. **Target / response / label** Y: what we predict.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/foundations/what-is-machine-learning

Types of Machine Learning

Regression, classification, clustering and forecasting; supervised vs unsupervised learning; parametric vs non-parametric models. How to tell which kind of problem you are looking at.

Four common tasks

The first question for any ML problem is: **what kind of thing are we predicting?**

TASK	WHAT IS PREDICTED	EXAMPLE	IN THIS COURSE
Regression	A number	House price, CO concentration	Section 3
Classification	A category	Spam or not; diabetic or not	Sections 4 and 6
Clustering	Groups, with no labels given	Customer segments	Section 6
Time-series forecasting	Future values in time order	Electricity demand next month	Time leakage, 2.8

A quick test: if the answer is a quantity you could average (price, temperature), it is regression. If it is a name from a fixed list (yes/no, cat/dog), it is classification.

Features and labels

Every row of a dataset is one observation. The columns you use as inputs are **features** (X). The column you want to predict is the **label** or target (y), the "answer" for that row.

PATIENT	GENE G1	GENE G2	GENE G3	CANCER (LABEL Y)
P1	1	0	1	**1**
P2	0	1	0	**0**

G1 to G3 are features; the cancer column is the label. The model's job is to learn how features relate to the label so it can predict the label for a new patient.

Supervised vs unsupervised learning

- **Supervised learning:** every training example has inputs X *and* a known output Y. We learn a mapping from X to Y. Regression and classification are supervised (linear and logistic regression, SVMs).
- **Unsupervised learning:** we only have X, no labels. We look for structure: groups (clustering), or a compressed representation. K-means is unsupervised.

> RULE OF THUMB
>
> If the dataset has a column you want to predict, it's supervised. If you're asking "what natural groups are in here?", it's unsupervised.

Parametric vs non-parametric models

A **parametric** model has a fixed shape described by a finite list of numbers (parameters). Linear regression is a straight line (or plane) defined by its coefficients $\beta_0, \beta_1, \ldots$. A **non-parametric** model assumes much less about the shape and lets the data decide; K-nearest neighbours predicts by averaging the closest training points.

	PARAMETRIC	NON-PARAMETRIC
Examples	Linear and logistic regression	K-nearest neighbours, decision trees
Data needed	Modest	Lots
Interpretability	High: read the coefficients	Often lower
Main risk	Wrong assumed shape (bias)	Instability, overfitting (variance)

Generative vs discriminative (in brief)

Some approaches model how the data are generated, including the inputs themselves; others focus only on the boundary or relationship needed to predict Y from X. Logistic regression and SVMs are discriminative: they go straight for the decision.

Common doubts

GOING DEEPER **What exactly does "label" mean?**

The label is the **actual answer** for a training row: the value of the target y that the model is trying to predict. Features X are the input information; the label y is the known outcome. In the gene table above, G1, G2 and G3 are features and the cancer column (1 or 0) is the label.

This distinction matters for data leakage later: "validation labels influenced feature selection" means the *answers* of validation rows were used to make a decision, which is not allowed.

FIRST PASS **Does non-parametric mean the model has no parameters?**

No. It means there is no **fixed, finite** set of parameters that defines the model. Linear regression is parametric: whatever the amount of data, it is defined by its coefficients. K-nearest neighbours is non-parametric: it keeps every training point and predicts from the closest ones, so what it must store grows with the data. It still has settings, such as K and the distance measure, but those are hyperparameters you choose.

Cheatsheet

Supervised vs unsupervised

Supervised: have X and Y; learn $X \to Y$ (regression, classification). **Unsupervised:** only X; find structure (clustering).

Parametric vs non-parametric

Parametric (linear/logistic regression): fixed shape, few parameters, less data, interpretable, risk of bias. **Non-parametric** (K-NN): flexible, needs more data, risk of variance.

Regression vs classification

Regression predicts a **number**. Classification predicts a **category**.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/foundations/types-of-machine-learning

The ML Modelling Process

The practical workflow from problem definition to deployment, why it is a loop and not a pipeline, and the difference between an algorithm and its complexity.

The workflow

1. **Define the problem** and the decision the prediction will support. Decide the task type.
2. **Obtain and explore the data.** Identify the target and candidate predictors. Look for missing values, odd units, duplicates.
3. **Split the data** into training, validation and test sets *before* learning anything from it.
4. **Choose candidate models** and fit them on the training data.
5. **Compare them** on validation data, or by cross-validation.
6. **Inspect errors and assumptions.** Go back and revise features, data or model.
7. **Evaluate the final model once** on the test set, then deploy and monitor it.

> A LOOP, NOT A PIPELINE
>
> Step 6 sends you back to earlier steps. Evaluation often reveals that the task definition, the features or the model choice needs to change. What must *not* loop is the test set: it is used once, at the very end.

A revision analogy

Think of studying with past papers. **Training data** are the questions you practise on. **Validation data** are a mock exam you use to decide how to revise. **Test data** are the real exam, which you must not see in advance. If you memorise the practice answers you score 100% on them and may still fail the real exam. That is overfitting.

Algorithm vs complexity

Two different choices hide inside "choosing a model":

	MODEL ALGORITHM	MODEL COMPLEXITY
What it is	The learning technique	How flexible that technique is allowed to be
Examples	Linear regression, logistic regression, decision tree, SVM	Polynomial degree, tree depth, number of features, K in K-NN, penalty λ
Tree example	"Use a decision tree"	"Depth 2" (simple) vs "depth 20" (very flexible)

Settings that control complexity and are chosen by you rather than fitted from the data are called **hyperparameters**. Coefficients are fitted on the training set; hyperparameters are chosen on validation data.

Feature construction

The process includes building new features from domain knowledge: a ratio of debt to income, the day of the week from a timestamp, or X^2 to capture curvature. Task definition, features, model choice and evaluation are connected decisions, not isolated steps.

Common doubts

GOING DEEPER **What is the difference between model algorithm and model complexity?**

The **algorithm** is the learning technique (logistic regression, decision tree, random forest…). The **complexity** is how sophisticated or flexible that particular model is allowed to be.

Example: a decision tree of depth 2 and one of depth 20 use the same algorithm but have very different complexity. The depth-20 tree can carve out tiny regions around individual training points, so it is far more likely to overfit.

Cheatsheet

ML workflow

Define problem → explore data → **split** → fit candidates on train → compare on validation/CV → inspect & revise → test **once** → deploy & monitor.

Hyperparameter

A setting chosen by you, not fitted from data (degree, depth, K, λ, C). Choose it on validation data or by CV.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/foundations/the-ml-modelling-process

Fundamental Limits of Machine Learning

The No Free Lunch theorem, why no algorithm is best for every problem, and the conditions that make learning from data possible at all.

No Free Lunch

The **No Free Lunch theorem** (Wolpert) says that, averaged over *all possible* problems, no learning algorithm beats any other. An algorithm that does brilliantly on one family of problems must do correspondingly worse on another.

WHAT IT MEANS IN PRACTICE

There is no universally best model. A method works when its assumptions suit the problem. So you compare candidate methods on data relevant to your task, and you make your assumptions explicit.

COMMON MISREADING

No Free Lunch does **not** say all algorithms perform equally on your dataset. It is an average over every conceivable problem, most of which look like noise. Real problems have structure, and an algorithm whose assumptions match that structure will do better.

Why learning is possible at all

Training examples alone can never pin down what happens at inputs you haven't seen. Learning works only because of two things:

- **Structure:** the real relationship is reasonably smooth or simple, so nearby inputs have similar outputs.
- **Representative data:** enough observations, drawn from the same population you will predict for.

Every model brings assumptions (its *inductive bias*): linear regression assumes a straight-line relationship, K-NN assumes nearby points behave alike. Those assumptions are what let it fill in the gaps between training points.

The promise behind every model

Most methods assume training and future data are independent draws from the **same** distribution. If you train on patients from one hospital and deploy in another country, that promise is broken, and the model can fail however clever the algorithm.

Two cultures (background)

Leo Breiman's essay *Statistical Modeling: The Two Cultures* contrasts the **data-modelling culture** (assume a stochastic model, check goodness of fit, interpret parameters) with the **algorithmic culture** (treat the mechanism as unknown, judge models by predictive accuracy). Two ideas from it are useful later:

- **Rashomon effect:** many quite different models can fit equally well, so individual coefficients can be unstable (see multicollinearity, 3.9).
- **Occam's razor:** a tension between simplicity and accuracy, which regularisation (Section 5) manages explicitly.

Cheatsheet

No Free Lunch

Averaged over all problems no algorithm wins. In practice: no universally best model, so compare candidates on relevant data and state your assumptions.

NFL misreading

Wrong: "all algorithms do equally well on my data". Right: it is an average over every possible problem; real problems have structure.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/foundations/fundamental-limits-of-machine-learning

Empirical Risk and Population Risk

Population risk is the error we truly care about; empirical risk is the error we can compute. The gap between them is the central problem of machine learning.

The setting

Assume each observation (X_i, Y_i) is an independent draw from some unknown **data-generating distribution** $P(X, Y)$. We never see P itself, only a finite sample of n draws from it. We want a function f whose predictions $f(X)$ are close to Y for *new* draws from the same P.

Loss: scoring a single prediction

A **loss function** $L(Y, f(X))$ measures how bad one prediction is. For regression the usual choice is squared loss $(y - \hat{y})^2$. Choosing the loss is a real modelling decision (lesson 2.1).

Two risks

POPULATION RISK: WHAT WE CARE ABOUT

$$R(f) = \mathbb{E}\big[L(Y, f(X))\big]$$

The average loss over the whole, unknown population.

EMPIRICAL RISK: WHAT WE CAN COMPUTE

$$\hat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} L\big(Y_i, f(X_i)\big)$$

The average loss over the n observations in our sample.

Because P is unknown, $R(f)$ can't be computed. $\hat{R}_n(f)$ is our estimate of it. On the training data it is the **training error**.

WORKED EXAMPLE: COMPUTING EMPIRICAL RISK

A model predicts $\hat{y} = (8, 5, 10, 3)$ for true values $y = (11, 5, 7, 4)$.

Residuals $y - \hat{y} = (3, 0, -3, 1)$.

Squared-loss empirical risk: $\frac{1}{4}(9 + 0 + 9 + 1) = \frac{19}{4} = 4.75$.

Absolute-loss empirical risk: $\frac{1}{4}(3 + 0 + 3 + 1) = 1.75$.

Same predictions, two different risks, because the two losses punish errors differently.

Why the gap matters

Fitting a model means making $\hat{R}_n(f)$ small. But a flexible enough model can make training error tiny by memorising the sample, noise and all, while its population risk stays large. The difference

$$\text{generalisation gap} = R(f) - \hat{R}_n(f)$$

is the heart of Section 2. We can't compute it exactly, so we estimate $R(f)$ using data the model was not trained on.

HOW IT'S ASKED

"Explain the difference between population risk and empirical risk. Which one can be computed?" Say: population risk is expected loss over the unknown distribution and cannot be computed; empirical risk is average loss on the sample and can. Minimising training loss does not guarantee minimising population risk.

Cheatsheet

Population risk

$R(f) = \mathbb{E}[L(Y, f(X))]$: expected loss over the unknown population. Cannot be computed.

Empirical risk

$\hat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} L(Y_i, f(X_i))$: average loss on the sample.

Generalisation gap

$R(f) - \hat{R}_n(f)$. Estimated with held-out data.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/foundations/empirical-risk-and-population-risk

Section 2

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 · 9 min

2.2 Empirical Risk Minimisation (ERM) · 6 min

2.3 Underfitting and Overfitting · 8 min

2.4 Generalisation and the Generalisation Gap · 7 min

2.5 The Bias–Variance Trade-off · 9 min

2.6 Training, Validation and Test Sets · 8 min

2.7 Cross-validation · 9 min

2.8 Data Leakage and Good ML Practices · 14 min

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

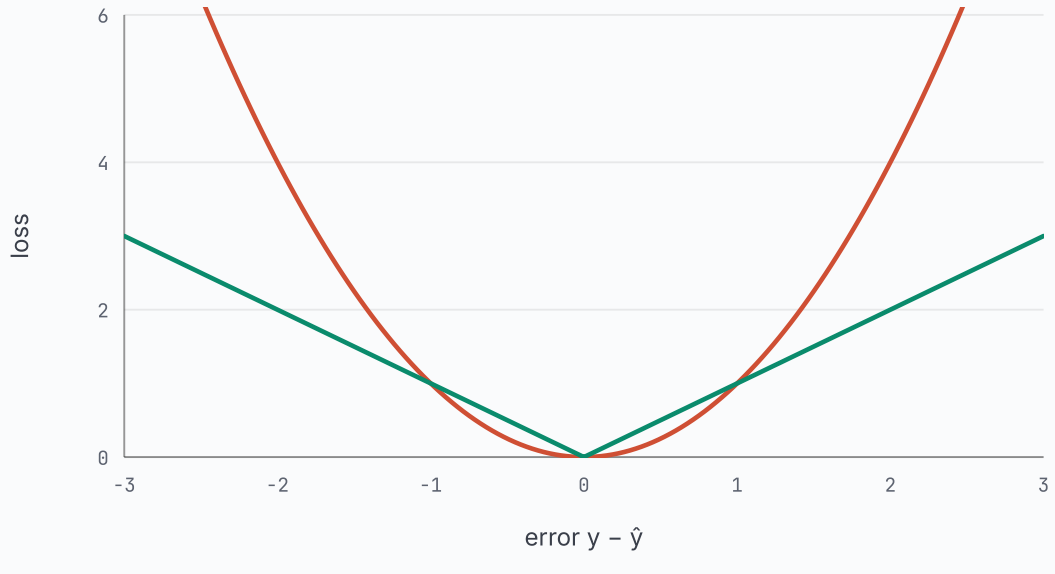
A loss function scores one prediction; risk averages it. Squared, absolute, percentage, 0–1, cross-entropy, cost-weighted and hinge loss: what each punishes, where each is used, and how to compute them by hand.

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

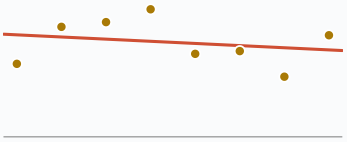
$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_i L(y_i, f(x_i))$. OLS = ERM + squared loss; logistic = ERM + cross-entropy.

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

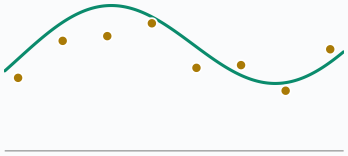
Underfitting and Overfitting

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.

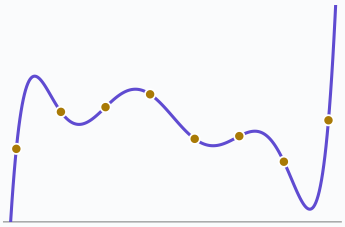
Three fits to the same data



Underfit: a straight line misses the curve.



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



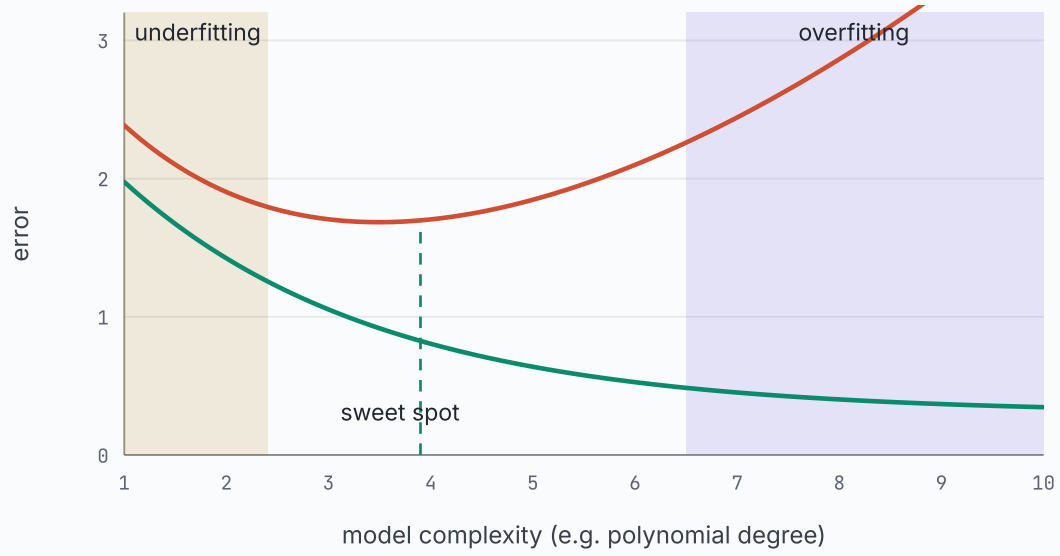
Overfit: a degree-7 polynomial hits every point and wiggles wildly between them.

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

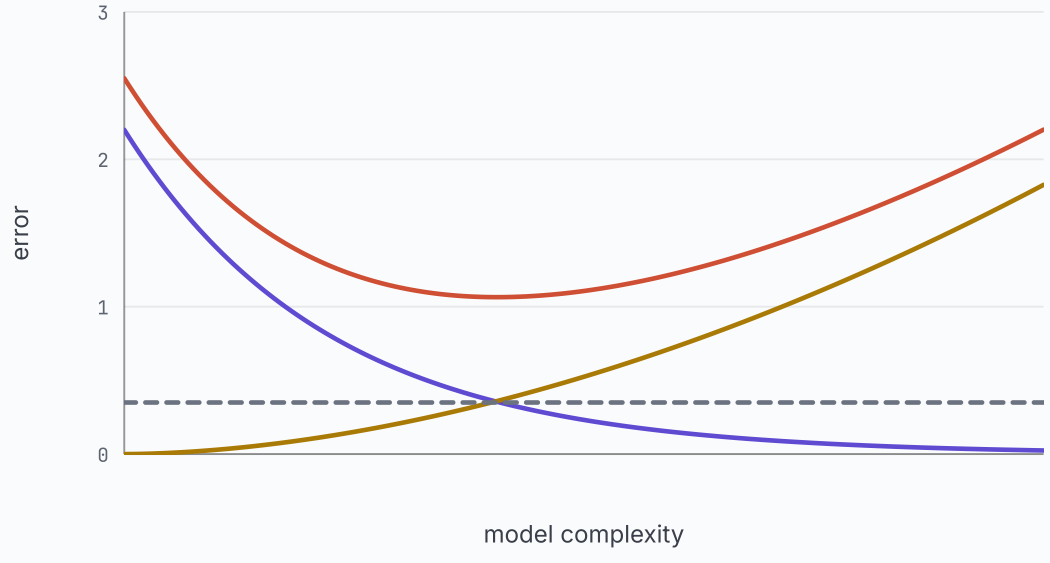
Imagine drawing many different training sets from the same population and fitting the same type of model to each. At one input x_0 you'd get a spread of predictions $\hat{f}(x_0)$. Three things determine how wrong they are:

- **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

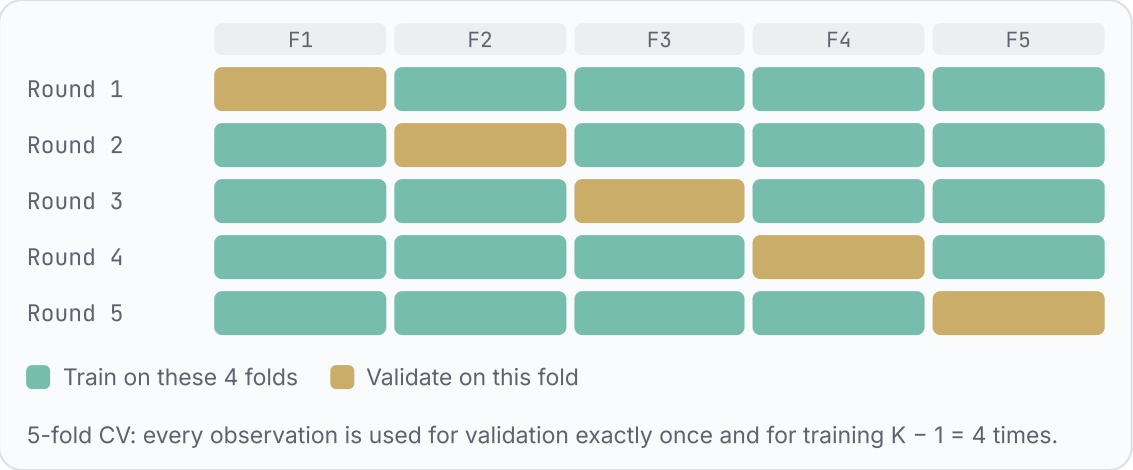
Cross-validation

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



1. Randomly split the data into K roughly equal folds (typically $K = 5$ or 10).
2. For each fold $i = 1, \ldots, K$: 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.

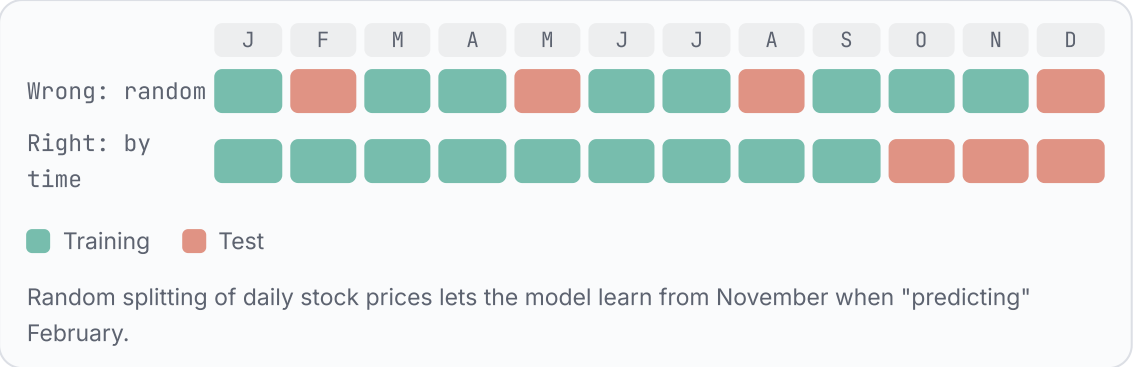
Leak 4: related observations across splits

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.

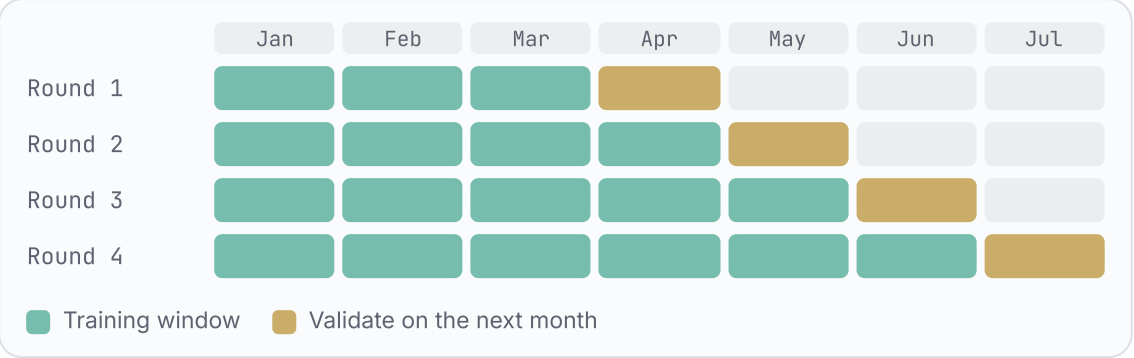
Fix: split by patient ID so all of a patient's records stay together (group split; in scikit-learn, `GroupKFold` and `GroupShuffleSplit`). The same applies to many transactions per customer, purchases per shopper, or measurements per machine.

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

Section 3

Linear Regression

How regression models are built, how their coefficients are calculated and interpreted, and how reliable their predictions are.

3.1 Regression Fundamentals · 6 min

3.2 Simple and Multiple Linear Regression · 8 min

3.3 Ordinary Least Squares (OLS) · 11 min

3.4 Calculating Regression Coefficients by Hand · 10 min

3.5 Interpreting Coefficients · 7 min

3.6 Residuals and Regression Assumptions · 8 min

3.7 R^2, Adjusted R^2, RMSE and Other Metrics · 10 min

3.8 Standard Errors, Confidence Intervals and p-values · 12 min

3.9 Multicollinearity and VIF · 9 min

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.

R1

SUPERVISED LEARNING · REGRESSION LANE · STEP R1

Choose the model

What shape of relationship am I assuming?

Lessons here: 3.1, 3.2

R2

SUPERVISED LEARNING · REGRESSION LANE · STEP R2

Fit it

Which coefficients fit the data best?

Lessons here: 3.3, 3.4 · feeds into Control complexity

R3

SUPERVISED LEARNING · REGRESSION LANE · STEP R3

Read and check it

What do the coefficients mean, and can I trust them?

Lessons here: 3.5, 3.6, 3.8, 3.9 · feeds into Control complexity, Choose the model

R4

SUPERVISED LEARNING · REGRESSION LANE · STEP R4

Score it

How far off are the predictions, in units people understand?

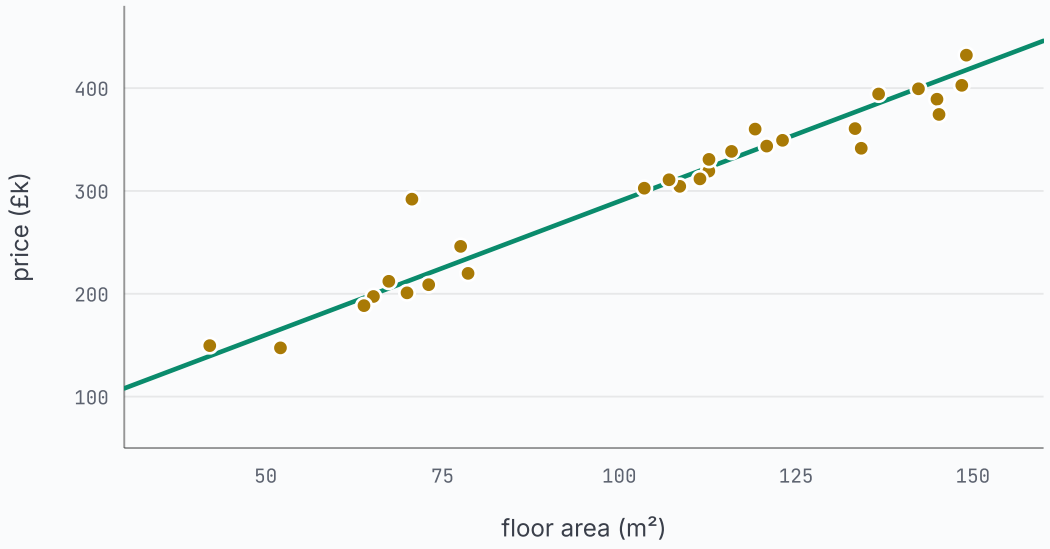
Lessons here: 3.7 · feeds into Evaluate honestly

Regression Fundamentals

Regression predicts a number from inputs. The model, the noise term, what "fitting" means, and how predictions are made, using house prices as a running example.

The question regression answers

Regression predicts a quantitative response Y (a number) from one or more predictors $X_1, \ldots, X_p$. How much will this house sell for? What will CO concentration be at noon? How many patients will arrive next week?



Each dot is a house. The line summarises the trend: on average, more floor area means a higher price. The scatter around it is the noise the model can't explain.

Signal plus noise

Regression models assume the response is a systematic part plus random noise:

$$Y = f(X) + \varepsilon$$

$f(X)$ is the **signal**: the part of Y that depends on the predictors. ε is **noise**: everything else (unmeasured factors, measurement error, pure chance), assumed to average out to zero. Two houses with identical area and bedrooms still sell for different prices; that difference is ε.

Linear regression assumes f is linear: $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$.

Fitting and predicting

- **Fitting** means estimating the unknown coefficients from training data, giving $\hat{\beta}_0, \hat{\beta}_1, \ldots$ (lesson 3.3).
- **Fitted values** $\hat{y}_i$ are the model's predictions for the training points.

- **Predicting** a new case means plugging its predictor values x^* into the fitted equation: $\hat{y}^* = \hat{\beta}_0 + \hat{\beta}_1 x_1^* + \cdots + \hat{\beta}_p x_p^*$.
- **Residuals** $r_i = y_i - \hat{y}_i$ are the observed minus fitted values: how far each training point sits from the line.

Why start with linear regression?

It is the simplest useful ML model: fast, interpretable, with a closed-form solution and a full toolkit for uncertainty (standard errors, p-values, confidence intervals). Many ideas in the rest of the course (loss, overfitting, regularisation) are clearest here first.

Cheatsheet

Signal + noise

$Y = f(X) + \varepsilon$, $\mathbb{E}[\varepsilon] = 0$. Linear regression: $f(X) = \beta_0 + \sum_j \beta_j X_j$.

Residual

$r_i = y_i - \hat{y}_i$ (observed − fitted). Positive → model under-predicted.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/linear-regression/regression-fundamentals

Simple and Multiple Linear Regression

One predictor or many, the model is a weighted sum of inputs. Why a model with X² or interaction terms is still "linear" regression: it only has to be linear in the coefficients.

Simple linear regression

One predictor:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

β_0 is the **intercept** (the predicted Y when $X = 0$) and β_1 the **slope** (the change in predicted Y per one-unit increase in X). The fitted model is a straight line.

Multiple linear regression

Several predictors:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

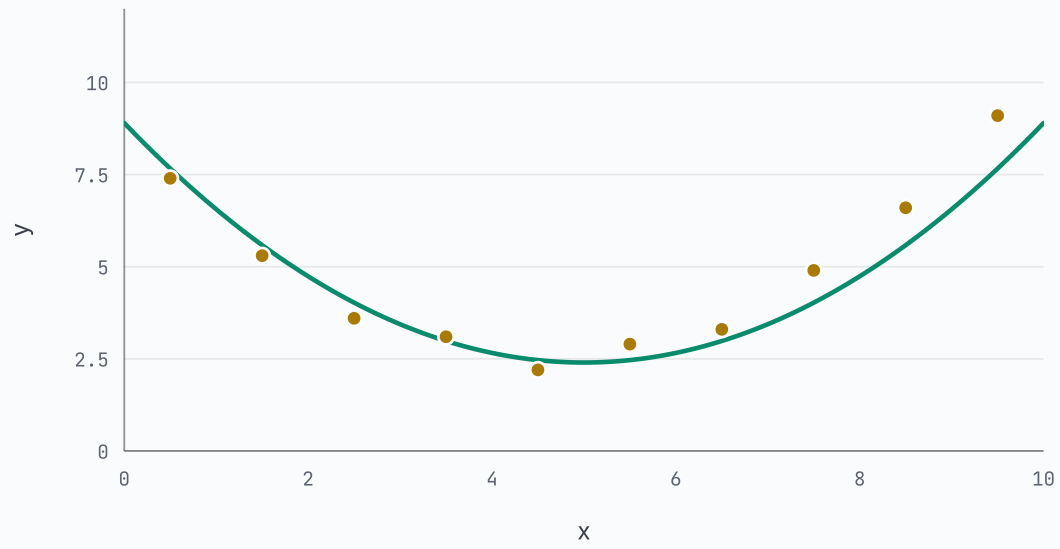
With two predictors the fitted model is a **plane** in 3-D (X_1, X_2, Y) space; with more it is a **hyperplane**. The algebra is identical, it just can't be drawn. It's a **parametric** model: a finite list of $p + 1$ coefficients.

"Linear" means linear in the coefficients

This surprises almost everyone. The model below is still **linear regression**:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \varepsilon$$

The curve is a parabola, but the *unknowns* $\beta_0, \beta_1, \beta_2$ each appear on their own, multiplied by a known number and added up. Create a new column $X_2 = X^2$ and it becomes an ordinary multiple regression with two predictors. The same fitting method (OLS) works unchanged.



ŷ = 8.9 − 2.6x + 0.26x² (linear in β)

A curved fit from linear regression: the predictors are x and x², and the model is a weighted sum of them.

Interactions are linear too

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \varepsilon$$

$X_1 X_2$ is just another column computed from the data. The interaction lets the effect of X_1 depend on X_2: the slope for X_1 is $\beta_1 + \beta_3 X_2$.

MODEL	LINEAR REGRESSION?	WHY
$Y = \beta_0 + \beta_1 X + \beta_2 X^2$	Yes	Each β multiplies a known column
$Y = \beta_0 + \beta_1 \log X$	Yes	$\log X$ is a known column
$Y = \beta_0 + \beta_1 X_1 X_2$	Yes	$X_1 X_2$ is a known column
$Y = \beta_0 e^{\beta_1 X}$	No	β_1 sits inside an exponential
$Y = \beta_0 + X^{\beta_1}$	No	β_1 is an exponent

TEST

Treat the data as fixed numbers. If the model is a sum of (coefficient × something computed from the data), it is linear regression, however curved the picture.

Common doubts

FIRST PASS **Why is $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$ still called linear regression when X is squared?**

Because "linear" refers to the **coefficients**, not the inputs. X^2 is computed from the data, so it's a known number for every row. The model is $\beta_0 \cdot 1 + \beta_1 \cdot (X) + \beta_2 \cdot (X^2)$: each unknown multiplies a known value, and the terms are added. Treat X^2 as a second predictor column and it's an ordinary multiple regression.

FIRST PASS **Why is a model with an interaction term $X_1 X_2$ still linear regression?**

Same reason: $X_1 X_2$ is a column you can compute from the data before fitting. The model $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$ is still a weighted sum with unknown weights $\beta_0, \ldots, \beta_3$, so OLS fits it exactly as before.

FIRST PASS **What does "linear in the coefficients" mean?**

The unknown parameters appear only to the first power, each multiplied by something known, and the pieces are added. The inputs themselves can be transformed in any way (squared, logged, multiplied together). A model like $\beta_0 e^{\beta_1 X}$ is **not** linear in its coefficients, because β_1 is inside the exponential.

Cheatsheet

Multiple linear regression

$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$; prediction $\hat{y}^* = \hat{\beta}_0 + \sum_j \hat{\beta}_j x_j^*$.

"Linear" regression

Linear in the **coefficients**, not the inputs. X^2, $\log X$, $X_1 X_2$ are allowed.

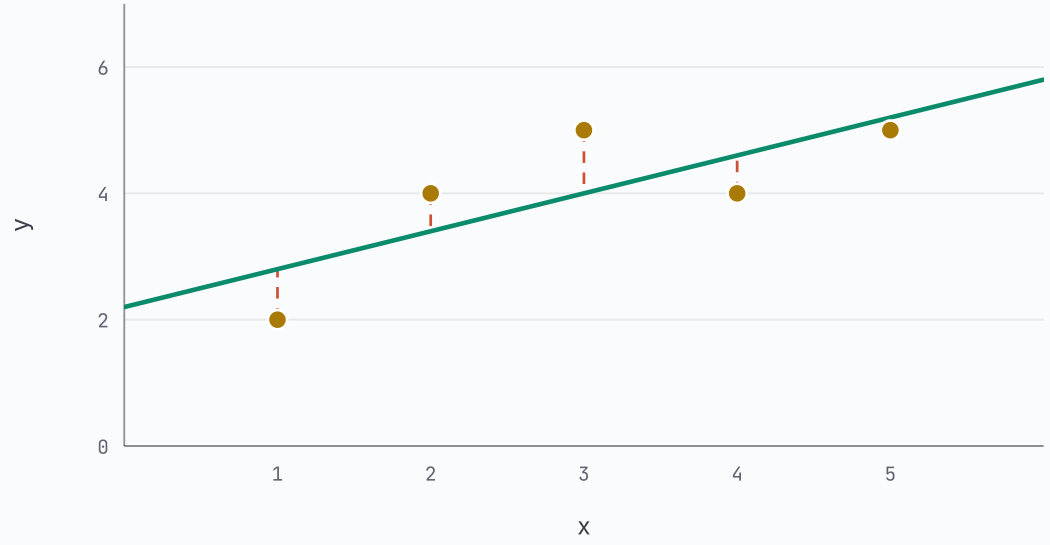
Online version with the interactive quiz: airesearchlabs.si/learn/ml/linear-regression/simple-and-multiple-linear-regression

Ordinary Least Squares (OLS)

OLS picks the line that minimises the residual sum of squares. Residuals and RSS, the design matrix and its dimensions, the normal equations, the closed-form solution, and when it breaks.

The problem we're solving

Many lines could pass through a cloud of points. Which is "best"? OLS's answer: the one with the smallest total **squared vertical distance** to the points.



Dataset B: five points. The dashed lines are residuals. OLS makes the sum of their squares as small as possible.

Residuals and RSS

For observation i, the residual is observed minus fitted: $r_i = y_i - \hat{y}_i$. OLS chooses coefficients to minimise the **residual sum of squares**:

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip}\right)^2 = \sum_i r_i^2$$

This is exactly ERM with squared loss (lesson 2.2).

WHY SQUARES?

Raw residuals would cancel out (positives and negatives). Absolute values are awkward to minimise. Squares are smooth, so calculus gives a closed-form answer, and they punish big misses more.

The design matrix

Stack the data into matrices. With n observations and p predictors, the **design matrix** X has n rows and $p+1$ columns: a first column of ones (for the intercept), then one column per predictor.

$$X = \underset{n\times(p+1)}{\begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}}, \quad \beta = \underset{(p+1)\times 1}{\begin{pmatrix} \beta_0 \\ \vdots \\ \beta_p \end{pmatrix}}, \quad y = \underset{n\times 1}{\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}}$$

Then all the fitted values at once are $\hat{y} = X\beta$, and $\mathrm{RSS}(\beta) = (y - X\beta)^\top (y - X\beta)$.

EXAMPLE: AREA, BEDROOMS, PRICE

HOUSE	AREA (M²)	BEDROOMS	PRICE (£K)
1	50	1	150
2	70	2	200
3	80	2	230
4	100	3	280

Price is the response, so it goes in y, not X. Two predictors means $p = 2$:

$$X = \underset{4\times 3}{\begin{pmatrix} 1 & 50 & 1 \\ 1 & 70 & 2 \\ 1 & 80 & 2 \\ 1 & 100 & 3 \end{pmatrix}}, \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix}, \quad y = \begin{pmatrix} 150 \\ 200 \\ 230 \\ 280 \end{pmatrix}$$

Three coefficients: intercept, area, bedrooms. One coefficient per column of X.

The normal equations

Setting the derivative of RSS to zero gives the **normal equations**, and if $X^\top X$ is invertible, the closed-form solution:

OLS SOLUTION

$$X^\top X\,\hat{\beta} = X^\top y \qquad \Longrightarrow \qquad \hat{\beta} = (X^\top X)^{-1} X^\top y$$

Shapes: $X^\top X$ is $(p+1)\times(p+1)$; $X^\top y$ is $(p+1)\times 1$; so $\hat{\beta}$ is $(p+1)\times 1$.

▶ Show the derivation (optional)

WORKED EXAMPLE VIA MATRICES

Data $(x, y) = (1, 3), (2, 5), (3, 7)$. Design matrix rows $(1, x_i)$.

$$X^\top X = \begin{pmatrix} 3 & 6 \\ 6 & 14 \end{pmatrix}, \quad X^\top y = \begin{pmatrix} 15 \\ 34 \end{pmatrix}$$

For a 2×2 matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ the inverse is $\frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$. Here $\det = 42 - 36 = 6$:

$$\hat{\beta} = \frac{1}{6}\begin{pmatrix} 14 & -6 \\ -6 & 3 \end{pmatrix}\begin{pmatrix} 15 \\ 34 \end{pmatrix} = \frac{1}{6}\begin{pmatrix} 210 - 204 \\ -90 + 102 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

So $\hat{y} = 1 + 2x$, and every residual is zero (the points lie exactly on the line).

When it breaks

$(X^\top X)^{-1}$ doesn't exist when $X^\top X$ is **singular**:

- **Perfect multicollinearity:** one predictor is an exact linear combination of others (e.g. height in cm and height in inches).
- **More coefficients than observations:** $p + 1 > n$.

If $X^\top X$ is *nearly* singular, the solution exists but is unstable (lesson 3.9). Ridge regression (5.4) fixes both cases.

Common doubts

FIRST PASS **The data have Area, Bedrooms and Price. Why does the coefficient vector have three entries, not four?**

Count columns of the **design matrix**, not columns of the spreadsheet. Price is the **response**, so it goes into y, not into X. The predictors are Area and Bedrooms ($p = 2$), plus the column of ones for the intercept: $p + 1 = 3$ columns, so $\beta = (\beta_0, \beta_1, \beta_2)$. There's one coefficient per column of X, and no β_3.

FIRST PASS **I find the OLS matrix derivation slow. What do I actually need to understand?**

Keep these, which carry most of the meaning:

- $Y = X\beta + \varepsilon$ and what each piece is (with its dimensions).
- Residual vector $y - X\beta$, and RSS = sum of squared residuals.
- OLS chooses β to minimise RSS.
- The result: normal equations $X^\top X\hat{\beta} = X^\top y$, solution $\hat{\beta} = (X^\top X)^{-1}X^\top y$, and that it fails when $X^\top X$ is singular.

The step-by-step calculus (expanding the quadratic and differentiating) can be deferred until you need it. It is collapsed above under "Show the derivation".

Cheatsheet

RSS

$\mathrm{RSS} = \sum_i (y_i - \hat{y}_i)^2 = (y - X\beta)^\top (y - X\beta)$.

OLS normal equations

$X^\top X \hat{\beta} = X^\top y \Rightarrow \hat{\beta} = (X^\top X)^{-1} X^\top y$ (if invertible).

Design matrix shape

X: $n \times (p+1)$, first column ones. β: $(p+1) \times 1$.
$X^\top X$: $(p+1) \times (p+1)$.

2×2 inverse

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Online version with the interactive quiz: airesearchlabs.si/learn/ml/linear-regression/ordinary-least-squares

Calculating Regression Coefficients by Hand

The slope and intercept formulas for simple regression, a full worked example with a table of deviations, and why a predictor with little variation gives an unreliable slope.

The formulas

SIMPLE REGRESSION $Y = B_0 + B_1 X + E$

$$\hat{\beta}_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} = \frac{S_{xy}}{S_{xx}}, \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

- The **numerator** S_{xy} measures how x and y move together: positive when above-average x goes with above-average y.
- The **denominator** S_{xx} measures how spread out x is.
- The fitted line always passes through the point of means $(\bar{x}, \bar{y})$.

Worked example: dataset B

$x = 1, 2, 3, 4, 5$ and $y = 2, 4, 5, 4, 5$. So $n = 5$, $\bar{x} = 3$, $\bar{y} = 4$.

X	Y	$X - \bar{X}$	$Y - \bar{Y}$	$(X - \bar{X})(Y - \bar{Y})$	$(X - \bar{X})^2$
1	2	-2	-2	4	4
2	4	-1	0	0	1
3	5	0	1	0	0
4	4	1	0	0	1
5	5	2	1	2	4
Sum				$S_{xy} = 6$	$S_{xx} = 10$

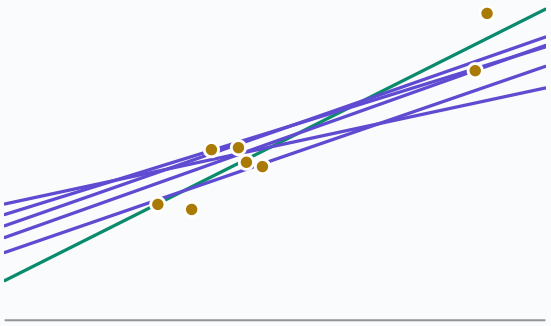
$\hat{\beta}_1 = 6/10 = 0.6$ and $\hat{\beta}_0 = 4 - 0.6 \times 3 = 2.2$. The fitted line is $\hat{y} = 2.2 + 0.6x$.

Fitted values: $(2.8, 3.4, 4.0, 4.6, 5.2)$. Residuals: $(-0.8, 0.6, 1.0, -0.6, -0.2)$. They sum to zero, which always happens when the model has an intercept.

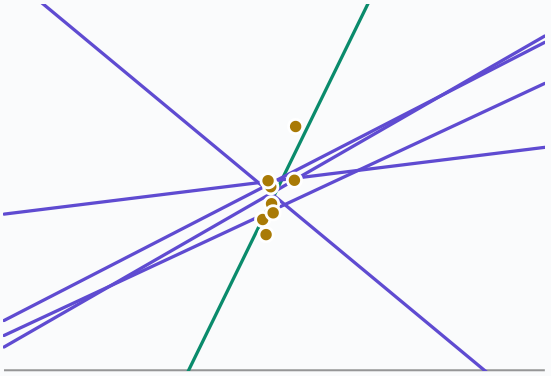
RSS $= 0.64 + 0.36 + 1.00 + 0.36 + 0.04 = 2.4$. We'll reuse this in 3.7 and 3.8.

Why low variance in X is a problem

Look at the denominator S_{xx}. If the x values barely vary, S_{xx} is tiny, and small changes in a few y values swing the slope a lot. If x is constant, $S_{xx} = 0$ and the slope can't be computed at all.



Spread-out x: six resampled datasets give six similar slopes.



Clustered x (4.5–5.5): same truth, same noise, wildly different slopes.

DESIGN LESSON

To estimate how Y changes with X, you need to observe X over a decent range. This is limited information about the slope, and it shows up later as a large standard error (3.8).

Common doubts

FIRST PASS **Do I need to do the full manual calculation (means, deviations, products, squares)?**

For a closed-book exam with "calculate by hand" questions, yes: practise the table above until you can do it in a few minutes, because it earns method marks even if the arithmetic slips. Conceptually, what matters is: the slope is "how x and y co-vary" divided by "how much x varies", and the line passes through $(\bar{x}, \bar{y})$. If you're short on time, understand that first and return to the drill before the exam.

FIRST PASS **Why does low variance in the predictor make the slope unreliable?**

The slope's denominator is $S_{xx} = \sum(x_i - \bar{x})^2$. When the x values are clustered, S_{xx} is small, so tiny changes in the y values (noise) produce big changes in $\hat{\beta}_1$. Intuitively, if you only see houses between 99 and 101 m², you have almost no information about how price changes with size. In 3.8 this appears as $\mathrm{SE}(\hat{\beta}_1) = \sqrt{\hat{\sigma}^2 / S_{xx}}$ getting large.

Cheatsheet

Simple regression by hand

$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}} = \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sum(x_i-\bar{x})^2}$, $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$.

Slope by hand

1) Means $\bar{x}, \bar{y}$. 2) Deviations. 3) Products & squares. 4) Sum → S_{xy}, S_{xx}. 5) Divide; then intercept. 6) Check residuals sum to 0.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/linear-regression/calculating-coefficients-by-hand

Interpreting Coefficients

What a regression coefficient means, the "holding other predictors fixed" clause, units and standardisation, why coefficients can change sign, and why association is not causation.

THE INTERPRETATION SENTENCE

$\hat{\beta}_j$ is the predicted change in Y for a one-unit increase in X_j, **holding all other included predictors fixed**.

WORKED EXAMPLE

Fitted model: $\hat{y} = 12 + 3x_1 - 2x_2$.

- Holding x_2 fixed, +1 in x_1 raises the prediction by 3.
- Holding x_1 fixed, +1 in x_2 lowers it by 2.
- The intercept 12 is the prediction when $x_1 = x_2 = 0$.
- For $(x_1, x_2) = (4, 5)$: $\hat{y} = 12 + 12 - 10 = 14$.

Four classic misreadings

1. **A coefficient is not a simple correlation.** It's the effect *given the other predictors*, so it can change size or even sign when you add or remove predictors.
2. **Size depends on units.** Measuring a predictor in grams instead of kilograms multiplies its coefficient by 1/1000. To compare importance fairly, **standardise** the predictors first; then each coefficient is the change in Y per one-SD change in X_j.
3. **Association is not causation.** In observational data, a positive coefficient doesn't mean changing X_j will change Y. A confounder may drive both.
4. **The intercept may be meaningless.** It's the prediction at $X = 0$, which may be impossible (a house with zero floor area).

The negative bedrooms puzzle

A house-price model with floor area, bedrooms and distance to centre gives bedrooms a **negative** coefficient. Does adding a bedroom make a house cheaper? Not necessarily. *Holding floor area fixed*, more bedrooms means each room is smaller, which buyers may value less. Correlation between area and bedrooms also makes individual coefficients unstable (3.9). The coefficient answers a narrower question than it first appears to.

HOW IT'S ASKED

"Interpret -4 in $\hat{y} = 5 + 2x_1 - 4x_2$." Answer: holding x_1 fixed, each one-unit increase in x_2 is associated with a 4-unit decrease in predicted y. Mention "holding fixed" and "associated with", not "causes".

Cheatsheet

Coefficient sentence

"Holding the other predictors fixed, a one-unit increase in X_j is associated with a $\hat{\beta}_j$ change in predicted Y."

Comparing coefficients

Raw sizes depend on units. Standardise predictors first. And association ≠ causation.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/linear-regression/interpreting-coefficients

Residuals and Regression Assumptions

The assumptions behind OLS inference, and how residual plots reveal curvature, non-constant variance, autocorrelation and outliers. How to describe a residual plot in an exam.

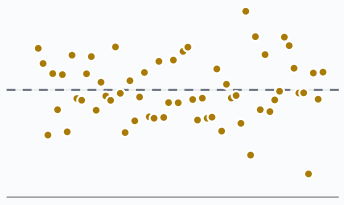
The assumptions

OLS *fitting* needs no assumptions: it just minimises RSS. But standard errors, p-values and confidence intervals (3.8) rely on assumptions about the noise ε:

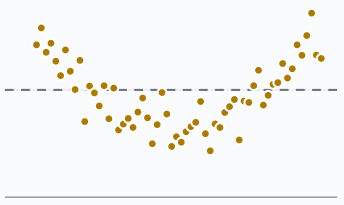
- **Mean zero** at every X (the linear form is right, nothing systematic is missing).
- **Constant variance** (homoskedasticity).
- **Independence** between observations.
- **Roughly normal**, for exact t-tests in small samples.

We can't see ε, but residuals $r_i = y_i - \hat{y}_i$ are our estimates of it. The main tool is a plot of **residuals against fitted values**.

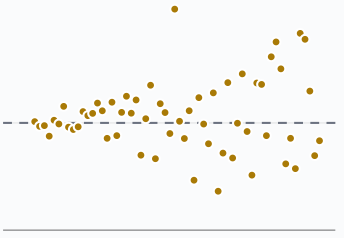
Reading residual plots



Random cloud, constant width: assumptions look reasonable.



U-shape: missing curvature. Try adding X^2.



Fan: spread grows with the fitted value (heteroskedasticity).

WHAT YOU SEE	WHAT IT SUGGESTS	WHAT YOU MIGHT DO
Random cloud around 0, constant width	Assumptions look reasonable	Proceed (this isn't proof)
Curve or systematic pattern	Nonlinearity or missing features	Add X^2, interactions, new predictors
Fan: spread grows with fitted value	Heteroskedasticity (non-constant variance)	Transform Y (e.g. log), weighted least squares. SEs and p-values are unreliable.
Waves or runs of same-sign residuals over time	Autocorrelation (time dependence)	Time-series methods
Heavy tails or extreme points in a Q–Q plot	Non-normal errors, outliers	Investigate outliers; robust loss

A **Q–Q plot** compares the residuals' quantiles with those of a normal distribution. Points along the diagonal mean "approximately normal"; curling away at the ends means heavy tails.

WORDING THAT EARNS MARKS

Describe the pattern, name the assumption it challenges, and suggest a possible cause or action. For example: "The residuals fan out as fitted values increase, suggesting non-constant variance; standard errors may be unreliable; consider modelling log(Y)." A plot never *proves* anything.

Cheatsheet

Residual plot patterns

Random cloud → OK. Curve → missing nonlinearity. Fan → heteroskedasticity. Waves in time → autocorrelation. Heavy tails → outliers / non-normal.

OLS noise assumptions

Mean zero · constant variance · independent · (roughly normal for t-tests).

Online version with the interactive quiz: airesearchlabs.si/learn/ml/linear-regression/residuals-and-regression-assumptions

R², Adjusted R², RMSE and Other Metrics

MAE, MSE, RMSE, MAPE and average error; R^2 as improvement over predicting the mean; out-of-sample R^2 that can go negative; and adjusted R^2, which penalises useless predictors. With worked calculations.

Error metrics

Let $e_i = y_i - \hat{y}_i$ on n evaluation observations, ideally held-out data.

MEASURE	FORMULA	NOTES
MAE	$\frac{1}{n}\sum \lvert e_i \rvert$	Average miss in units of Y. Robust to outliers.
Average error	$\frac{1}{n}\sum e_i$	Signed bias: positives and negatives cancel.
MSE	$\frac{1}{n}\sum e_i^2$	Squared units.
RMSE	$\sqrt{\frac{1}{n}\sum e_i^2}$	Units of Y. Penalises large errors more than MAE.
RSS / SSE	$\sum e_i^2$	Total squared error; grows with n.
MAPE	$\frac{100\%}{n}\sum \left\lvert \frac{e_i}{y_i} \right\rvert$	Percentage error; breaks if $y_i \approx 0$.

WORKED EXAMPLE

Test residuals $2, -1, 3$. MAE $= (2+1+3)/3 = 2$. MSE $= (4+1+9)/3 = 4.67$. RMSE $= \sqrt{4.67} = 2.16$. Average error $= (2-1+3)/3 = 1.33$ (the model tends to under-predict).

Always compare against a **baseline**: the "average predictor" that predicts $\bar{y}_{\text{train}}$ for everyone. A model is only useful if it beats that.

R²

COEFFICIENT OF DETERMINATION

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}, \qquad \text{TSS} = \sum_i (y_i - \bar{y})^2$$

TSS is the squared error of the baseline "always predict the mean". So R^2 is the fraction of the baseline's squared error the model removes. In-sample with an intercept, $0 \le R^2 \le 1$.

DATASET B

RSS $= 2.4$ (lesson 3.4). TSS $= 4 + 0 + 1 + 0 + 1 = 6$. $R^2 = 1 - 2.4/6 = 0.6$: the line removes 60% of the mean-only model's squared error.

R² CAUTIONS

- $R^2 = 0.75$ does **not** mean 75% of predictions are correct.
- A high in-sample R^2 doesn't guarantee good out-of-sample prediction.
- A low R^2 doesn't mean the predictors are useless: the signal may be nonlinear, or Y very noisy.
- R^2 says nothing about correct specification: check residuals.

Out-of-sample R²

$$R^2_{\text{OS}} = 1 - \frac{\sum_{\text{test}}(y_i - \hat{y}_i)^2}{\sum_{\text{test}}(y_i - \bar{y}_{\text{train}})^2}$$

It uses held-out data and the **training** mean as the baseline, so it can be **negative**: the model does worse than just predicting the training average. You want it high and close to the in-sample value.

Adjusted R²

Ordinary R^2 **never decreases** when you add a predictor, even pure noise, because OLS can always set its coefficient to zero and usually finds a tiny bit of in-sample improvement. Adjusted R^2 penalises model size:

$$R^2_{\text{adj}} = 1 - \frac{\text{RSS}/(n-p-1)}{\text{TSS}/(n-1)}$$

It goes down when an unhelpful predictor is added and up when a predictor improves fit enough to justify itself.

WORKED EXAMPLE

RSS $= 40$, TSS $= 100$, $n = 20$, $p = 2$. $R^2 = 0.60$. $R^2_{\text{adj}} = 1 - \frac{40/17}{100/19} = 1 - \frac{2.353}{5.263} = 0.553$.

Dataset B ($n = 5$, $p = 1$): $R^2_{\text{adj}} = 1 - \frac{2.4/3}{6/4} = 1 - 0.533 = 0.467$. The penalty is large when n is tiny.

CHOOSING A METRIC	USE
Typical error in plain units, robust to outliers	MAE
Large errors are especially bad	RMSE
Relative error, y never near 0	MAPE
Improvement over the mean baseline	R^2 (out-of-sample)
Comparing in-sample fits with different numbers of predictors	Adjusted R^2 (or better, CV)

Cheatsheet

MAE / RMSE

MAE $= \frac{1}{n}\sum|e_i|$ (robust). RMSE $= \sqrt{\frac{1}{n}\sum e_i^2}$ (punishes big errors). Both in units of Y.

R^2

$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$, $\text{TSS} = \sum(y_i - \bar{y})^2$. Share of the mean-baseline's error removed.

Adjusted R^2

$1 - \frac{\text{RSS}/(n-p-1)}{\text{TSS}/(n-1)}$. Falls when a useless predictor is added.

Out-of-sample R^2

Baseline is $\bar{y}_{\text{train}}$ on test data. Can be negative.

R^2 misreadings

Not "% correct". High in-sample ≠ good prediction. Low ≠ useless predictors.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/linear-regression/r-squared-adjusted-r-squared-and-rmse

Standard Errors, Confidence Intervals and p-values

How uncertain is each coefficient? Standard errors, t-statistics, p-values and confidence intervals, with a full worked example, a real regression output to read, and what a p-value is not.

Coefficients are estimates

Collect a new sample and you'd get different $\hat{\beta}$s. Their sampling uncertainty, under uncorrelated errors with equal variance σ^2, is

COEFFICIENT UNCERTAINTY

$$\mathrm{Cov}(\hat{\beta}) = \sigma^2 (X^\top X)^{-1}, \qquad \hat{\sigma}^2 = \frac{\mathrm{RSS}}{n-p-1}, \qquad \mathrm{SE}(\hat{\beta}_j) = \sqrt{\hat{\sigma}^2 \left[(X^\top X)^{-1}\right]_{jj}}$$

$n-p-1$ are the **residual degrees of freedom**: n observations minus $p+1$ estimated coefficients.

For simple regression this reduces to $\mathrm{SE}(\hat{\beta}_1) = \sqrt{\hat{\sigma}^2 / S_{xx}}$. The SE is smaller with **less noise, more data, more spread in X, and less collinearity**.

The t-test

Is the true β_j really non-zero? Test $H_0 : \beta_j = 0$ against $H_1 : \beta_j \neq 0$ with

$$t_j = \frac{\hat{\beta}_j}{\mathrm{SE}(\hat{\beta}_j)} \sim t_{n-p-1} \text{ under } H_0$$

The **p-value** is the probability, *assuming H_0 is true* and the model assumptions hold, of a $|t|$ at least as large as the one observed. If $p < 0.05$ we call the coefficient statistically significant at 5%.

Confidence intervals

$$\hat{\beta}_j \pm t_{n-p-1,\,1-\alpha/2} \times \mathrm{SE}(\hat{\beta}_j)$$

With $\alpha = 0.05$: about 95% of intervals built this way (over repeated samples) contain the true β_j. If the interval excludes 0, the two-sided test rejects H_0 at 5%. For large n, $t_{0.975} \approx 1.96$, so "estimate ± 2 SE" is a good shortcut.

WORKED EXAMPLE: DATASET B ($Y = 2.2 + 0.6X$)

RSS $= 2.4$, df $= 5 - 1 - 1 = 3$, so $\hat{\sigma}^2 = 0.8$.

$\mathrm{SE}(\hat{\beta}_1) = \sqrt{0.8/10} = \sqrt{0.08} = 0.283$.

$t = 0.6/0.283 = 2.12$. The critical value $t_{3,\,0.975} = 3.182$. Since $2.12 < 3.182$ we **fail to reject** H_0 ($p \approx 0.12$).

95% CI: $0.6 \pm 3.182 \times 0.283 = 0.6 \pm 0.90 = (-0.30,\ 1.50)$. It includes 0, consistent with the test.

Moral: a positive slope from 5 points is not reliable evidence of a positive true slope.

Size is not significance

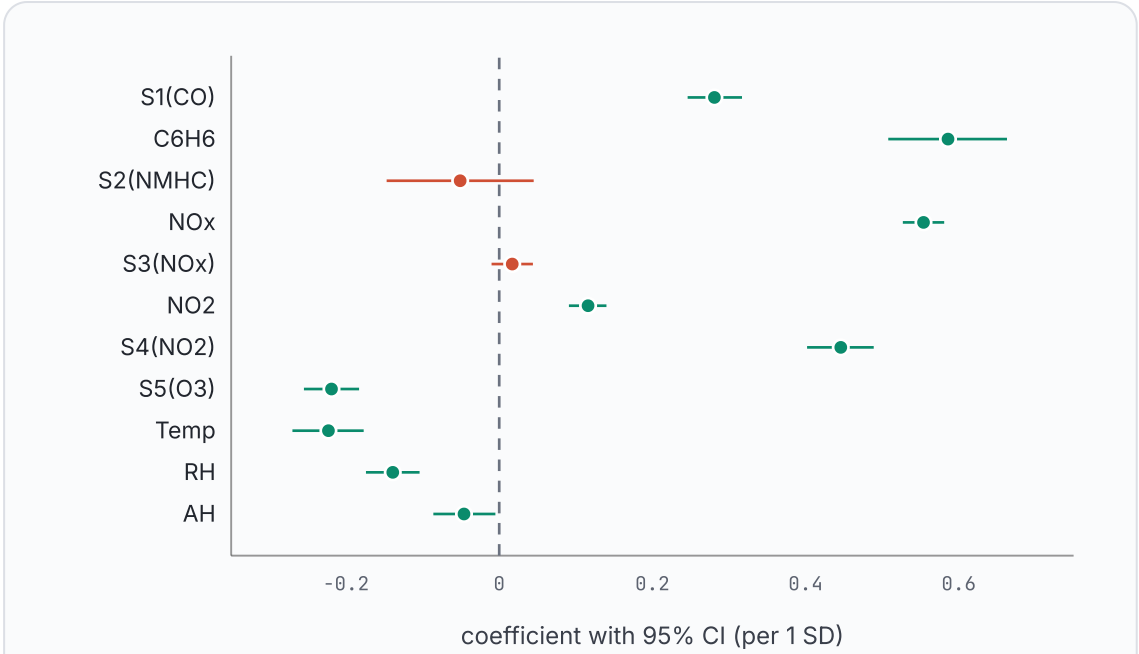
Predictor 1: $\hat{\beta} = 6$, SE $= 3$, so $t = 2$. Predictor 2: $\hat{\beta} = 2$, SE $= 0.5$, so $t = 4$. Predictor 2 has the smaller coefficient but stronger evidence of being non-zero, because it's estimated more precisely. A large coefficient can come with large uncertainty.

WHAT A P-VALUE IS NOT

- Not the probability that H_0 is true.
- Not a measure of effect size: with huge n, trivial effects become "significant".
- A large p-value isn't proof of no effect: small n or collinear predictors can hide a real one.

Reading a real output

Predicting carbon monoxide (mg/m³) from 11 standardised sensor and weather readings in a public air-quality dataset ($n \approx 7{,}700$, 80/20 split). Coefficients are the change in CO per one-SD change in each predictor.



Two intervals (coral) cross zero: those coefficients are not significant at 5%. Training R^2 = 0.888; test R^2 = 0.885.

PREDICTOR	COEF	SE	T	P	VIF
C6H6 (benzene)	0.586	0.040	14.8	0.000	39.9
S2(NMHC) sensor	−0.051	0.049	−1.04	0.298	61.1
NOx	0.554	0.014	39.7	0.000	5.0
S4(NO2) sensor	0.446	0.022	20.0	0.000	12.7
Temperature	−0.223	0.024	−9.5	0.000	14.2

1. **Overall fit:** $R^2 \approx 0.89$ in and out of sample, so little overfitting (n is large relative to $p = 11$).
2. **Significance:** nine of eleven predictors have tiny p-values; S2(NMHC) and S3(NOx) don't.
3. **Why is S2(NMHC) insignificant?** Its VIF is 61 and it correlates 0.98 with benzene. It's nearly a duplicate, so its SE is inflated. That's not evidence the sensor is physically irrelevant: given benzene is in the model, it adds little.
4. **Association ≠ causation:** the negative temperature coefficient doesn't mean cooling the air raises CO; it reflects correlated seasonal and traffic patterns.

Cheatsheet

Standard errors

$\hat{\sigma}^2 = \frac{\text{RSS}}{n-p-1}$, $\text{SE}(\hat{\beta}_j) = \sqrt{\hat{\sigma}^2[(X^\top X)^{-1}]_{jj}}$. Simple: $\sqrt{\hat{\sigma}^2 / S_{xx}}$.

t-test & CI

$t = \hat{\beta}_j / \text{SE}$, df $= n - p - 1$. CI: $\hat{\beta}_j \pm t_{1-\alpha/2}\,\text{SE}$ (≈ ±2 SE).

p-value sentence

"If the true coefficient were zero (and assumptions hold), a t-statistic at least this extreme would occur with probability p."

p-value misreadings

Not P(H_0 true). Not effect size. Large p ≠ no effect.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/linear-regression/standard-errors-confidence-intervals-and-p-values

Multicollinearity and VIF

When predictors carry nearly the same information, coefficients become unstable and standard errors explode, even though predictions can stay fine. The variance inflation factor, how to compute it, and what to do.

The problem

Recall $\mathrm{Cov}(\hat{\beta}) = \sigma^2(X^\top X)^{-1}$. If $X^\top X$ is **nearly singular**, its inverse has huge entries, so coefficient variances explode. Two causes:

- **Multicollinearity:** a predictor is (almost) a linear combination of others, e.g. two sensors measuring nearly the same pollutant, or height in cm and in inches.
- **Over-parameterisation:** the number of predictors p is close to, equal to or larger than n.

TWO PEOPLE CARRYING ONE SOFA

If two predictors carry nearly the same information, the model can't tell which deserves the credit. Their combined effect is well determined, so predictions are fine, but the split between them is arbitrary and flips from sample to sample.

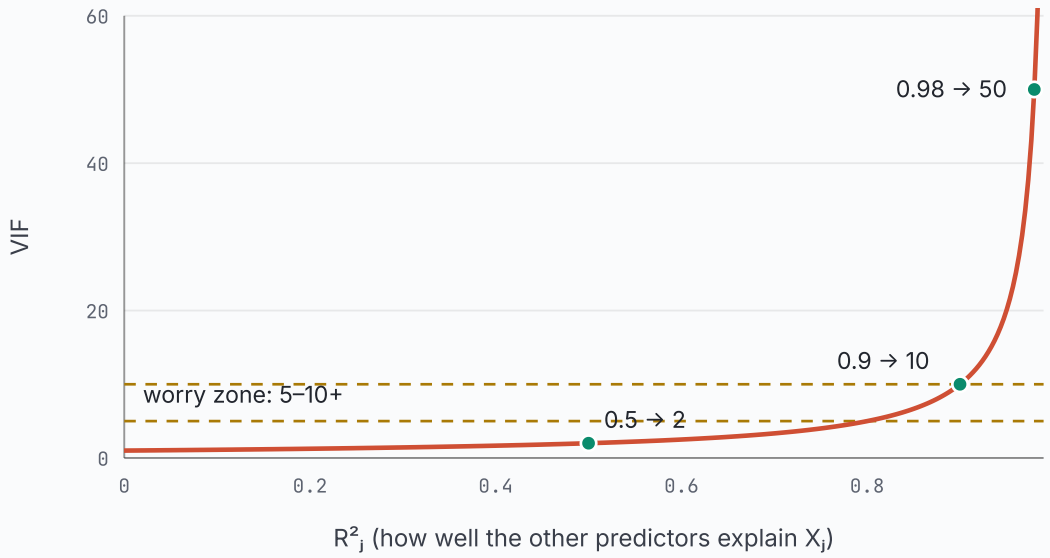
In a simulation with true $\beta_1 = 2$, uncorrelated predictors give estimates clustered around 2; predictors correlated at 0.97 give estimates scattered widely, even though the signal never changed.

Variance inflation factor

VIF

$$\mathrm{VIF}_j = \frac{1}{1 - R_j^2}$$

R_j^2 is the R^2 from regressing predictor X_j on all the *other* predictors (not on Y).



VIF explodes as a predictor becomes predictable from the others. The variance of its coefficient is inflated by this factor.

Rule of thumb: worry above 5–10. A related diagnostic is the **condition number** of $X^\top X$: huge values mean near-singularity. VIF is a diagnostic, not a decision rule.

WORKED EXAMPLE

A predictor regressed on the others has $R_j^2 = 0.95$. $\mathrm{VIF} = 1/(1-0.95) = 20$: its coefficient's variance is 20 times what it would be with uncorrelated predictors, so its SE is $\sqrt{20} \approx 4.5$ times larger.

What it does and doesn't do

MULTICOLLINEARITY DOES	IT DOES NOT NECESSARILY
Inflate standard errors (predictors look insignificant)	Harm predictions on similar data
Make coefficients unstable, sometimes wrongly signed	Reduce overall R^2
Make interpretation unreliable	Mean the predictors are irrelevant

Remedies: drop or combine redundant predictors; collect more spread-out data; or regularise. Ridge regression (5.4) shares the credit between correlated predictors instead of giving them huge opposite-sign coefficients.

HOW IT'S ASKED

"Two sensors correlate at 0.98. One has a large p-value although it relates to the response on its own. Why?" Given the other sensor, it adds little unique information, so its coefficient is poorly determined (inflated SE, high VIF). Insignificance here isn't irrelevance.

Cheatsheet

VIF

$\mathrm{VIF}_j = \frac{1}{1-R_j^2}$, R_j^2 from X_j on other predictors. Worry > 5–10.

Multicollinearity effects

Inflated SEs, unstable/wrong-sign coefficients. Predictions often fine. Fix: drop/combine, more data, ridge.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/linear-regression/multicollinearity-and-vif

Section 4

Logistic Regression and Classification

How classifiers produce probabilities, how probabilities become decisions, and how to evaluate classification beyond accuracy.

4.1 Regression versus Classification · 7 min

4.2 Odds, Log-odds and the Sigmoid Function · 9 min

4.3 The Logistic Regression Model · 10 min

4.4 Maximum Likelihood Estimation · 9 min

4.5 Cross-entropy / Log-loss · 7 min

4.6 The Confusion Matrix · 7 min

4.7 Accuracy, Precision, Recall and Specificity · 10 min

4.8 Sensitivity and Specificity Trade-offs · 7 min

4.9 Threshold Selection and Classification Costs · 9 min

4.10 ROC Curve and AUC · 10 min

4.11 Class Imbalance · 8 min

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.

?

SUPERVISED LEARNING · THE FORK

A number or a category?

Is the target a quantity, or one of a few classes?

Lessons here: 4.1 · feeds into Choose the model, Choose the model

C1

SUPERVISED LEARNING · CLASSIFICATION LANE · STEP C1

Choose the model

How do I turn inputs into a class, or a probability of one?

Lessons here: 4.2, 4.3 · feeds into Turn predictions into decisions

C2

SUPERVISED LEARNING · CLASSIFICATION LANE · STEP C2

Fit it

Which coefficients make the observed labels most probable?

Lessons here: 4.4, 4.5 · feeds into Control complexity

C3

SUPERVISED LEARNING · CLASSIFICATION LANE · STEP C3

Score it

How good is it at catching the class that matters?

Lessons here: 4.6, 4.7, 4.10, 4.11 · feeds into Turn predictions into decisions, Evaluate honestly

6

SUPERVISED LEARNING · STEP 6

Turn predictions into decisions

Where do I draw the line, given what each mistake costs?

Lessons here: 4.8, 4.9

Regression versus Classification

When the outcome is a category, we model the probability of each class. Why ordinary linear regression is a poor fit for yes/no outcomes, and why logistic "regression" is a classifier.

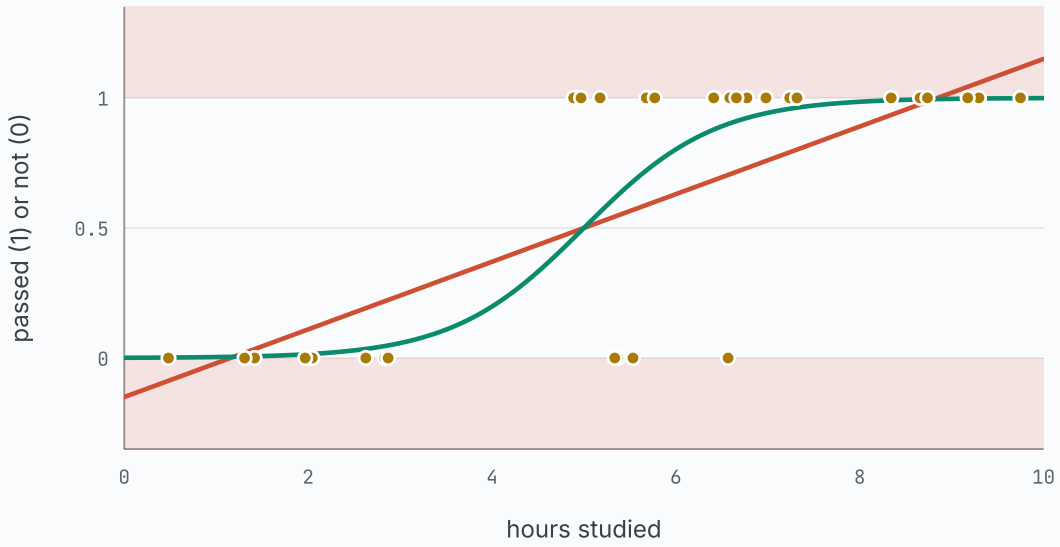
Categories, not quantities

Classification predicts a category: spam or not, diabetic or not, which digit. For two classes we code $Y \in \{0, 1\}$, where 1 is the class of interest ("positive"). The natural target is the **probability** of class 1:

$$P(X) = \Pr(Y = 1 \mid X)$$

A class prediction then comes from comparing that probability with a threshold (lesson 4.9).

Why not just use linear regression?



Fitting a straight line to 0/1 outcomes produces "probabilities" outside [0, 1] (shaded). The S-shaped logistic curve doesn't.

- **Impossible predictions:** a line goes below 0 and above 1, which can't be probabilities.
- **Broken assumptions:** when Y is only ever 0 or 1, the noise can't be "constant variance, roughly normal".
- **Fake ordering:** with more than two unordered classes (red, green, blue), coding them 1, 2, 3 imposes a meaningless order and spacing.

Why is it called logistic *regression*?

Because it **is** a regression, of the **log-odds** on the predictors: $\ln \frac{P}{1-P} = \beta_0 + \beta_1 X_1 + \dots$ (4.3). It outputs a continuous probability. It becomes a **classifier** once you add a decision rule like "predict 1 if $P > 0.5$". The name describes the model; the use is classification.

	LINEAR REGRESSION	LOGISTIC REGRESSION
Outcome	A number	A class (via a probability)
Models	$\mathbb{E}[Y \mid X]$ directly	Log-odds of $Y = 1$
Output range	$(-\infty, \infty)$	$(0, 1)$
Fitted by	Least squares (closed form)	Maximum likelihood (numerical)
Evaluated by	RMSE, MAE, R^2	Confusion matrix, sensitivity, specificity, AUC, log loss

Common doubts

FIRST PASS **Why is logistic regression used for classification when it's called regression?**

The model itself is a regression: it fits a linear function of the predictors to the **log-odds** of the outcome and outputs a probability between 0 and 1. Classification happens in an extra step, when you turn that probability into a class with a threshold (e.g. "predict 1 if $p > 0.5$"). So: regression for the probability, classification for the decision.

Cheatsheet

Linear vs logistic regression

Linear: number, models $\mathbb{E}[Y|X]$, OLS, RMSE/R^2.
Logistic: class via $P(Y = 1|X) \in (0, 1)$, linear in log-odds, MLE, confusion matrix/AUC.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/regression-versus-classification

Odds, Log-odds and the Sigmoid Function

Probability, odds and log-odds are three views of the same thing. How to convert between them by hand, and how the sigmoid function squashes any score into a probability.

Odds: betting language

$$\text{odds} = \frac{p}{1-p}$$

$p = 0.75$ gives odds of $0.75/0.25 = 3$: "3 to 1 in favour". $p = 0.5$ is odds 1 (even money). $p = 0.2$ is odds 0.25 ("1 to 4"). Odds run from 0 to ∞.

Log-odds (the logit)

$$\text{logit}(p) = \ln \frac{p}{1-p} \in (-\infty, \infty)$$

Taking the log stretches odds over the whole real line, which is perfect for a linear model: any value of $\beta_0 + \beta_1 X$ is a valid log-odds.

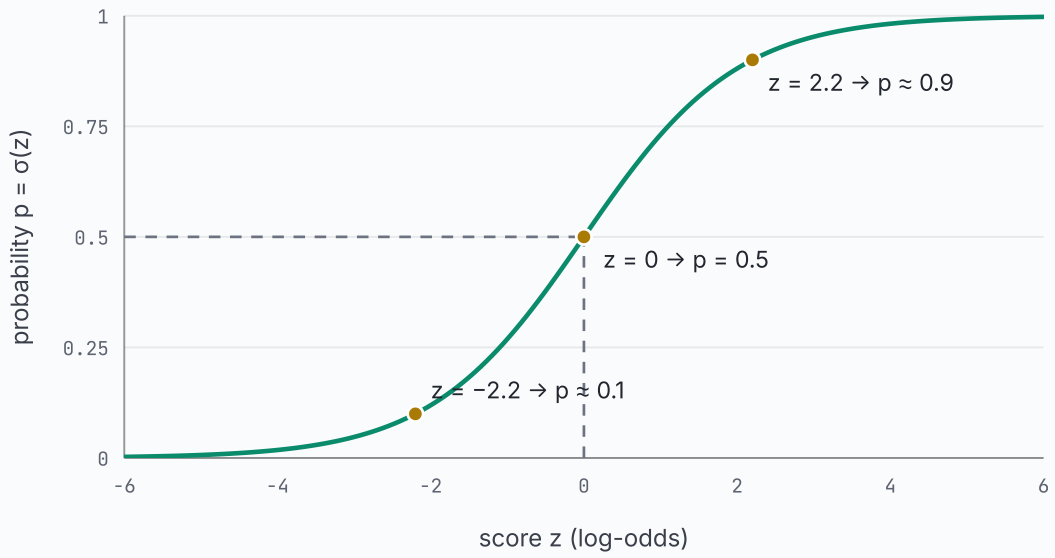
PROBABILITY P	ODDS $P/(1-P)$	LOG-ODDS
0.10	0.111	−2.20
0.20	0.25	−1.39
0.50	1	0
0.75	3	1.10
0.80	4	1.39
0.90	9	2.20

Notice the symmetry: p and $1-p$ have log-odds of equal size and opposite sign.

The sigmoid: back from score to probability

The inverse of the logit is the **logistic (sigmoid) function**:

$$p = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$$



Any real score maps to a probability strictly between 0 and 1. The curve is steepest at z = 0 (p = 0.5) and flattens near 0 and 1.

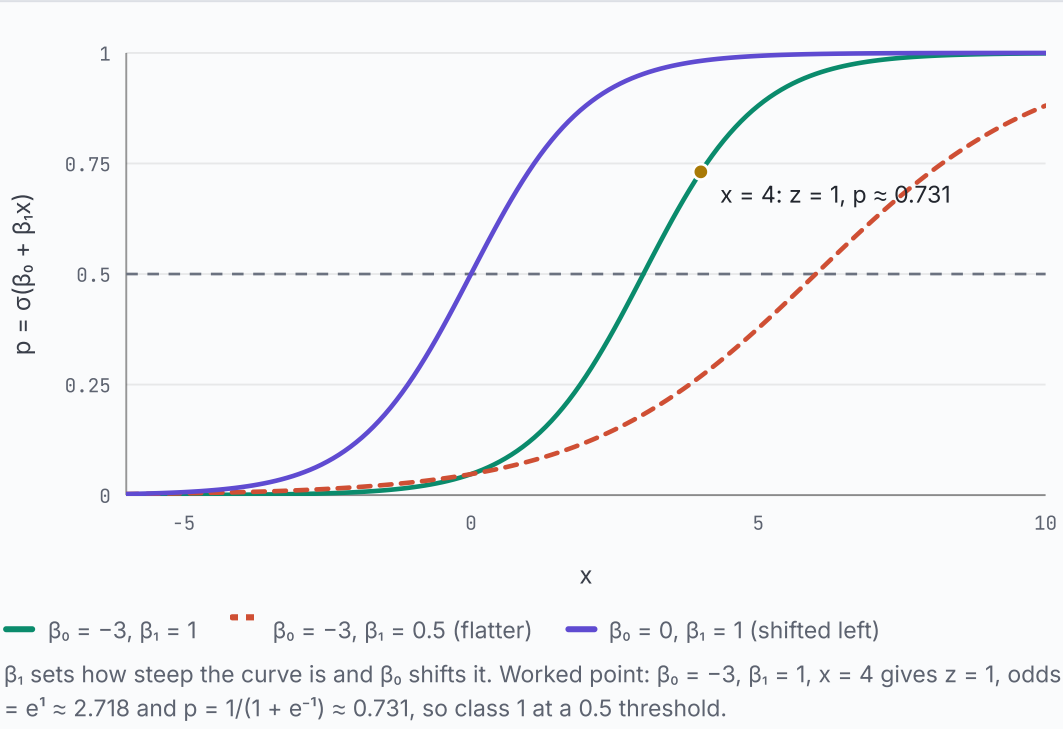
CONVERSIONS YOU MUST BE ABLE TO DO

Probability → odds → log-odds: $p = 0.8 \to$ odds $0.8/0.2 = 4 \to \ln 4 = 1.386$.

Log-odds → probability: $z = -0.6 \to$ odds $e^{-0.6} = 0.549 \to p = 0.549/1.549 = 0.354$. Equivalently $p = 1/(1 + e^{0.6})$.

Handy values: $e^{-1} \approx 0.368$, $e^{0.5} \approx 1.649$, $\ln 2 \approx 0.693$, $\ln 3 \approx 1.099$, $\ln 10 \approx 2.303$.

FROM SCORE TO PROBABILITY



β_1 sets how steep the curve is and β_0 shifts it. Worked point: $\beta_0 = -3$, $\beta_1 = 1$, $x = 4$ gives $z = 1$, odds $= e^1 \approx 2.718$ and $p = 1/(1 + e^{-1}) \approx 0.731$, so class 1 at a 0.5 threshold.

Cheatsheet

Odds & log-odds

odds $= \frac{p}{1-p}$; logit $= \ln \frac{p}{1-p}$. $p = \frac{\text{odds}}{1+\text{odds}}$.

Sigmoid

$\sigma(z) = \frac{1}{1+e^{-z}}$. $z = 0 \to 0.5$; $z = \pm 2.2 \to 0.9/0.1$.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/odds-log-odds-and-sigmoid

The Logistic Regression Model

The model as a linear score passed through a sigmoid, or as linear regression on the log-odds. Interpreting coefficients as odds ratios, computing predictions by hand, and the linear decision boundary.

Two equivalent views

THE MODEL

$$z = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p, \qquad P(X) = \Pr(Y = 1 \mid X) = \frac{1}{1 + e^{-z}}$$

$$\iff \qquad \ln \frac{P(X)}{1 - P(X)} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

View 1 squashes a linear score into a probability. View 2 is a linear model of the log-odds.

Interpreting coefficients

INTERPRETATION

β_j is the change in **log-odds** of $Y = 1$ for a one-unit increase in X_j, holding the other predictors fixed. Equivalently, e^{β_j} is the factor by which the **odds** are multiplied: the **odds ratio**.

- $\beta_j > 0 \Rightarrow e^{\beta_j} > 1$: higher X_j raises the probability of $Y = 1$.
- $\beta_j < 0 \Rightarrow e^{\beta_j} < 1$: higher X_j lowers it. $\beta_j = 0$: odds ratio 1, no effect.
- The effect on the **probability** is not constant: it depends on where you start (biggest near $p = 0.5$, tiny near 0 or 1).

WORKED EXAMPLE: CUSTOMER CHURN

log-odds $= -2 + 0.8x_1 - 0.5x_2$, where x_1 = number of complaints and x_2 = years as a customer.

Odds ratios: $e^{0.8} = 2.23$ and $e^{-0.5} = 0.61$. Each extra complaint multiplies the odds of churning by about 2.2, holding tenure fixed. Each extra year multiplies them by 0.61, a 39% reduction.

Prediction for 3 complaints, 2 years: $z = -2 + 2.4 - 1.0 = -0.6$, so $p = 1/(1 + e^{0.6}) = 0.354$. At threshold 0.5 we predict "no churn".

Probability isn't additive: going to 4 complaints moves z from -0.6 to 0.2 and p from 0.354 to 0.550 (+0.196). The odds rose by the same factor, 2.23, but the probability change depends on the starting point.

What does a predicted 0.85 mean?

A predicted $P(X) = 0.85$ means: among cases like this one, the model estimates that **85% belong to class 1**. It's a probability of class membership, not an 85% "amount" of anything. A patient with $p = 0.85$ for diabetes isn't "85% diabetic"; the model gives an 85% chance that they are diabetic.

The decision boundary is linear

With the default rule "predict 1 if $\hat{P} > 0.5$", and since $P > 0.5 \iff z > 0$, the boundary is where the linear score is zero:

$$\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0$$

That's a straight line in 2-D, so logistic regression is a **linear classifier** even though its probability curve is S-shaped. For the churn model the boundary is $0.8x_1 - 0.5x_2 = 2$. Changing the threshold to t shifts the line parallel to itself (to $z = \ln \frac{t}{1-t}$); it doesn't change the model.

THREE COMMON ERRORS

1. "A one-unit increase raises the probability by β." No: it adds β to the log-odds.
2. Treating an odds ratio of 2 as "twice as likely". It means twice the odds; the probability ratio is smaller unless p is small.
3. Reading coefficients causally from observational data, or comparing magnitudes across predictors with different units.

Common doubts

FIRST PASS **What does a predicted probability of 0.85 actually mean? Is it 85% of something?**

It's the model's estimated probability that this case **belongs to class 1**. Among many cases with the same predictor values, it expects about 85% to be positive. It isn't a measurement like "85% progressed" or "85% severity". The output is about class membership; the class itself is still yes or no.

Cheatsheet

Logistic coefficient

β_j: change in log-odds per unit X_j. e^{β_j}: odds ratio (multiplies the odds). Not a constant change in probability.

Decision boundary

Predict 1 if $\hat{p} > t \iff \hat{\beta}_0 + \hat{\beta}^\top x > \ln \frac{t}{1-t}$ (0 when $t = 0.5$). Linear.

Odds ratio 2 ≠ "twice as likely"

It doubles the **odds**. The probability ratio is smaller unless p is small.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/logistic-regression-model

Maximum Likelihood Estimation

How logistic regression is fitted: choose coefficients that make the observed labels as probable as possible. The likelihood, the log-likelihood, a worked example, and why there is no closed form.

The idea

Unlike OLS there's no formula for logistic regression's coefficients. Instead we use **maximum likelihood estimation (MLE)**: choose β to make the labels we actually observed as probable as possible under the model.

Likelihood for one observation

With predicted probability $P_i = P(x_i)$:

$$\Pr(Y_i = y_i) = P_i^{y_i}(1 - P_i)^{1-y_i}$$

This is just a compact way to write "P_i if $y_i = 1$, and $1 - P_i$ if $y_i = 0$".

All observations, and the log

For independent observations, the likelihood multiplies them. Products of many small numbers are awkward, so we take logs. The log is increasing, so the maximiser doesn't change:

LOG-LIKELIHOOD

$$\ell(\beta) = \sum_{i=1}^{n} \left[y_i \ln P_i + (1 - y_i) \ln(1 - P_i) \right]$$

Each term is the log of the probability the model gave to what actually happened. ℓ is always ≤ 0, and closer to 0 is better.

WORKED EXAMPLE

Labels $y = (1, 0, 1)$; predicted probabilities $p = (0.9, 0.2, 0.6)$.

Probabilities of what happened: 0.9, $1 - 0.2 = 0.8$, 0.6.

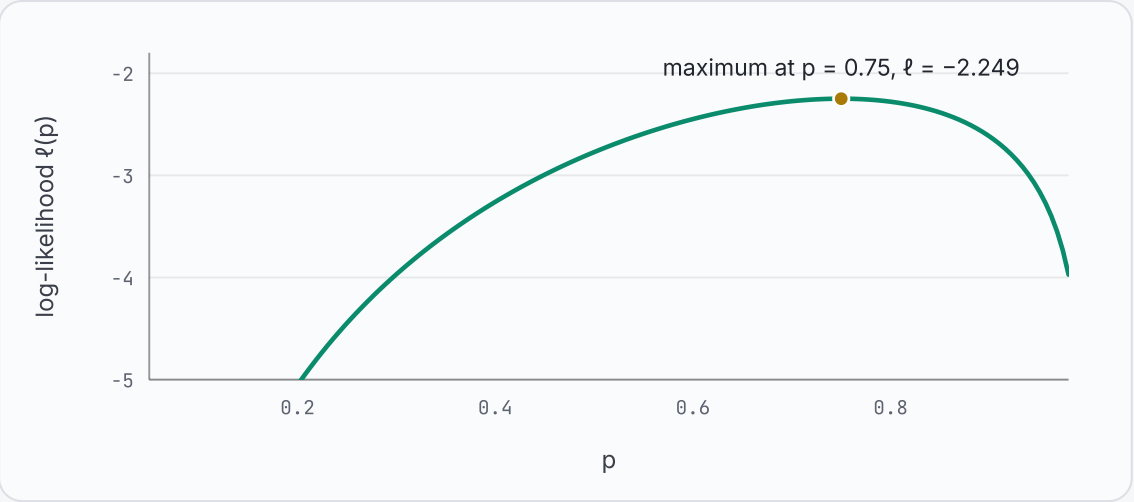
$\ell = \ln 0.9 + \ln 0.8 + \ln 0.6 = -0.105 - 0.223 - 0.511 = -0.839$.

Mean cross-entropy $= 0.839/3 = 0.280$ (lesson 4.5).

THE SIMPLEST MLE: NO PREDICTORS

Labels $y = (1, 1, 0, 1)$ and a single probability p for everyone. $\ell(p) = 3 \ln p + \ln(1 - p)$.

Derivative: $3/p - 1/(1 - p) = 0 \Rightarrow p = 3/4$. The MLE is the proportion of ones, which matches intuition.



Fitting in practice

- Maximise $\ell(\beta)$, which is the same as minimising $-\ell(\beta)$, which is the same as minimising the cross-entropy loss. So logistic regression is ERM with cross-entropy.
- There's **no closed form**: setting the derivative to zero gives equations in which β sits inside the sigmoid. They're solved numerically (gradient descent, Newton's method).
- The log-likelihood is **concave**, so there are no bad local optima: the numerical solver finds the global best.

Testing coefficients (Wald z-test)

As with OLS, test $H_0 : \beta_j = 0$ with $z_j = \hat{\beta}_j / \mathrm{SE}(\hat{\beta}_j)$, which is approximately standard normal for large samples. Reject at 5% if $|z_j| > 1.96$. A 95% CI for the odds ratio is $\exp\left(\hat{\beta}_j \pm 1.96\,\mathrm{SE}\right)$.

WORKED EXAMPLE

$\hat{\beta}_1 = 0.8$, SE $= 0.3$: $z = 2.67$, $p \approx 0.008$, significant. CI for β_1: $0.8 \pm 0.59 = (0.21, 1.39)$; for the odds ratio: $(e^{0.21}, e^{1.39}) = (1.23, 4.01)$. It excludes 1, consistent with the test.

PERFECT SEPARATION

If a predictor perfectly splits the classes, the likelihood keeps increasing as the coefficient grows, so the MLE diverges (huge $\hat{\beta}$ and SEs). Regularisation (5.x) fixes this.

Common doubts

GOING DEEPER **Why does logistic regression need gradient descent when OLS has a formula?**

Both set the **gradient** (the vector of slopes of the objective, one per coefficient) to zero. For OLS the objective is quadratic, so the gradient is linear in β and the equations can be solved directly: $\hat{\beta} = (X^\top X)^{-1} X^\top y$.

For logistic regression, β sits inside the sigmoid, so the equations can't be rearranged into a formula. Instead you start somewhere and repeatedly step **against** the gradient of the cross-entropy, $\beta \leftarrow \beta - \eta \nabla J(\beta)$, until it stops improving. The objective is convex, so this finds the single best β. The one exception is perfectly separable data, where the coefficients keep growing forever.

Cheatsheet

Logistic log-likelihood

$\ell(\beta) = \sum_i [y_i \ln P_i + (1 - y_i) \ln(1 - P_i)]$. Maximise ⇔ minimise cross-entropy.

Wald test

$z = \hat{\beta} / \mathrm{SE}$, reject if $|z| > 1.96$. OR CI: $e^{\hat{\beta} \pm 1.96 \mathrm{SE}}$.

OLS vs MLE fitting

OLS: minimise RSS, closed form. Logistic MLE: maximise log-likelihood, numerical (concave, no bad local optima).

Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/maximum-likelihood-estimation

Cross-entropy / Log-loss

Cross-entropy is the negative average log-likelihood. Why it punishes confident wrong predictions so hard, how to compute it, and how it differs from accuracy as a measure.

BINARY CROSS-ENTROPY (LOG-LOSS)

$$\mathrm{CE} = -\frac{1}{n}\sum_{i=1}^{n}\Big[y_i \ln p_i + (1-y_i)\ln(1-p_i)\Big] = -\frac{\ell(\beta)}{n}$$

For each observation, only one term is active: $-\ln p_i$ if $y_i = 1$, and $-\ln(1-p_i)$ if $y_i = 0$. So the loss is "minus the log of the probability you gave to the truth".



Loss is near zero when the model is confident and right, and shoots towards infinity when it is confident and wrong.

P GIVEN TO THE TRUTH	0.99	0.9	0.5	0.1	0.01
$-\ln p$	0.010	0.105	0.693	2.303	4.605

Why use it to fit classifiers?

- It is the maximum-likelihood loss for a yes/no (Bernoulli) outcome.
- It rewards good **probabilities**, not just correct classes. Predicting 0.51 and 0.99 for a true positive both count as "correct" for accuracy, but cross-entropy prefers 0.99.
- It is smooth and, for logistic regression, convex: easy to optimise.

Cross-entropy vs accuracy

	ACCURACY	CROSS-ENTROPY
Uses	Hard class predictions	Predicted probabilities
Depends on threshold	Yes	No
Rewards confidence	No	Yes (and punishes overconfidence)
Good for fitting	No (not smooth)	Yes

SIGN CONVENTIONS

The log-likelihood ℓ has no minus sign and is **maximised**. Cross-entropy is $-\ell/n$ and is **minimised**. Both are correct; don't mix them in one sentence.

Cheatsheet

Log-loss values

$-\ln p$: 0.99→0.01, 0.9→0.105, 0.5→0.693, 0.1→2.303, 0.01→4.6.

Sign of the log-likelihood

ℓ is maximised (≤ 0). Cross-entropy $= -\ell/n$ is minimised (≥ 0).

Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/cross-entropy-log-loss

The Confusion Matrix

Once you pick a threshold, every prediction is a true positive, false positive, true negative or false negative. How to build the table, read it, and which errors are "misses" and which are "false alarms".

Four outcomes

Fix a threshold and classify every observation. Choose one class as the **positive** class: the important one (disease present, fraud, alert needed). Each prediction then falls into one of four cells:

	PREDICTED POSITIVE	PREDICTED NEGATIVE
Actual positive	**TP** true positive (a hit)	**FN** false negative (a **miss**)
Actual negative	**FP** false positive (a **false alarm**)	**TN** true negative

Statisticians also call FP a **Type I error** and FN a **Type II error**. Row totals are the actual class sizes ($P = TP + FN$, $N = FP + TN$); column totals are how many the model flagged.

CHECK THE LAYOUT

Some books and software put predictions in rows and actuals in columns. Always read the labels before computing anything.

Building one

WORKED EXAMPLE

Six patients, threshold 0.5:

PATIENT	p	PREDICTED	ACTUAL	CELL
A	0.92	+	+	TP
B	0.71	+	−	FP
C	0.55	+	+	TP
D	0.40	−	+	FN
E	0.15	−	−	TN
F	0.08	−	−	TN

So TP = 2, FP = 1, FN = 1, TN = 2.

Which error is worse?

It depends on the decision, never on the maths. In cancer screening, a miss (FN) can be fatal while a false alarm (FP) means an extra test: FNs are worse. In spam filtering, a false alarm hides a real email from you: FPs are worse. This judgement drives the choice of metric (4.7) and threshold (4.9).

Cheatsheet

Confusion matrix

Rows actual, columns predicted. TP hit · FN **miss** · FP **false alarm** · TN. FP = Type I, FN = Type II.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/confusion-matrix

Accuracy, Precision, Recall and Specificity

Every metric you can compute from a confusion matrix, what question each one answers, a full worked example, and why accuracy can be dangerously misleading.

THE METRICS

$$\text{Accuracy} = \frac{TP + TN}{n}, \qquad \text{Error rate} = \frac{FP + FN}{n} = 1 - \text{Accuracy}$$

$$\text{Sensitivity (recall, TPR)} = \frac{TP}{TP + FN} \qquad \text{Specificity (TNR)} = \frac{TN}{TN + FP}$$

$$\text{FPR} = \frac{FP}{FP + TN} = 1 - \text{specificity} \qquad \text{Precision (PPV)} = \frac{TP}{TP + FP}$$

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

METRIC	QUESTION IT ANSWERS	DENOMINATOR
Sensitivity / recall	Of the actual positives, what fraction did we catch?	Actual positives
Specificity	Of the actual negatives, what fraction did we correctly clear?	Actual negatives
Precision	Of the cases we flagged, what fraction were truly positive?	Predicted positives
Accuracy	What fraction of all predictions were right?	Everything

SMOKE ALARM

Sensitivity: when there's a fire, how often does the alarm ring? **Specificity:** when there's no fire, how often does it stay quiet? **Precision:** when the alarm rings, how often is there really a fire? A hair-trigger alarm has high sensitivity and low specificity; a sleepy one the reverse.

WORKED EXAMPLE: ALL METRICS FROM ONE TABLE

$n = 1000$: TP = 40, FN = 10, FP = 30, TN = 920.

- Accuracy $= 960/1000 = 0.96$
- Sensitivity $= 40/50 = 0.80$
- Specificity $= 920/950 = 0.968$; FPR $= 30/950 = 0.032$
- Precision $= 40/70 = 0.571$
- $F_1 = \frac{2(0.571)(0.80)}{0.571+0.80} = 0.667$

Baseline: predicting "negative" for everyone gives accuracy $950/1000 = 0.95$ with sensitivity 0. Our 0.96 barely beats it. Precision (0.57) is much lower than sensitivity or specificity here, because positives are rare.

Prevalence matters for precision

Sensitivity and specificity condition on the **actual** class, so they don't depend on how common positives are. Precision conditions on the **predicted** class, so it does.

THE RARE-DISEASE SURPRISE

Prevalence 1%, sensitivity 90%, specificity 90%, 1,000 people. Sick: 10, of whom 9 test positive. Healthy: 990, of whom 99 test positive (10%). Total positives 108, of which only 9 are sick: **precision = 9/108 = 8.3%**. A "90% accurate" test is wrong more than 9 times out of 10 when it says positive.

ACCURACY HIDES RARE CLASSES

A fraud model with 99.5% accuracy on data with 0.5% fraud may simply predict "not fraud" for everything. Always compare with the majority-class baseline and report class-specific metrics.

Cheatsheet

Sensitivity / recall / TPR

$\frac{TP}{TP+FN}$: share of actual positives caught.

Specificity / TNR

$\frac{TN}{TN+FP}$: share of actual negatives cleared. FPR $= 1-$ spec.

Precision / PPV

$\frac{TP}{TP+FP}$: share of flagged cases truly positive. Depends on prevalence.

Accuracy & F1

Acc $= \frac{TP+TN}{n}$. $F_1 = \frac{2PR}{P+R}$.

High accuracy, rare class

Compare with the majority-class baseline. Report sensitivity, specificity, precision.

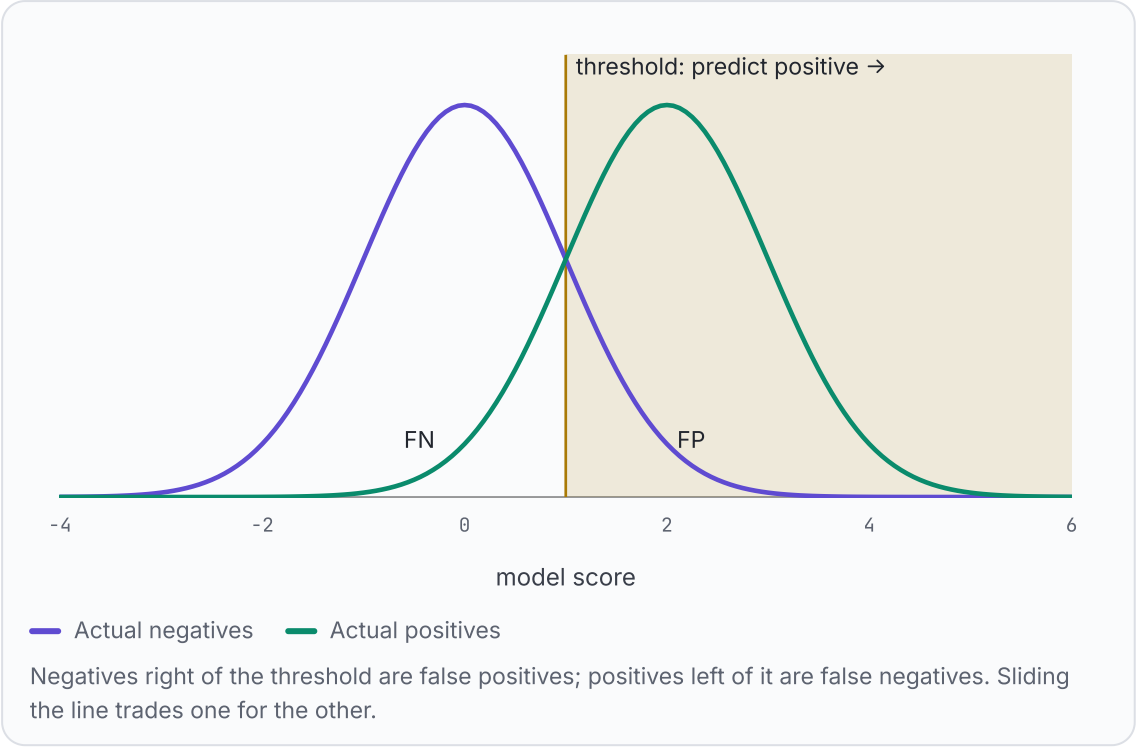
Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/accuracy-precision-recall-and-specificity

Sensitivity and Specificity Trade-offs

Moving the threshold trades misses for false alarms. Why you can't raise sensitivity and specificity together with the same model, and how to reason about which to favour.

Two overlapping distributions

A classifier gives each case a score (for logistic regression, $\hat{p}$). Positives tend to score higher than negatives, but the two distributions overlap. Wherever you put the threshold, some negatives land above it (false positives) and some positives below it (false negatives).



Negatives right of the threshold are false positives; positives left of it are false negatives. Sliding the line trades one for the other.

Which way does each move?

THRESHOLD	FLAG	SENSITIVITY	SPECIFICITY	FN	FP
Lower (e.g. 0.5 → 0.2)	More cases	↑	↓	↓	↑
Raise (e.g. 0.5 → 0.8)	Fewer cases	↓	↑	↑	↓

At the extremes: threshold 0 flags everyone (sensitivity 1, specificity 0); threshold 1 flags no one (sensitivity 0, specificity 1). Neither is useful.

ONLY A BETTER MODEL MOVES BOTH

With a fixed model, the threshold just slides along a trade-off. To improve sensitivity *and* specificity together you need a model whose score distributions overlap less: better features, more data, a better method. That's what the ROC curve and AUC measure (4.10).

Precision and recall trade off too

Lowering the threshold usually raises recall (sensitivity) and lowers precision, since you flag more marginal cases. Precision–recall is the more informative pair when positives are rare and true negatives are plentiful and uninteresting.

HOW IT'S ASKED

"A team lowers the threshold until sensitivity is 100%, then boasts about it. What's the catch?" Flagging nearly everyone achieves perfect sensitivity at the cost of a huge number of false positives (specificity and precision collapse). Report both sides of the trade-off.

Cheatsheet

Threshold effects

Lower t → more flagged → sens ↑, spec ↓, FN ↓, FP ↑, precision usually ↓. Raise t → the reverse.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/sensitivity-and-specificity-trade-offs

Threshold Selection and Classification Costs

When a miss and a false alarm cost different amounts, choose the threshold that minimises expected cost. The theory, a worked table, and how to choose it without leaking the test set.

0.5 is a choice, not a law

The default "predict 1 if $\hat{p} > 0.5$" minimises the number of mistakes when both kinds cost the same. When they don't, pick the threshold that minimises **expected cost**:

$$\text{Cost}(t) = c_{FP} \cdot FP(t) + c_{FN} \cdot FN(t)$$

The theoretical threshold

For a single case with (well-calibrated) probability p: alerting costs $(1-p)\,c_{FP}$ in expectation; not alerting costs $p\,c_{FN}$. Alert when the second is bigger:

COST-OPTIMAL THRESHOLD

$$\text{alert if} \quad p > \frac{c_{FP}}{c_{FP} + c_{FN}} \qquad (\text{equal costs} \Rightarrow 0.5)$$

If a miss is costlier than a false alarm, the threshold moves **down**, so you flag more cases.

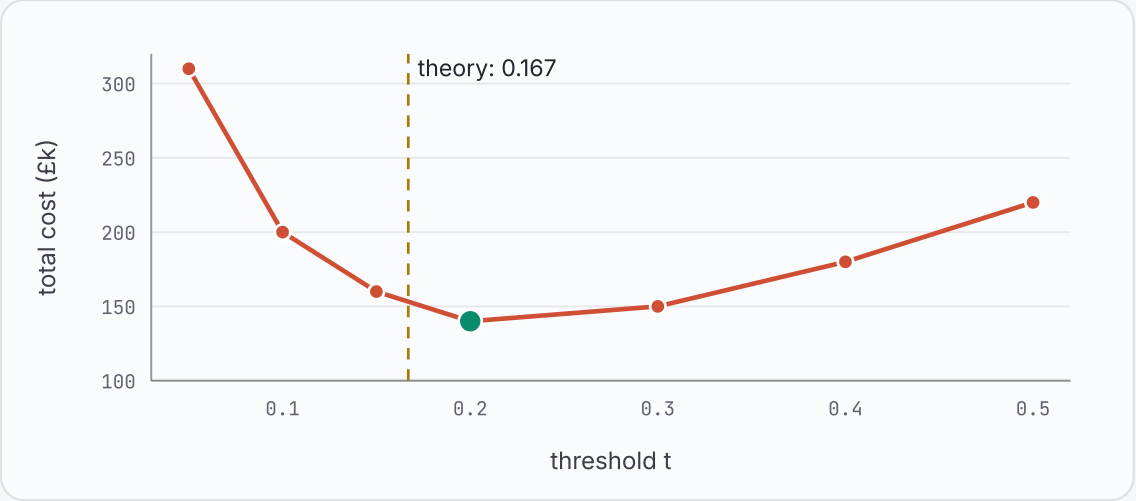
WORKED EXAMPLE: POLLUTION ALERTS

A false alarm costs £2,000; a missed dangerous reading costs £10,000. Theory: alert if $p > 2000/12000 = 0.167$.

Empirically, on 1,000 validation cases:

THRESHOLD	FP	FN	COST $= 2000\,FP + 10000\,FN$
0.5	10	20	£220,000
0.2	40	6	**£140,000** (best)
0.1	90	2	£200,000

The lowest cost is at 0.2, close to the theoretical 0.167. Going lower produces too many false alarms.



The same logic applies when the score is a regression prediction: e.g. alert when predicted CO exceeds t, where the real event is CO above a legal limit of 5. Because misses are costlier, the best t ends up a little *below* 5: alert slightly early.

MOVING THE THRESHOLD (40 POSITIVES, 160 NEGATIVES)

T	TP	FN	FP	TN	ACCURACY	SENSITIVITY	SPECIFICITY	PRECISION	COST
0.1	40	0	122	38	0.390	1.000	0.237	0.247	£244,000
0.2	38	2	93	67	0.525	0.950	0.419	0.290	£206,000
0.3	37	3	61	99	0.680	0.925	0.619	0.378	£152,000
0.5	28	12	28	132	0.800	0.700	0.825	0.500	£176,000
0.7	16	24	9	151	0.835	0.400	0.944	0.640	£258,000
0.9	5	35	1	159	0.820	0.125	0.994	0.833	£352,000

Cost uses £2,000 per false positive and £10,000 per false negative. The cost-minimising threshold for calibrated probabilities is t* = 2000 / (2000 + 10000) ≈ 0.167, far below 0.5. Try it interactively in the online lesson.

CHOOSE THE THRESHOLD ON VALIDATION DATA

The threshold is a hyperparameter. Choosing the cost-minimising threshold on the test set and reporting the test cost leaks the test set: the reported cost is optimistic. Pick it by cross-validation or on a validation set; test once. Choosing it on training predictions is also optimistic, because the model is overconfident on data it was fitted to.

Cheatsheet

Cost-optimal threshold

Alert if $p > \frac{c_{FP}}{c_{FP}+c_{FN}}$. Costlier misses → lower threshold.

Expected cost

$\text{Cost}(t) = c_{FP}\,FP(t) + c_{FN}\,FN(t)$; choose t on validation data.

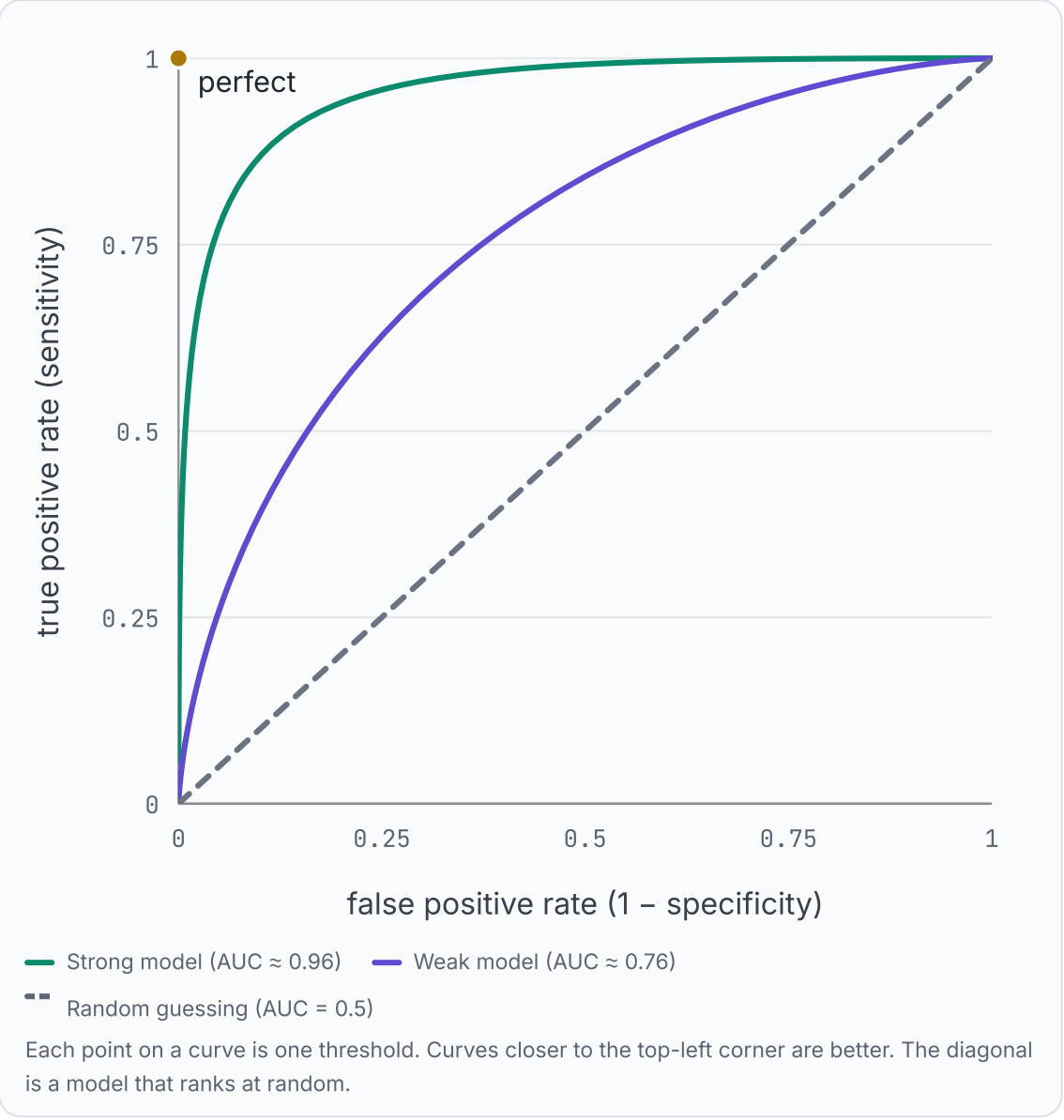
Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/threshold-selection-and-classification-costs

ROC Curve and AUC

The ROC curve shows every sensitivity–specificity trade-off a model can offer. AUC summarises it as the probability a random positive outranks a random negative. How to compute both by hand.

The curve

For every threshold t from 1 down to 0, compute the true positive rate (sensitivity) and the false positive rate (1 – specificity). Plot TPR against FPR. That's the **ROC curve** (receiver operating characteristic).



Each point on a curve is one threshold. Curves closer to the top-left corner are better. The diagonal is a model that ranks at random.

- **Top-left corner** (FPR 0, TPR 1) is perfect.
- **The diagonal** is random guessing: flagging a random fraction of cases catches the same fraction of positives and negatives.
- Moving along a curve = moving the threshold. The curve itself doesn't change with the threshold.

AUC

MEANING

AUC (area under the ROC curve) is the probability that a randomly chosen positive gets a **higher score** than a randomly chosen negative. 0.5 = random ranking; 1.0 = perfect ranking.

AUC BY COUNTING PAIRS

Positives scored $(0.9, 0.6)$; negatives $(0.7, 0.3, 0.2)$. There are $2 \times 3 = 6$ positive–negative pairs. A pair is "correct" if the positive scores higher:

- 0.9 beats all three negatives: 3
- 0.6 beats 0.3 and 0.2 but not 0.7: 2

AUC $= 5/6 = 0.833$. (Ties count as half.)

TRACING THE CURVE FROM THE SAME SCORES

THRESHOLD	FLAGGED	TP	FP	TPR	FPR
0.8	0.9	1	0	0.5	0
0.65	0.9, 0.7	1	1	0.5	1/3
0.5	0.9, 0.7, 0.6	2	1	1	1/3
0.1	all five	2	3	1	1

Raising the threshold lowers both TPR and FPR.

What AUC does and doesn't tell you

AUC TELLS YOU	AUC DOES NOT TELL YOU
How well the model **ranks** positives above negatives	Which threshold to use
Performance across all thresholds (threshold-independent)	Accuracy, precision or cost at your operating point
A fair comparison of rankers, unaffected by class balance	Whether the probabilities are well calibrated

AUC ≠ ACCURACY

An AUC of 0.92 doesn't mean 92% accuracy. And a model with AUC 0.55 isn't necessarily useless at every threshold, though it ranks only slightly better than chance.

Cheatsheet

ROC & AUC

ROC: TPR vs FPR over all thresholds. AUC = P(random positive scored above random negative). 0.5 random, 1 perfect.

AUC by counting

For every (positive, negative) pair, score 1 if the positive ranks higher, 0.5 if tied. AUC = total / (#pos × #neg).

AUC misreadings

AUC 0.9 ≠ 90% accuracy. It doesn't choose a threshold or account for costs.

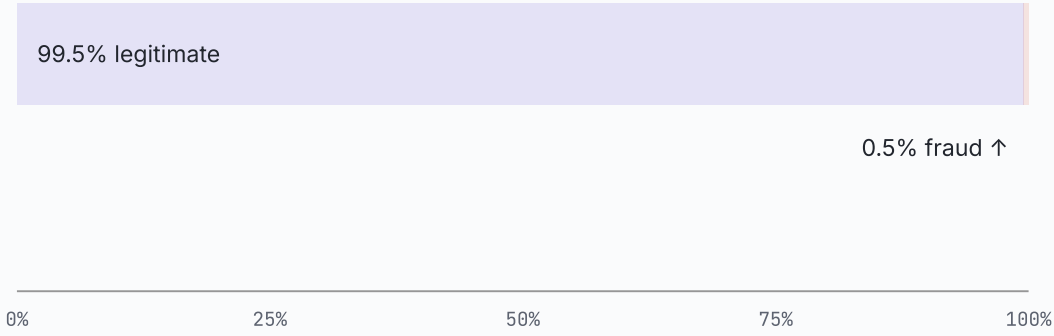
Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/roc-curve-and-auc

Class Imbalance

When the interesting class is rare, a model can score high accuracy while catching nothing. Baselines, the right metrics, threshold moves, oversampling and stratified sampling, and the oversampling traps.

The accuracy trap

When the interesting class is rare (fraud, a rare disease, equipment failure), a classifier minimising overall error can learn to ignore it. With 0.5% fraud, predicting "not fraud" for everyone scores **99.5% accuracy** and catches zero fraud.



Share of transactions by class. A model can be right 99.5% of the time by never predicting the tiny class that matters.

Remedies

1. **Use the right metrics:** sensitivity, specificity, precision, ROC/AUC, a confusion matrix; and compare with the **majority-class baseline**.
2. **Move the threshold** using the costs of each error (4.9).
3. **Oversample the rare class** (or undersample the common one) in the *training* data, so the model pays attention to it.
4. **Use class weights**: count each rare-class error as more expensive in the loss.

A stratified oversampling procedure

1. Split the data into set A (the class of interest) and set B (everything else).
2. **Training set:** randomly take 50% of A, plus an equal number of cases from B. This makes training balanced.
3. **Validation set:** the remaining 50% of A, plus enough of B to **restore the original class ratio**.

Result: a somewhat worse overall error rate, but typically a much better error rate on the cases that matter. Validation stays representative of the real world.

OVERSAMPLING TRAPS

- **Never oversample before splitting.** Copies of the same record land in both training and validation, inflating scores (leakage).
- **Evaluate on data with the original class ratio**, otherwise precision and accuracy are meaningless for deployment.
- After oversampling, predicted probabilities are no longer calibrated to the real prevalence; adjust before using them for cost-based thresholds.

HOW IT'S ASKED

"We balanced the data by duplicating the rare class, then cross-validated. Comment." Duplicates of the same case end up in training and validation folds, so the model is tested on near-copies of its training data. Oversample only within each training fold.

Cheatsheet

Oversampling rule

Split first. Oversample only the training part (inside each CV fold). Evaluate on the original class ratio.

Imbalance toolkit

Right metrics + majority baseline · cost-based threshold · oversample/undersample training data · class weights.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/logistic-regression/class-imbalance

Section 5

Regularisation and Model Selection

How penalties control model complexity, how ridge and LASSO differ, and how the regularisation strength λ shapes the fitted model.

5.1 Why Regularisation is Needed · 8 min

5.2 Model Complexity and Feature Selection · 7 min

5.3 Best Subset and Stepwise Selection · 9 min

5.4 Ridge Regression (L2) · 10 min

5.5 LASSO Regression (L1) · 9 min

5.6 Understanding λ (Lambda) · 8 min

5.7 Ridge versus LASSO · 8 min

5.8 Mathematical Objectives and Worked Calculations · 9 min

5.9 Selecting λ Using Cross-validation · 8 min

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.

5

SUPERVISED LEARNING · STEP 5

Control complexity

How simple can the model be and still capture the pattern?

Lessons here: 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9 · feeds into Turn predictions into decisions

Why Regularisation is Needed

Regularisation trades a little training fit for a simpler, more robust model. The two competing objectives, the benefits of parsimony, and the kinds of datasets where regularisation matters most.

Two goals in tension

Recall the U-shaped curve from 2.3: error on unseen data falls and then rises as complexity grows. **Regularisation** is a way to choose where to sit on that curve. It balances two conflicting objectives:

1. **Performance on the training data:** minimise in-sample error (RSS, misclassification).
2. **Robustness to new data:** minimise model complexity (number of variables, size of coefficients, tree depth, number of clusters). A simpler model is less likely to fit noise.

THE REGULARISED PROBLEM IN ONE LINE

$$\min_{\beta} \left\{ \underbrace{\text{training error}(\beta)}_{\text{fit the data}} + \lambda \cdot \underbrace{\text{Penalty}(\beta)}_{\text{keep it simple}} \right\}$$

The tuning parameter $\lambda \geq 0$ sets the exchange rate between the two.

There are two main families of penalty: **subset selection** (penalise how many coefficients are non-zero) and **shrinkage** (penalise how large they are: ridge and LASSO).

Why simpler models are worth it

BENEFIT	WHY
Performance	More likely to capture the true pattern than quirks of the training data
Robustness	Less likely to change dramatically when new data arrive
Computation	Easier to implement, maintain and re-estimate
Interpretability	Easier to explain and act on
Practicality	Fewer inputs to collect: lower storage, privacy and data costs (fewer sensors to buy)

When it matters most

- **Many predictors relative to observations** ($p \approx n$ or $p > n$). Genomics may have thousands of patients but a million gene features. Plain OLS fails outright when $p + 1 > n$.
- **Highly correlated predictors** (multicollinearity): OLS coefficients become unstable (3.9).
- **Noisy data**: more noise to accidentally fit.

In these cases OLS has high variance or no unique solution, while a regularised model still finds something robust.



Increasing λ raises training error steadily, but test error first falls (less variance) and then rises (too much bias).

Common doubts

FIRST PASS **How does regularisation change the objective?**

It adds a second term. Ordinary least squares minimises $\mathrm{RSS}(\beta)$; ridge minimises $\mathrm{RSS}(\beta) + \lambda \sum_j \beta_j^2$ and LASSO $\mathrm{RSS}(\beta) + \lambda \sum_j |\beta_j|$. The first term rewards fitting the data; the second charges for large coefficients. $\lambda = 0$ gives back OLS, and a large λ pushes the coefficients towards zero. The intercept is not penalised.

Cheatsheet

Regularised objective

$\min_\beta \{\text{training error}(\beta) + \lambda\,\text{Penalty}(\beta)\}$, $\lambda \geq 0$.

When to regularise

$p \approx n$ or $p > n$ · correlated predictors · noisy data.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/regularisation/why-regularisation-is-needed

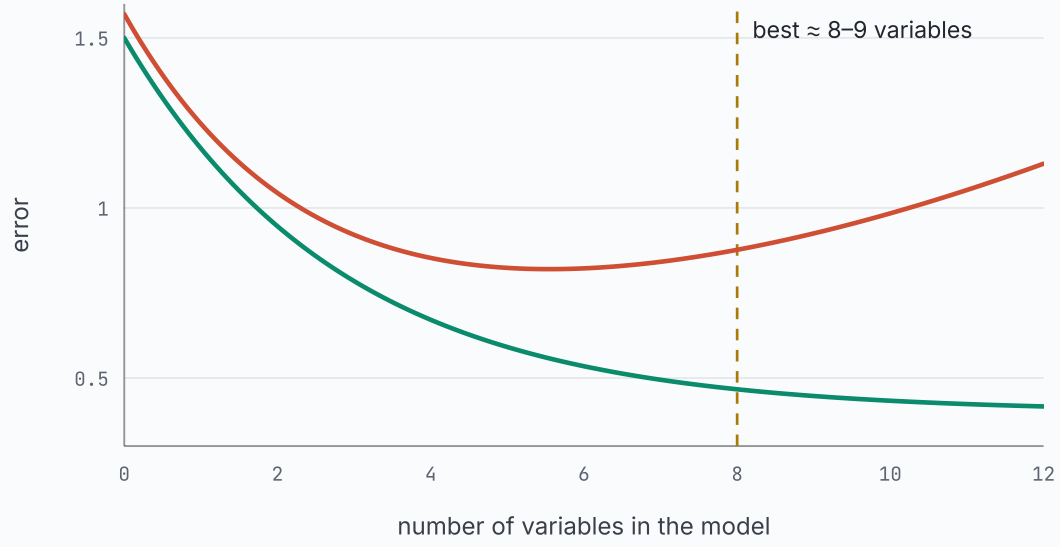
Model Complexity and Feature Selection

Complexity has many dials: number of features, coefficient size, depth, K. Why adding features always helps in-sample, why it eventually hurts out-of-sample, and how feature selection fits in.

Many dials, one idea

MODEL	COMPLEXITY DIAL	MORE COMPLEX WHEN...
Linear regression	Number of predictors; size of coefficients	More variables, bigger $\lvert\beta_j\rvert$
Polynomial regression	Degree	Higher degree
K-nearest neighbours	K	*Smaller* K (more local, wigglier)
Decision tree	Depth	Deeper
K-means	Number of clusters K	Larger K
SVM	Cost C; kernel bandwidth	Larger C

More features: in-sample vs out-of-sample



In-sample, adding a variable can only help. Out-of-sample, the best performance comes at a moderate number of variables; beyond that, the model overfits.

Adding a predictor to OLS can never increase training RSS: at worst its coefficient is set to zero. So training error always prefers the biggest model. Choosing **which** features to keep must use validation data, cross-validation, or a criterion that penalises size (adjusted R^2, AIC/BIC).

Ways to choose features

- **Domain knowledge:** fine when you genuinely know what's relevant.
- **Subset selection:** best subset, forward or backward stepwise (5.3).
- **Shrinkage that selects:** LASSO sets some coefficients exactly to zero (5.5).

THE P-VALUE TRAP

"We removed every variable with $p > 0.05$, refitted, and now all $p < 0.05$." That's not a sound selection procedure. It ignores the uncertainty of the selection itself, makes the final p-values over-confident (they're reported as if no selection happened), and targets in-sample significance rather than out-of-sample prediction. Use CV-based stepwise selection or LASSO.

AND THE LEAKAGE TRAP

Feature selection is a learning step. It must happen inside each CV training fold, never on the full dataset first (lesson 2.8).

Cheatsheet

Select by p-values

Dropping $p > 0.05$ and refitting ignores selection uncertainty and over-states significance. Use CV stepwise or LASSO.

Complexity dials

#features, $|\beta|$, degree, tree depth, small K (K-NN), large K (K-means), large C (SVM).

Online version with the interactive quiz: airesearchlabs.si/learn/ml/regularisation/model-complexity-and-feature-selection

Best Subset and Stepwise Selection

Best subset selection tries every combination of predictors; forward and backward stepwise approximate it greedily. The algorithms step by step, how many models each fits, and a worked forward-stepwise example.

The L0 penalty

Subset selection penalises the **number** of non-zero coefficients:

$$\min_{\beta} \ \mathrm{RSS}(\beta) + \lambda \, \|\beta\|_0, \qquad \|\beta\|_0 = \#\{j : \beta_j \neq 0\}$$

Best subset selection

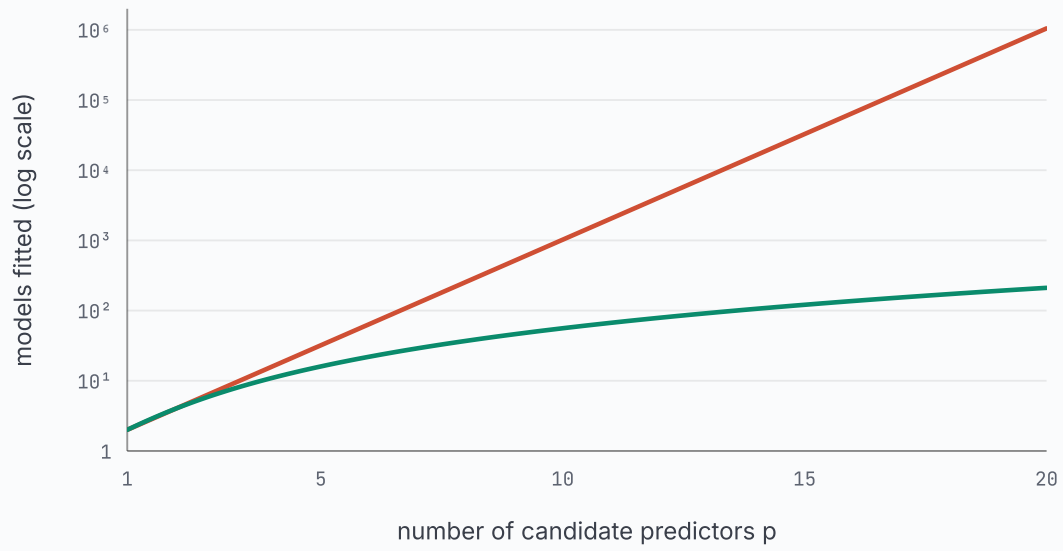
1. For each size $k = 0, 1, \ldots, p$: fit all $\binom{p}{k}$ models with exactly k predictors and keep the one with the lowest training RSS. Call it M_k.
2. Choose among $M_0, \ldots, M_p$ using a measure that accounts for complexity: validation or CV error, adjusted R^2, AIC/BIC. Not training RSS, which always favours M_p.

Benefit: directly implements parsimony and finds the best subset of each size. **Problem:** it fits 2^p models. $p = 10$: 1,024. $p = 20$: over a million. $p = 64$: hopeless.

Greedy approximations

FORWARD STEPWISE	BACKWARD STEPWISE
1. Start with M_0 (mean only). 2. For $k = 0, \ldots, p-1$: try adding each remaining predictor; keep the one that most improves the fit, giving M_{k+1}. 3. Choose the best of $M_0 .. M_p$ by validation/CV.	1. Start with the full model M_p. 2. For $k = p, \ldots, 1$: try removing each predictor; keep the removal that hurts least, giving M_{k-1}. 3. Choose the best size by validation/CV.

Models fitted: $1 + p + (p-1) + \cdots + 1 = 1 + \frac{p(p+1)}{2}$. For $p = 10$: 56 instead of 1,024; for $p = 20$: 211 instead of 1,048,576.



Best subset grows exponentially; stepwise grows only quadratically. In one benchmark, best subset took 163 s for 9 variables and was infeasible at 10, while stepwise handled 64 in milliseconds.

Trade-off: stepwise isn't guaranteed to find the best subset. Once forward stepwise adds a variable, it never removes it, even if a later combination would be better without it.

WORKED EXAMPLE: FORWARD STEPWISE BY HAND

Three candidate predictors A, B, C. TSS $= 100$.

- **Step 0:** M_0 (mean only): RSS $= 100$.
- **Step 1:** one-variable models: A 60, B 45, C 80. Keep B: $M_1 = \{B\}$.
- **Step 2:** add one more to B: A+B 30, B+C 40. Keep $M_2 = \{A, B\}$.
- **Step 3:** $M_3 = \{A, B, C\}$: RSS 28.

Training RSS keeps falling (100, 45, 30, 28), as it always does. Now compare by **validation RMSE**: 10.0, 7.0, 5.5, 5.8. Choose $M_2 = \{A, B\}$.

Models fitted: $1 + 3 + 2 + 1 = 7$, versus $2^3 = 8$ for best subset. The gap grows explosively with p.

Cheatsheet

Forward stepwise

Start M_0. Repeatedly add the predictor that most lowers RSS → $M_1..M_p$. Choose the size by validation/CV.

Best subset

For each k, best of $\binom{p}{k}$ models by RSS → M_k. Pick among M_k by CV / adj. R^2 / AIC / BIC. Cost 2^p.

Model counts

Best subset: 2^p. Stepwise: $1 + \frac{p(p+1)}{2}$.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/regularisation/best-subset-and-stepwise-selection

Ridge Regression (L2)

Ridge adds a penalty on the sum of squared coefficients. Its closed-form solution, why it shrinks, why it always has a solution, how it handles correlated predictors, and two worked examples.

RIDGE OBJECTIVE AND SOLUTION

$$\min_{\beta} \ \mathrm{RSS}(\beta) + \lambda \sum_{j=1}^{p} \beta_j^2 \qquad \Longrightarrow \qquad \hat{\beta}_\lambda^{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y$$

The intercept β_0 is not penalised (center the data, or exclude it from the penalty).

▶ Show the derivation

Why it shrinks

Adding λI makes the "denominator" bigger. In the simplest case $X^\top X = I$ (orthonormal predictors):

$$\hat{\beta}_j^{\text{ridge}} = \frac{\hat{\beta}_j^{\text{OLS}}}{1 + \lambda}$$

Every coefficient is divided by something larger than 1. In general all coefficients move towards zero as λ grows, and $\hat{\beta} \to 0$ as $\lambda \to \infty$. They never become exactly zero.

EXAMPLE A: ONE PREDICTOR, BY HAND

Centred data $x = (-1, 0, 1)$, $y = (-2, 0, 2)$. $\sum xy = 4$, $\sum x^2 = 2$.

$$\hat{\beta}_1^{\text{ridge}} = \frac{\sum xy}{\sum x^2 + \lambda} = \frac{4}{2 + \lambda}$$

λ	0	2	6	∞
slope	2	1	0.5	0
training RSS	0	2	4.5	8 (= TSS)

As λ grows, the slope shrinks and training RSS rises.

EXAMPLE B: TWO CORRELATED PREDICTORS

$$X^\top X = \begin{pmatrix} 4 & 3.6 \\ 3.6 & 4 \end{pmatrix}, \quad X^\top y = \begin{pmatrix} 5 \\ 4.6 \end{pmatrix}$$

OLS: $\det = 16 - 12.96 = 3.04$, so $\hat{\beta} = \frac{1}{3.04}\begin{pmatrix} 4(5) - 3.6(4.6) \\ -3.6(5) + 4(4.6) \end{pmatrix} = \frac{1}{3.04}\begin{pmatrix} 3.44 \\ 0.4 \end{pmatrix} = (1.132,\ 0.132)$.

Ridge, $\lambda = 1$: $X^\top X + I = \begin{pmatrix} 5 & 3.6 \\ 3.6 & 5 \end{pmatrix}$, $\det = 25 - 12.96 = 12.04$, so $\hat{\beta} = \frac{1}{12.04}\begin{pmatrix} 25 - 16.56 \\ -18 + 23 \end{pmatrix} = (0.701,\ 0.415)$.

Ridge shrinks the large coefficient and shares credit with its correlated twin. The sum barely moves (1.26 vs 1.12), while the split is much more even and stable.

Properties

- **Always solvable:** $X^\top X + \lambda I$ is invertible for any $\lambda > 0$, even when $X^\top X$ is singular ($p > n$, perfect collinearity). Adding λ to every eigenvalue makes them all positive.
- **Bias–variance:** as λ increases, bias increases and variance decreases.
- **Correlated predictors:** shares the credit instead of producing huge opposite-sign coefficients.
- **Cons:** many small but non-zero coefficients (no variable selection), and no convenient significance test for coefficients.

Cheatsheet

Ridge

$\min\ \mathrm{RSS} + \lambda \sum \beta_j^2$; $\hat{\beta} = (X^\top X + \lambda I)^{-1} X^\top y$.
Orthonormal: $\hat{\beta}^{\mathrm{OLS}}/(1+\lambda)$.

Ridge properties

Shrinks, never zero. Always solvable. Shares credit among correlated predictors. Bias ↑, variance ↓ with λ.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/regularisation/ridge-regression

LASSO Regression (L1)

LASSO penalises the sum of absolute coefficients, which shrinks some exactly to zero: built-in variable selection. Soft thresholding, a worked comparison with ridge, and when to prefer each.

LASSO OBJECTIVE

$$\min_{\beta} \ \mathrm{RSS}(\beta) + \lambda \sum_{j=1}^{p} |\beta_j|$$

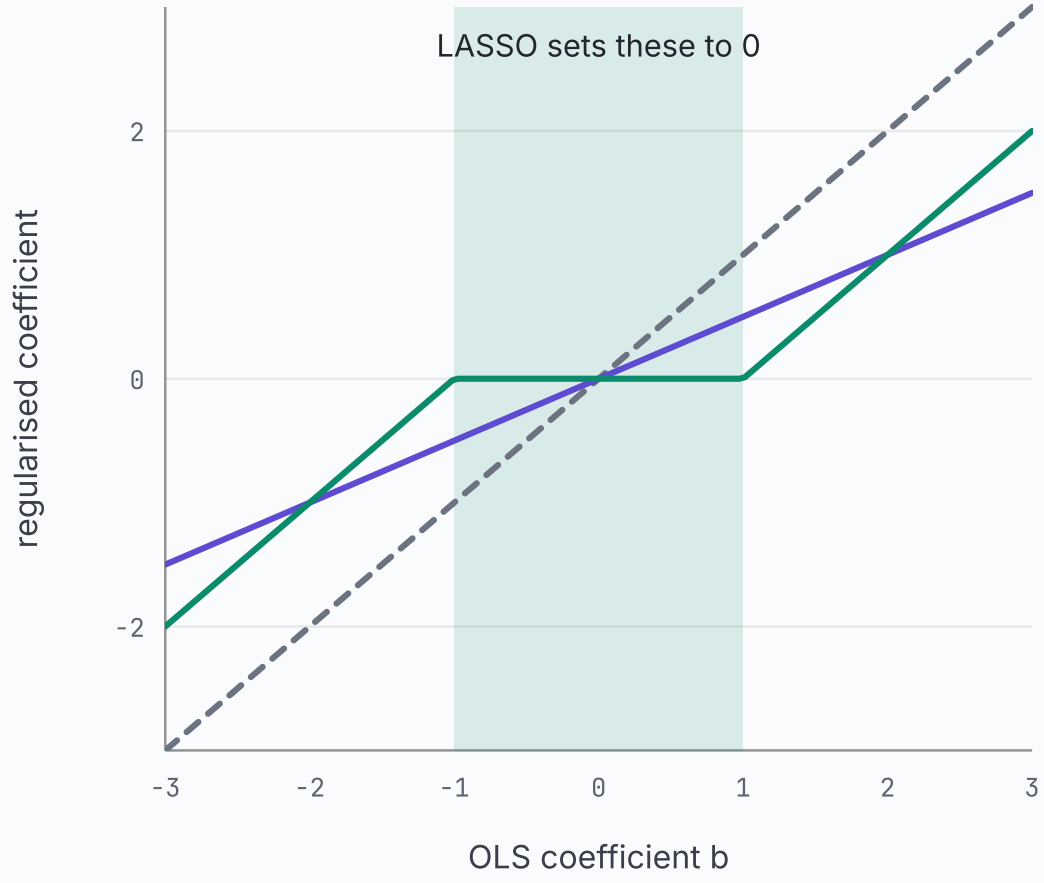
LASSO = Least Absolute Shrinkage and Selection Operator.

- Higher λ: smaller coefficients, higher bias, lower variance, just like ridge.
- **The key difference:** LASSO shrinks some coefficients to **exactly zero**, so it performs variable selection and gives sparse, interpretable models.
- No closed form: it's solved iteratively (e.g. coordinate descent). Harder than ridge, far easier than best subset.
- Like ridge, no convenient significance test for coefficients.

Soft thresholding

In the orthonormal case ($X^\top X = I$), with $b_j = \hat{\beta}_j^{\mathrm{OLS}}$ and the objective written as RSS $+\lambda\|\beta\|_1$:

$$\hat{\beta}_j^{\mathrm{ridge}} = \frac{b_j}{1+\lambda} \ \text{(proportional shrinkage)}, \qquad \hat{\beta}_j^{\mathrm{lasso}} = \mathrm{sign}(b_j)\ \max\Big(|b_j| - \tfrac{\lambda}{2},\ 0\Big) \ \text{(soft thresholding)}$$



Ridge scales every coefficient down. LASSO subtracts a constant and sets small coefficients exactly to zero.

WORKED EXAMPLE: RIDGE VS LASSO ON THE SAME OLS COEFFICIENTS

OLS: $b = (3, -1.5, 0.4)$, $\lambda = 2$, so the LASSO threshold is $\lambda/2 = 1$.

Ridge: $b/3 = (1.0, -0.5, 0.133)$. All non-zero.

LASSO: $(3-1,\ -(1.5-1),\ 0) = (2, -0.5, 0)$. The small coefficient becomes exactly 0 (variable dropped); the others shrink by a constant 1.

Ridge or LASSO?

SITUATION	FAVOUR
Many predictors, you believe only a few matter, you want interpretability	**LASSO** (sparse)
Many correlated predictors that all plausibly contribute a little	**Ridge** (shares credit, stable)
Groups of correlated predictors *and* you want selection	Elastic net (a mix of both)

LASSO WITH CORRELATED PREDICTORS

Among highly correlated predictors LASSO tends to pick one and drop the rest, somewhat arbitrarily; which one can change with a small change in the data. Don't read a dropped variable as "irrelevant", and never describe LASSO-selected variables as "the causes".

Cheatsheet

LASSO

$\min\ \mathrm{RSS} + \lambda \sum |\beta_j|$. Orthonormal: $\mathrm{sign}(b)\max(|b| - \lambda/2, 0)$ (soft threshold).

Ridge vs LASSO in one line

Ridge divides (never zero); LASSO subtracts (exact zeros → selection).

Online version with the interactive quiz: airesearchlabs.si/learn/ml/regularisation/lasso-regression

Understanding λ (Lambda)

What happens at λ = 0, small, intermediate, large and infinite λ. Why regularisation never beats OLS on training error, and how bias, variance and coefficients move along the way.

The regimes of λ

Λ	COEFFICIENTS	TRAINING ERROR	BIAS / VARIANCE	TYPICAL TEST ERROR
0	OLS	**Minimum possible**	Lowest bias, highest variance	High if overfitting
Small	Slightly shrunk	Rises a little	Variance falls	Often falls
Intermediate	Moderately shrunk (LASSO: some zero)	Higher	Balanced	**Best** (chosen by CV)
Large	Heavily shrunk	High	High bias	Rises again
$\to \infty$	All slopes → 0 (LASSO: exactly 0 beyond a finite λ)	= TSS (predicts $\bar{y}$)	Pure bias	Underfitting

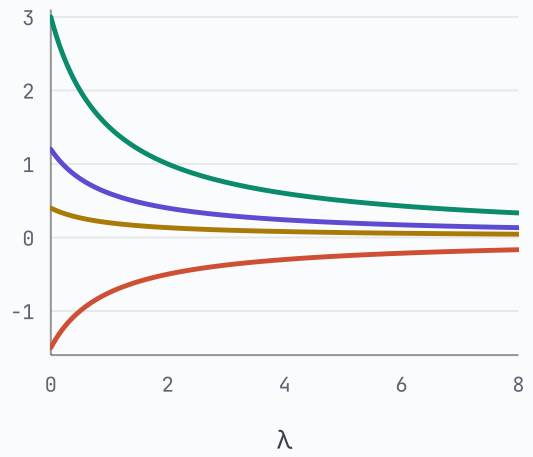
WHEN IS TRAINING ERROR LOWEST?

At $\lambda = 0$. With no penalty, ridge and LASSO both reduce to OLS, which by definition minimises RSS. Training error then increases (weakly) as λ grows. Regularisation buys better *test* error with extra bias; it never beats OLS on the training data.

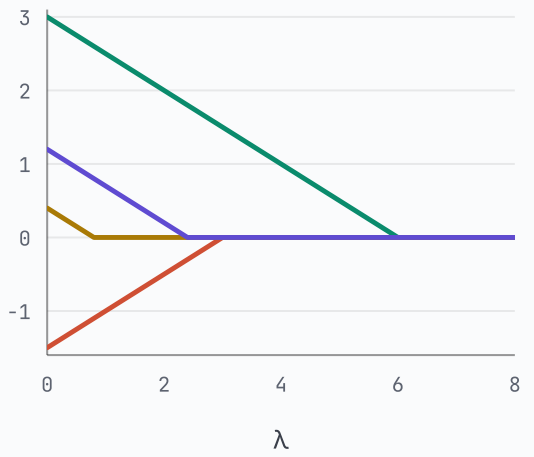
TRAINING LOSS VS PENALISED OBJECTIVE

For each λ, the regularised solution minimises *RSS + penalty*, so it beats OLS on that objective. But OLS beats it on plain RSS. If a question asks about "training error", it means RSS/MSE.

Coefficient paths



Ridge: every coefficient shrinks smoothly toward 0 but never reaches it.



LASSO: coefficients hit exactly 0 one after another (0.4 first, then 1.2, then −1.5) and stay there.

OLS coefficients 3 (teal), −1.5 (coral), 0.4 (amber) and 1.2 (violet), orthonormal design.

The order in which LASSO drops variables is a rough ranking of importance: the coefficients that survive longest matter most. In practice λ is plotted on a log scale.

INTERPRETING A CHANGE IN λ

Training MSE is 0.8 at $\lambda = 0$ and 1.1 at $\lambda = 5$; test MSE is 3.0 at $\lambda = 0$ and 1.9 at $\lambda = 5$. The unpenalised model overfits: the penalty raised training error slightly but cut test error substantially, by reducing variance.

Across many random training sets, regularised models give more **consistent** coefficients (less variable), at the price of bias. The net effect on prediction is best at an intermediate λ.

Cheatsheet

λ regimes

λ=0: OLS, min training error. Small: slight shrink, test often ↓. Middle: best (CV). Large: high bias. ∞: predicts $\bar{y}$.

Lowest training error?

Always at λ = 0. Regularisation never beats OLS on training RSS.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/regularisation/understanding-lambda

Ridge versus LASSO

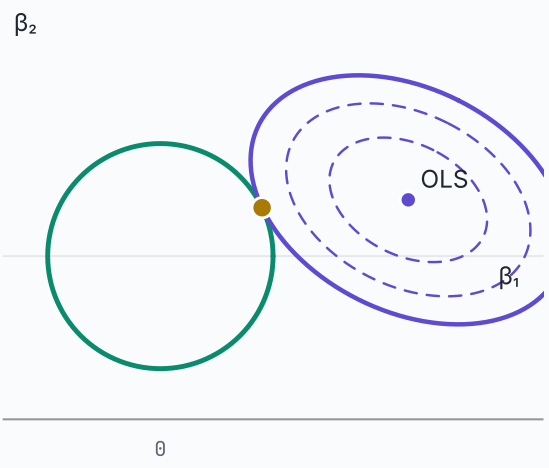
Why LASSO produces exact zeros and ridge does not, explained with the constraint geometry of a disc versus a diamond. A full side-by-side comparison with subset selection, plus an interactive shrinkage explorer.

The constraint view

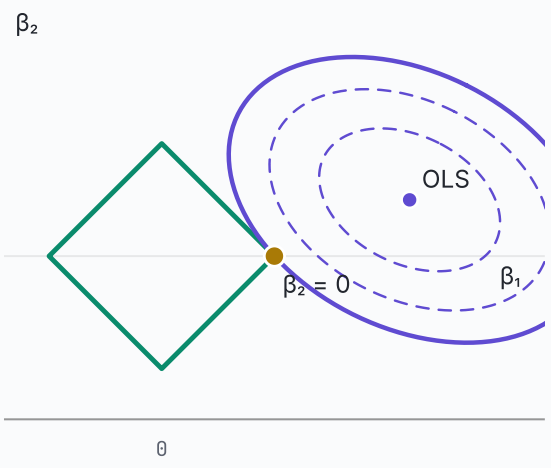
Each penalised problem is equivalent to minimising RSS **subject to a budget** on the coefficients:

$$\text{Ridge: } \min \text{RSS} \text{ s.t. } \sum \beta_j^2 \leq s \qquad \text{LASSO: } \min \text{RSS} \text{ s.t. } \sum |\beta_j| \leq s$$

RSS contours are ellipses centred on the OLS solution. The answer is where the expanding ellipse **first touches** the allowed region.



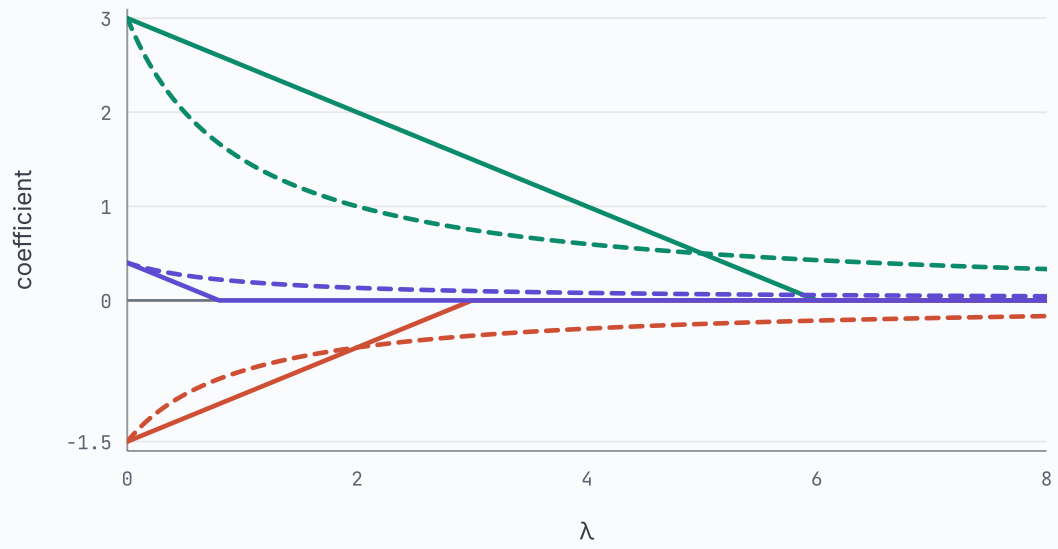
Ridge (disc): the ellipse touches a smooth boundary, so both coefficients are usually non-zero.



LASSO (diamond): the ellipse first touches a **corner** on an axis, where β_2 is exactly 0.

The diamond has corners sitting on the axes, and an ellipse coming from a typical direction very often hits a corner first. In high dimensions the L1 ball has many corners and edges, so many coefficients end up exactly zero. The ridge disc is round, with no corners to snag on.

RIDGE VS LASSO SHRINKAGE AS Λ GROWS



OLS coefficients b = (3, −1.5, 0.4) with an orthonormal design. Ridge divides every coefficient, so none reaches zero. LASSO subtracts, so 0.4 hits zero at λ = 0.8 and −1.5 at λ = 3.

Side by side

	SUBSET (L0)	RIDGE (L2)	LASSO (L1)
Penalty	Number of non-zeros	$\sum \beta_j^2$	$\sum \lvert\beta_j\rvert$
Exact zeros?	Yes	No	Yes
Closed form?	No (search over subsets)	Yes: $(X^\top X + \lambda I)^{-1} X^\top y$	No (iterative)
Computation	Very hard (2^p)	Easy	Moderate
$X^\top X$ singular / $p > n$	Subsets still small	Fine	Fine (selects at most n variables)
Correlated predictors	Picks one arbitrarily	Shares credit	Tends to pick one, drop others (unstable)
Interpretability	High	Medium (many small values)	High
Coefficient p-values	No convenient test	No convenient test	No convenient test

Elastic net mixes both penalties: a weight of 0 on the L1 part is ridge, 1 is LASSO, and values in between combine selection with ridge's stability for correlated groups.

"LASSO IS A MIDDLE GROUND BETWEEN BEST SUBSET AND RIDGE"

Like subset selection it produces sparse models (exact zeros); like ridge it is a continuous shrinkage problem that's cheap to solve. It gets most of subset selection's interpretability without the 2^p search.

Cheatsheet

Subset vs ridge vs LASSO

Zeros: yes / no / yes. Closed form: no / yes / no. Cost: 2^p / easy / moderate. Correlated: arbitrary / shares / picks one.

Why LASSO gives zeros

L1 constraint = diamond with corners on the axes; RSS ellipses usually touch a corner first.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/regularisation/ridge-versus-lasso

Mathematical Objectives and Worked Calculations

Norms, penalised objectives and how to compute ridge, LASSO and subset "errors" by hand, including choosing between two candidate fits, and penalised logistic regression.

Norms

For $\beta = (\beta_1, \ldots, \beta_p)$ (the intercept is excluded):

$$\|\beta\|_0 = \#\{j : \beta_j \neq 0\}, \qquad \|\beta\|_1 = \sum_j |\beta_j|, \qquad \|\beta\|_2^2 = \sum_j \beta_j^2$$

$\|\beta\|_1$ is the "Manhattan distance" from zero; $\|\beta\|_2^2$ the squared Euclidean distance.

EXAMPLE

$\beta = (3, -1, 0, 2)$: $\|\beta\|_0 = 3$, $\|\beta\|_1 = 3 + 1 + 0 + 2 = 6$, $\|\beta\|_2^2 = 9 + 1 + 0 + 4 = 14$.

The three objectives

$$\text{Ridge: RSS} + \lambda\|\beta\|_2^2 \qquad \text{LASSO: RSS} + \lambda\|\beta\|_1 \qquad \text{Subset: RSS} + \lambda\|\beta\|_0$$

COMPUTING THE "ERRORS" (OBJECTIVE VALUES)

A fit has RSS $= 12$ and $\hat{\beta} = (3, -2, 0.5, 0)$. Take $\lambda = 2$.

- Ridge: $12 + 2(9 + 4 + 0.25 + 0) = 12 + 26.5 = \mathbf{38.5}$
- LASSO: $12 + 2(3 + 2 + 0.5 + 0) = 12 + 11 = \mathbf{23}$
- Subset: $12 + 2 \times 3 = \mathbf{18}$ (three non-zero coefficients)

CHOOSING BETWEEN TWO CANDIDATE FITS

Fit A: RSS $= 10$, $\beta = (4, -3)$. Fit B: RSS $= 13$, $\beta = (2, -1)$.

OBJECTIVE	A	B	WINNER
Ridge, $\lambda = 1$	$10 + 25 = 35$	$13 + 5 = 18$	B
LASSO, $\lambda = 1$	$10 + 7 = 17$	$13 + 3 = 16$	B
Ridge, $\lambda = 0.1$	$10 + 2.5 = 12.5$	$13 + 0.5 = 13.5$	A

With a small penalty the better-fitting model wins; with a big penalty the smaller-coefficient model wins.

CONVENTIONS DIFFER

Some software divides RSS by $2n$ (scikit-learn's Lasso) or uses $\frac{1}{2}$RSS, which rescales what "λ" means. The story is identical (bigger penalty → smaller coefficients). If an exam question gives a formula, use that one.

Regularised logistic regression

The same idea applies to classification. Penalise the negative log-likelihood:

$$\min_{\beta} \{ -\ell(\beta) + \lambda \,\text{Penalty}(\beta)\}, \qquad \text{ridge: } \lambda \sum \beta_j^2, \quad \text{LASSO: } \lambda \sum |\beta_j|$$

- Ridge-logistic: smooth shrinkage, stabilises optimisation.
- LASSO-logistic: sparsity; effective when $p > n$ or features are correlated.
- It fixes perfect separation: coefficients can no longer diverge.
- λ is chosen by CV, e.g. minimising log-loss or maximising AUC.

PENALISED LOGISTIC OBJECTIVE BY HAND

$-\ell = 5.2$, $\beta = (1.5, -2)$, $\lambda = 0.5$. Ridge: $5.2 + 0.5(2.25 + 4) = 8.325$. LASSO: $5.2 + 0.5(1.5 + 2) = 6.95$.

Non-parametric models are regularised too: larger K in K-NN, shallower trees, and in SVMs a smaller cost C (6.7).

Cheatsheet

Norms

$\|\beta\|_0$ = #non-zeros; $\|\beta\|_1 = \sum |\beta_j|$; $\|\beta\|_2^2 = \sum \beta_j^2$ (intercept excluded).

Penalised objective by hand

1) Compute RSS (or $-\ell$). 2) Compute the norm of the slopes. 3) Add λ × norm. Compare candidates on the same λ.

Regularised logistic

$\min\{-\ell(\beta) + \lambda \sum \beta_j^2\}$ (ridge) or $+\lambda \sum |\beta_j|$ (LASSO).

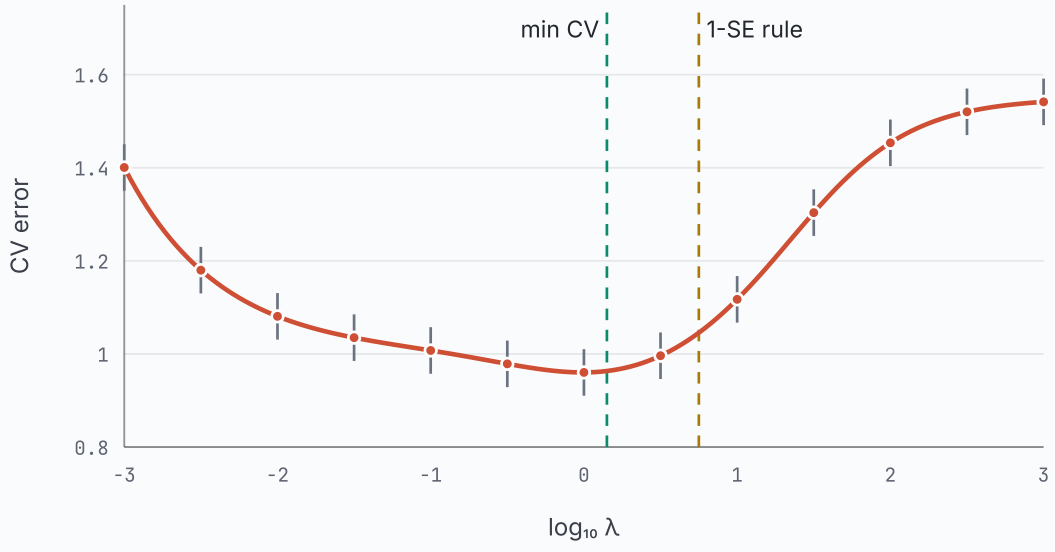
Online version with the interactive quiz: airesearchlabs.si/learn/ml/regularisation/mathematical-objectives-and-worked-calculations

Selecting λ Using Cross-validation

λ is a hyperparameter: choose it by cross-validation on standardised training data, then test once. The full procedure, the one-standard-error rule, a case study, and the mistakes to avoid.

The procedure

1. **Split** off a test set and leave it alone.
2. Choose a **grid of λ values**, log-spaced (e.g. 10^{-4} to 10^{2}).
3. For each λ, run **K-fold CV** on the training data. In each fold, **standardise using that fold's training part only**, fit, and record the validation error.
4. Pick the λ with the lowest mean CV error (or use the 1-SE rule below).
5. **Refit** on all training data with that λ.
6. **Evaluate once** on the test set.



Small λ (left) overfits; large λ (right) underfits. Bars show ± 1 standard error across folds. The 1-SE rule picks the largest λ within one SE of the minimum: a simpler model with nearly the same accuracy.

Why standardise first?

The penalty treats every coefficient the same, but a coefficient's size depends on its predictor's units. A variable measured in small units needs a big coefficient and gets penalised unfairly. Standardising (mean 0, SD 1) puts every predictor on the same footing. And to avoid leakage, the mean and SD must come from training folds only. The intercept is never penalised.

The one-standard-error rule

CV error is itself noisy. The **1-SE rule** chooses the largest λ whose CV error is within one standard error of the minimum. You get a simpler (more regularised, sparser for LASSO) model that is statistically indistinguishable in accuracy.

CASE STUDY: AN AIR-QUALITY SENSOR MODEL

Predicting CO from 11 standardised sensors ($n \approx 6{,}000$ training rows), λ by 5-fold CV:

	TEST R^2	TEST RMSE	NMHC SENSOR COEF.
OLS	0.885	0.478	−0.051
Ridge	0.885	0.477	−0.006
LASSO	0.885	0.477	0.000

Prediction: essentially identical. With 6,000 rows and only 11 predictors, OLS already has low variance; regularisation buys little accuracy.

Coefficients: the near-duplicate NMHC sensor (correlation 0.98 with benzene, VIF 61) is shrunk almost to zero by ridge and dropped by LASSO; the inflated benzene coefficient falls from 0.586 to 0.553. Coefficients become more stable and the model cheaper: one fewer sensor to buy.

Takeaway: regularisation's benefit isn't always accuracy. Stability, interpretability and cost can matter as much.

COMMON MISTAKES

- Selecting λ on the test set (then the test score is optimistic).
- Standardising on the whole dataset before CV (leakage).
- Forgetting to standardise at all.
- Calling LASSO-selected variables "the causes", or reporting p-values from an OLS refit on them as if no selection happened.

Cheatsheet

Choosing λ

Split off test → log-spaced λ grid → K-fold CV (standardise inside folds) → pick min CV or 1-SE → refit on all training → test once.

Regularisation & leakage

Standardise within training folds. Never choose λ on the test set.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/regularisation/selecting-lambda-using-cross-validation

Section 6

Clustering and Support Vector Machines

How K-means groups observations without labels, and how SVMs build the widest-margin decision boundary.

6.1 K-means Clustering · 8 min

6.2 Distance Calculations and Cluster Assignment · 8 min

6.3 Updating Centroids and Convergence · 9 min

6.4 SVM Fundamentals · 7 min

6.5 Margins and Support Vectors · 10 min

6.6 Hard-margin and Soft-margin SVM · 10 min

6.7 The C Parameter and Kernels · 10 min

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.

C1

SUPERVISED LEARNING · CLASSIFICATION LANE · STEP C1

Choose the model

How do I turn inputs into a class, or a probability of one?

Lessons here: 6.4, 6.5 · feeds into Turn predictions into decisions

C2

SUPERVISED LEARNING · CLASSIFICATION LANE · STEP C2

Fit it

Which coefficients make the observed labels most probable?

Lessons here: 6.6 · feeds into Control complexity

5

SUPERVISED LEARNING · STEP 5

Control complexity

How simple can the model be and still capture the pattern?

Lessons here: 6.7 · feeds into Turn predictions into decisions

1

UNSUPERVISED LEARNING · STEP 1

Measure similarity

What does it mean for two customers to be "close"?

Lessons here: 6.2

2

UNSUPERVISED LEARNING · STEP 2

Assign and update

How does K-means find the groups?

Lessons here: 6.1

3

UNSUPERVISED LEARNING · STEP 3

Choose K and judge the result

How many groups, and are they any use?

Lessons here: 6.3 · feeds into Control complexity, Measure similarity

K-means Clustering

K-means finds K groups in unlabelled data by alternating two simple steps. What clustering is for, the algorithm in four steps, what it minimises, and how it differs from K-nearest neighbours.

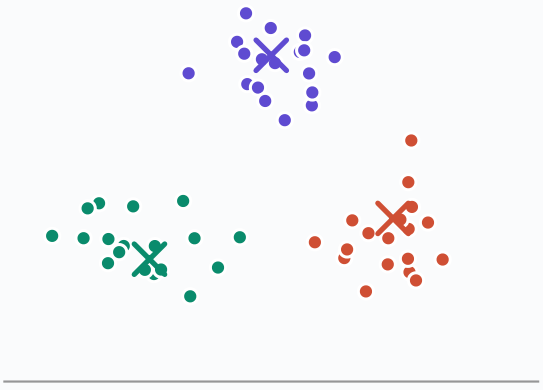
Clustering

So far every model had a label Y to predict. **Clustering** has no labels: it partitions observations into groups so that points in the same cluster are similar and points in different clusters are different. Typical uses: customer segmentation, grouping patients with similar profiles, compressing data into K prototypes, exploratory analysis.

Because there's no "right answer" column, there's no test-set accuracy. Quality is judged by internal measures (how tight the clusters are) and by usefulness.



Before: unlabelled points.



After K-means, K = 3: each point joins its nearest centroid (×).

The algorithm (Lloyd's algorithm)

Input: data $x_1, \dots, x_n \in \mathbb{R}^p$ and the number of clusters K. μ_k is the centroid (mean) of cluster k.

1. **Initialise:** pick K starting centroids, e.g. K random data points (or the smarter k-means++).
2. **Assignment step:** assign every point to its nearest centroid: $c_i = \arg\min_k \|x_i - \mu_k\|^2$.
3. **Update step:** move each centroid to the mean of its assigned points: $\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$.
4. **Repeat** steps 2–3 until the assignments stop changing.

TWO ALTERNATING GUESSES

Step 2 asks "given where the centres are, who belongs to whom?" Step 3 asks "given who belongs to whom, where is the best centre?" Each step can only improve the objective, so the process settles down.

What it minimises

WITHIN-CLUSTER SUM OF SQUARES (WCSS, INERTIA)

$$J = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2$$

The total squared distance from each point to its own cluster's centre.

It's a squared-error loss with no labels: each point's "target" is its own centroid. The algorithm finds a **local** minimum of J, not necessarily the global one (6.3).

K-means vs K-nearest neighbours

	K-MEANS	K-NEAREST NEIGHBOURS
Learning type	Unsupervised (no labels)	Supervised (labels needed)
Task	Clustering	Regression or classification
What K is	Number of clusters	Number of neighbours used to predict
Output	Cluster assignments + centroids	Predicted value or class

WRONG TOOL

"K-means is used to predict whether a patient will be readmitted." If labels exist and prediction is the goal, use supervised learning. Clusters can be useful *features*, but they aren't predictions.

Cheatsheet

K-means

1) Initialise K centroids. 2) Assign each point to the nearest centroid. 3) Move each centroid to its cluster mean. 4) Repeat until assignments stop changing.

K-means objective

$J = \sum_k \sum_{i \in C_k} \|x_i - \mu_k\|^2$ (WCSS). Local minimum only.

K-means vs K-NN

K-means: unsupervised, K = clusters. K-NN: supervised, K = neighbours.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/clustering-and-svm/k-means-clustering

Distance Calculations and Cluster Assignment

Euclidean and Manhattan distance, how to assign points to centroids by hand, a fully worked 2-D example, and why unscaled features let one variable dominate the clusters.

Measuring distance

DISTANCES BETWEEN A AND B IN P DIMENSIONS

$$\text{Euclidean: } \|a-b\| = \sqrt{\sum_{j=1}^{p}(a_j-b_j)^2} \qquad \text{Manhattan: } \sum_{j=1}^{p}|a_j-b_j|$$

K-means uses (squared) Euclidean distance. For assignment, comparing squared distances gives the same answer as comparing distances, and skips the square root.

QUICK CHECK

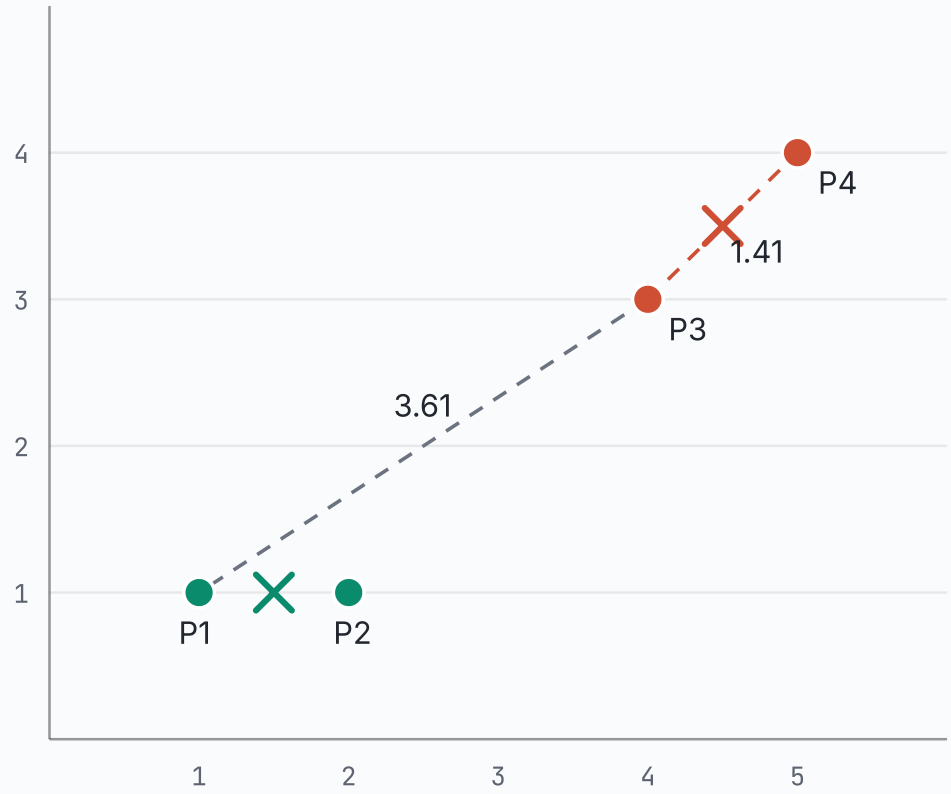
$a=(1,1)$, $b=(4,5)$: Euclidean $=\sqrt{9+16}=5$; squared $=25$; Manhattan $=3+4=7$.

Worked example: one assignment step

Points $P_1(1,1)$, $P_2(2,1)$, $P_3(4,3)$, $P_4(5,4)$; $K=2$; initial centroids $\mu_1=P_1=(1,1)$ and $\mu_2=P_4=(5,4)$.

POINT	DISTANCE TO $M_1(1,1)$	DISTANCE TO $M_2(5,4)$	ASSIGN
$P_1(1,1)$	0	$\sqrt{16+9}=5$	C_1
$P_2(2,1)$	$\sqrt{1+0}=1$	$\sqrt{9+9}=4.24$	C_1
$P_3(4,3)$	$\sqrt{9+4}=3.61$	$\sqrt{1+1}=1.41$	C_2
$P_4(5,4)$	5	0	C_2

So $C_1=\{P_1,P_2\}$ and $C_2=\{P_3,P_4\}$. The update step comes next (6.3).



P3 is 1.41 from μ_2 but 3.61 from μ_1, so it joins cluster 2. Crosses show where the centroids move after the update step.

Scale matters

Distance adds up squared differences across features, so a feature with a large numerical range dominates. Cluster customers on annual income (£20,000–£200,000) and store visits (1–40), and the clusters will split purely by income: a £1,000 income difference outweighs a 30-visit difference.

FIX

Standardise features (mean 0, SD 1) before K-means, unless the raw units are genuinely comparable. The same applies to K-NN and SVMs.

CATEGORIES

Means and Euclidean distance are undefined for categorical data ("red", "green"). Use variants such as k-modes or a suitable distance.

Cheatsheet

Distances

Euclidean $\sqrt{\sum(a_j - b_j)^2}$; Manhattan $\sum |a_j - b_j|$. K-means compares squared Euclidean.

Unscaled features

The largest-range feature dominates distances. Standardise before K-means, K-NN, SVM.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/clustering-and-svm/distance-calculations-and-cluster-assignment

Updating Centroids and Convergence

The update step, why K-means always converges but only to a local optimum, the effect of a bad start, choosing K with the elbow method, and the main ways K-means goes wrong.

The update step, continued

From 6.2: $C_1 = \{P_1(1,1), P_2(2,1)\}$, $C_2 = \{P_3(4,3), P_4(5,4)\}$. Move each centroid to its cluster mean:

$$\mu_1 = \left(\tfrac{1+2}{2}, \tfrac{1+1}{2}\right) = (1.5, 1), \qquad \mu_2 = \left(\tfrac{4+5}{2}, \tfrac{3+4}{2}\right) = (4.5, 3.5)$$

Iteration 2: reassign. Every point is still closest to the same centroid, so assignments don't change: **converged**.

$J = (0.25 + 0.25) + (0.5 + 0.5) = 1.5$.

Why it converges

- The assignment step can't increase J: each point moves to a closer (or equally close) centre.
- The update step can't increase J: the mean minimises the sum of squared distances within a cluster.
- There are finitely many ways to assign points, so J can't keep decreasing forever.

But it converges to a **local** minimum, which depends on the starting centroids.

WORKED EXAMPLE: A POOR START THAT RECOVERS

1-D points $1, 2, 4, 11, 12, 14$; $K = 2$; start at $\mu_1 = 1$, $\mu_2 = 2$.

1. Assign: 1 → C_1; 2, 4, 11, 12, 14 → C_2. Update: $\mu_1 = 1$, $\mu_2 = 43/5 = 8.6$.
2. Assign: 1, 2 (1 vs 6.6), 4 (3 vs 4.6) → C_1; 11, 12, 14 → C_2. Update: $\mu_1 = 7/3 = 2.33$, $\mu_2 = 37/3 = 12.33$.
3. Assign: no change. Converged. $J = (1.78 + 0.11 + 2.78) + (1.78 + 0.11 + 2.78) = 9.33$.

Here a bad start still found the natural groups. On other data it can get stuck in a poor solution, so K-means is run **several times from different starts** and the run with the lowest J is kept. k-means++ chooses spread-out starting centroids to make bad starts rarer.

Choosing K in K-means

J always decreases as K grows (with $K = n$, every point is its own cluster and $J = 0$), so you can't choose K by minimising J, just as training error can't choose model complexity.



Elbow method: big drops up to K = 3, then only small gains. Choose the elbow.

- **Elbow method:** plot J against K and choose the bend where extra clusters stop helping much.
- **Silhouette score:** for each point, $s = \frac{b-a}{\max(a,b)} \in [-1, 1]$, where a is the mean distance to its own cluster and b to the nearest other cluster. Higher average is better.
- **Usefulness and stability:** are the clusters actionable, and do they stay similar on resampled data?

When K-means goes wrong

ISSUE	WHY	WHAT TO DO
Features on different scales	Large-scale variable dominates distance	Standardise first
Sensitive to initialisation	Only a local optimum	Many restarts; k-means++; keep lowest J
K must be chosen	Not learned	Elbow, silhouette, domain use
Assumes round, similar-sized clusters	Euclidean distance to a mean	Other methods (mixtures, density-based) for elongated or nested clusters
Outliers	Means are pulled by extremes	Handle outliers; consider k-medians
Hard assignments only	Each point fully in one cluster	Soft clustering (mixture models)

Cheatsheet

Choosing K

Not by min J (falls with K). Use the elbow, silhouette, stability and usefulness.

K-means failure modes

Unscaled features · bad starts (local optima) · wrong K · non-round clusters · outliers · categorical data.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/clustering-and-svm/updating-centroids-and-convergence

SVM Fundamentals

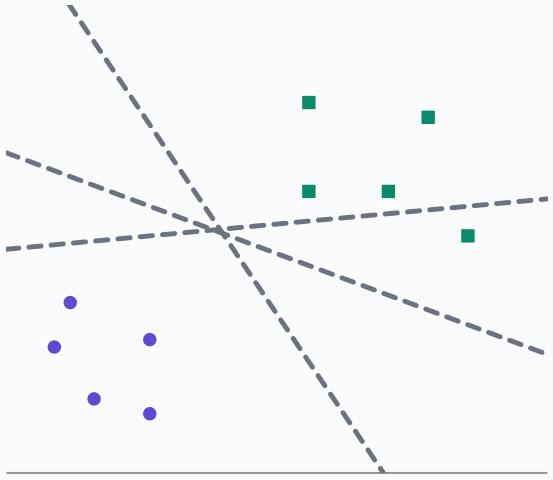
A support vector machine is a linear classifier that chooses the separating line with the widest safety margin. The idea, the ±1 label convention, the decision rule, and the link to regularisation.

The idea: the widest street

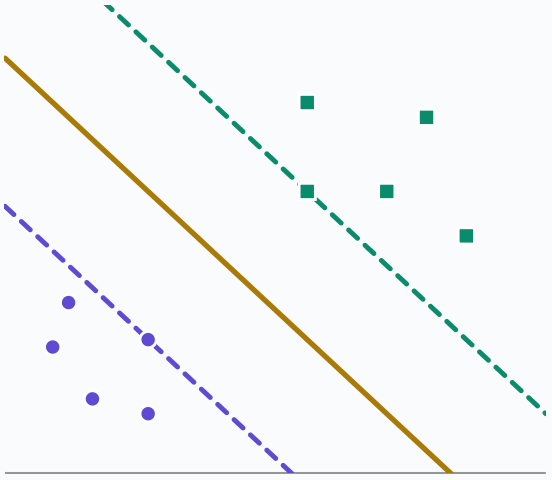
For two classes labelled $y \in \{-1, +1\}$ (note: not 0/1), a linear classifier computes a score and takes its sign:

$$f(x) = w^\top x + b, \qquad \hat{y} = \operatorname{sign}\big(f(x)\big)$$

If the classes can be separated by a line, infinitely many lines work. Which is best?



Many lines separate the training data, but some pass very close to points.



The SVM picks the line in the middle of the **widest empty street** between the classes.

WHY A WIDE MARGIN?

A line squeezed against the training points could be tipped over by a little noise in new data. A line in the middle of a wide gap tolerates more. A wider margin means a simpler, more robust classifier: this is the link to regularisation.

How SVMs fit into the course

	LOGISTIC REGRESSION	LINEAR SVM
Labels	$\{0, 1\}$	$\{-1, +1\}$
Loss	Cross-entropy	Hinge loss (6.6)
Output	A probability $P(Y = 1 \mid x)$	A score / class only (probabilities need extra calibration)
Which points matter	All points	Only support vectors: points on or inside the margin
Regularisation	λ (ridge or LASSO)	C (acts like $1/\lambda$), L2 on w
Nonlinear boundaries	Add features by hand	Kernels (6.7)
Needs standardised features	For penalties: yes	Yes

Both are linear classifiers with a boundary $w^\top x + b = 0$. They differ in which line they choose, because they use different losses.

Cheatsheet

Logistic regression vs linear SVM

Labels 0/1 vs ±1 · cross-entropy vs hinge · probability vs score · all points vs support vectors · λ vs C · features vs kernels.

SVM decision rule

$f(x) = w^\top x + b$; predict $\text{sign}(f(x))$. Labels $y \in \{-1, +1\}$.

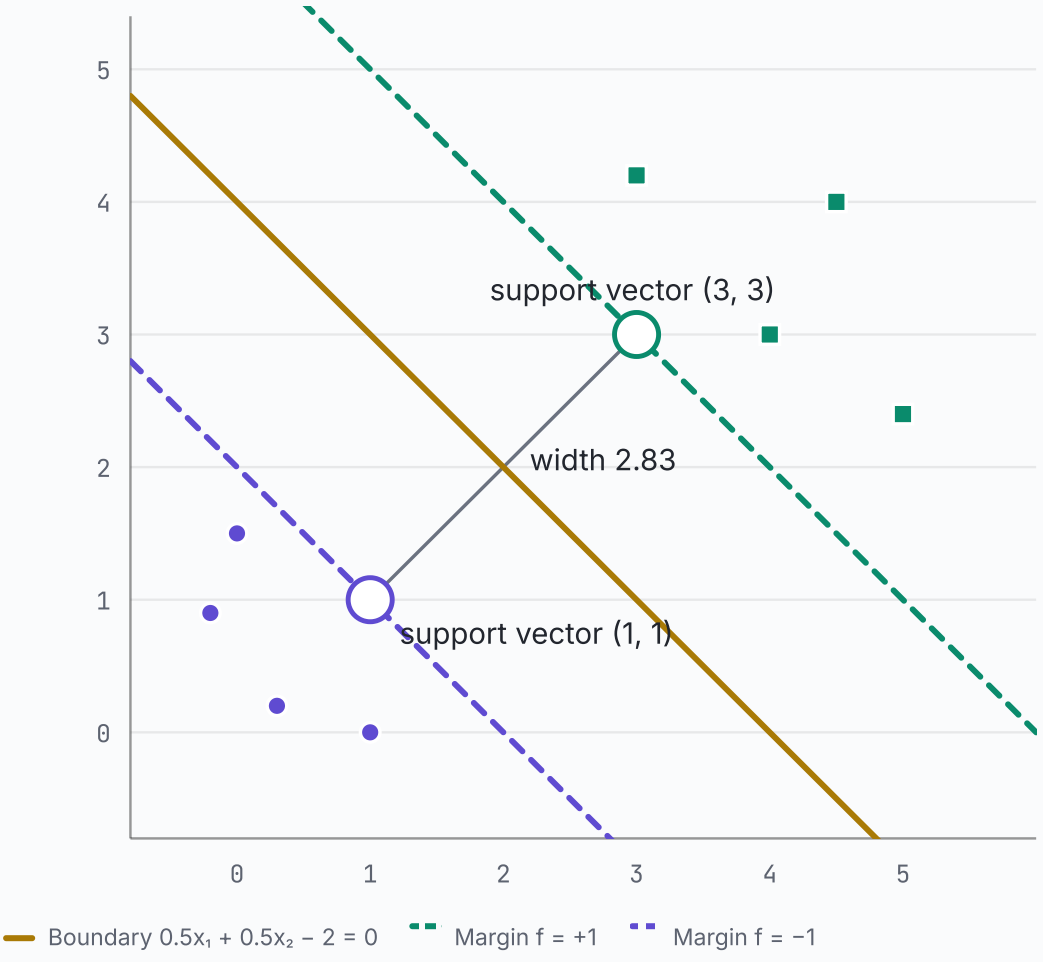
Online version with the interactive quiz: airesearchlabs.si/learn/ml/clustering-and-svm/svm-fundamentals

Margins and Support Vectors

The geometry of the margin: distance from a point to the boundary, the margin width 2/‖w‖, what support vectors are, and a complete SVM solved by hand.

Geometry

- The **decision boundary** is the hyperplane $w^\top x + b = 0$.
- The two **margin lines** are $w^\top x + b = +1$ and $w^\top x + b = -1$.
- The distance from a point x to the boundary is $\frac{|w^\top x + b|}{\|w\|}$.
- So the **margin width** (gap between the two margin lines) is $\frac{2}{\|w\|}$. A wider margin means a **smaller** $\|w\|$.



Circled points sit exactly on the margin lines: they are the support vectors. Moving any other point (without crossing the margin) leaves the solution unchanged.

Support vectors

Support vectors are the training points lying exactly on the margin lines (and, in the soft-margin case, inside the margin or misclassified). They alone determine the solution: delete or move any other point, as long as it stays outside the margin, and the boundary doesn't change. That's why SVMs can be efficient and robust to points far from the boundary.

AN SVM BY HAND

Negative support vector $(1, 1)$, positive support vector $(3, 3)$, other points further away. By symmetry, $w = (a, a)$. Support vectors satisfy $f = \pm 1$:

$$f(1,1) = 2a + b = -1, \qquad f(3,3) = 6a + b = +1$$

Subtract: $4a = 2$, so $a = 0.5$ and $b = -2$. Hence $w = (0.5, 0.5)$, $b = -2$.

$\|w\| = \sqrt{0.25 + 0.25} = 0.707$, so margin width $= 2/0.707 = 2.83$. Check: it equals the distance between the two support vectors, $\sqrt{4 + 4} = 2.83$.

Classify $(2, 3)$: $f = 0.5(2) + 0.5(3) - 2 = 0.5 > 0$, so $+1$. Its distance to the boundary is $0.5/0.707 = 0.71$. Because $|f| < 1$, it lies *inside* the margin.

MARGIN FROM A GIVEN W, B

$w = (3, 4)$, $b = -5$: $\|w\| = 5$, margin width $= 2/5 = 0.4$. Point $(2, 1)$: $f = 6 + 4 - 5 = 5 > 0$, class $+1$, distance $= 5/5 = 1$, outside the margin since $|f| \geq 1$.

Cheatsheet

SVM geometry

Boundary $w^\top x + b = 0$; margins $= \pm 1$. Distance $\frac{|w^\top x + b|}{\|w\|}$. Width $\frac{2}{\|w\|}$.

Support vectors

Points on (or inside) the margin. They alone determine the boundary.

SVM by hand (2 support vectors)

Use symmetry for the direction of w; set $f = -1$ at the negative SV and $f = +1$ at the positive SV; solve; width $= 2/\|w\|$.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/clustering-and-svm/margins-and-support-vectors

Hard-margin and Soft-margin SVM

The hard-margin optimisation problem, why it fails on overlapping data, slack variables and hinge loss, the soft-margin objective, and how hinge loss compares with logistic and 0–1 loss.

Hard margin

HARD-MARGIN SVM (PERFECTLY SEPARABLE DATA)

$$\min_{w,b} \ \tfrac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_i(w^\top x_i + b) \geq 1 \ \text{ for all } i$$

The constraint says every point is on the correct side and outside the margin. Minimising $\|w\|^2$ maximises the width $2/\|w\|$ (squaring and halving just make the maths easier).

The problem: real data are rarely separable. One overlapping point makes the constraints impossible, and a single outlier can force a very narrow margin.

Soft margin and hinge loss

Allow violations, but charge for them. For each point define the **hinge loss** (or slack):

$$\xi_i = \max\left(0,\ 1 - y_i f(x_i)\right)$$

$Y_I F(X_I)$	WHERE THE POINT IS	Ξ_I
≥ 1	Correct side, outside the margin	0
between 0 and 1	Correct side, inside the margin	between 0 and 1
< 0	Misclassified	> 1

SOFT-MARGIN SVM

$$\min_{w,b} \ \underbrace{\tfrac{1}{2}\|w\|^2}_{\text{wide margin (simplicity)}} + C \ \underbrace{\sum_{i=1}^{n} \xi_i}_{\text{hinge loss on training data}}$$

Equivalently: $\min \frac{1}{n}\sum_i \max(0, 1 - y_i f(x_i)) + \lambda\|w\|^2$ with $\lambda \propto 1/C$. That's hinge loss plus an L2 penalty, the ridge idea applied to classification.

HINGE LOSS AND THE OBJECTIVE BY HAND

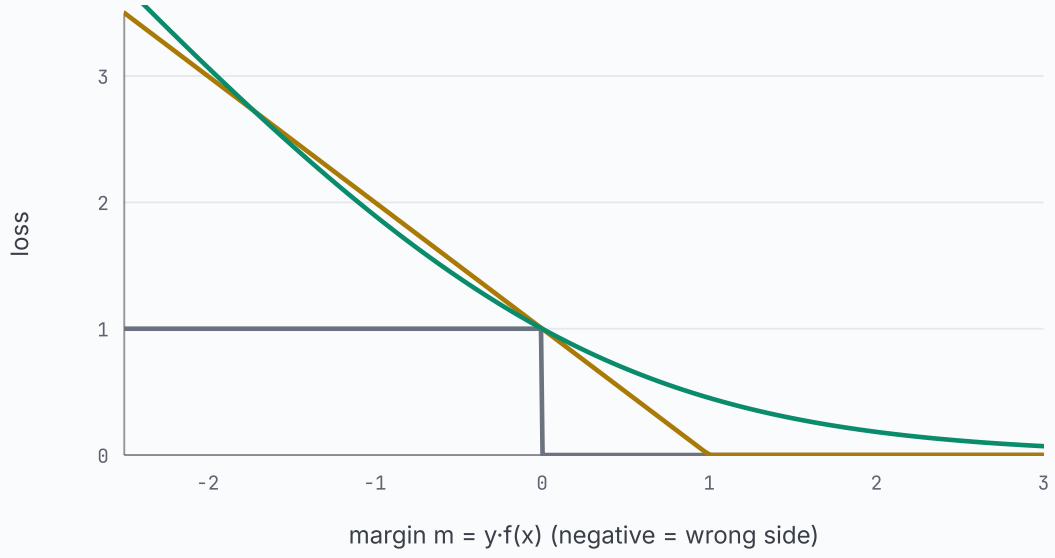
$y = +1$: $f = 2 \to 0$; $f = 0.3 \to 0.7$; $f = -0.5 \to 1.5$. $y = -1$, $f = -2$: $\max(0, 1 - 2) = 0$.

Suppose $w = (1, -1)$, $b = 0$ and four points have $\xi = (0, 0, 1, 0.5)$. Then $\frac{1}{2}\|w\|^2 = \frac{1}{2}(1 + 1) = 1$ and $\sum \xi = 1.5$.

$C = 1$: objective $= 1 + 1.5 = 2.5$. $C = 2$: objective $= 1 + 3 = 4$. A larger C makes violations count more, pushing toward a narrower margin with fewer violations.

Hinge vs the other classification losses

Write each loss in terms of the margin $m = yf(x)$ (positive = correct):



Hinge loss is exactly zero once a point is beyond the margin ($m \geq 1$); logistic loss is smooth and never exactly zero. Both are convex stand-ins for the 0–1 loss, which is hard to optimise.

Because hinge loss is zero for points comfortably on the right side, only margin violators (the support vectors) influence the SVM. Logistic regression is influenced by every point.

Cheatsheet

Hard-margin SVM

$\min \frac{1}{2}\|w\|^2$ s.t. $y_i(w^\top x_i + b) \geq 1$.

Soft-margin SVM

$\min \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$, $\xi_i = \max(0, 1 - y_i f(x_i))$. ≡ hinge + L2, $\lambda \propto 1/C$.

Hinge loss

$\max(0, 1 - yf(x))$. 0 beyond the margin; 0–1 inside; >1 if misclassified.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/clustering-and-svm/hard-margin-and-soft-margin-svm

The C Parameter and Kernels

C trades margin width against training violations, exactly like an inverse regularisation strength. Kernels let an SVM draw curved boundaries by working in a richer feature space. Linear, polynomial and RBF kernels, and tuning C and γ.

The role of C

	LARGE C (SMALL λ)	SMALL C (LARGE λ)
Margin	Narrow	Wide
Training violations	Few tolerated	Many tolerated
Bias / variance	Low bias, high variance (overfitting risk)	High bias, low variance (underfitting risk)
Support vectors	Fewer	More
Analogue	$\lambda \to 0$ in ridge/LASSO	Large λ

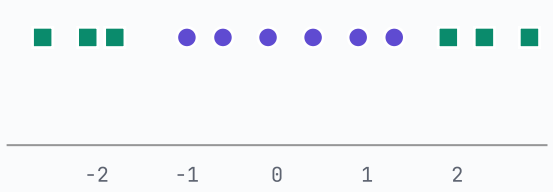
C is a hyperparameter: choose it by cross-validation, on standardised features.

DIRECTION CHECK

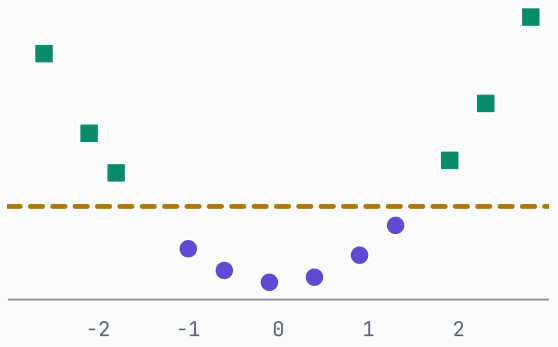
C multiplies the *training error*, while λ multiplies the *penalty*. So more C means less regularisation. It's easy to get backwards.

Kernels: curved boundaries

If no straight line separates the classes, map x into a richer feature space $\phi(x)$ where a hyperplane does.



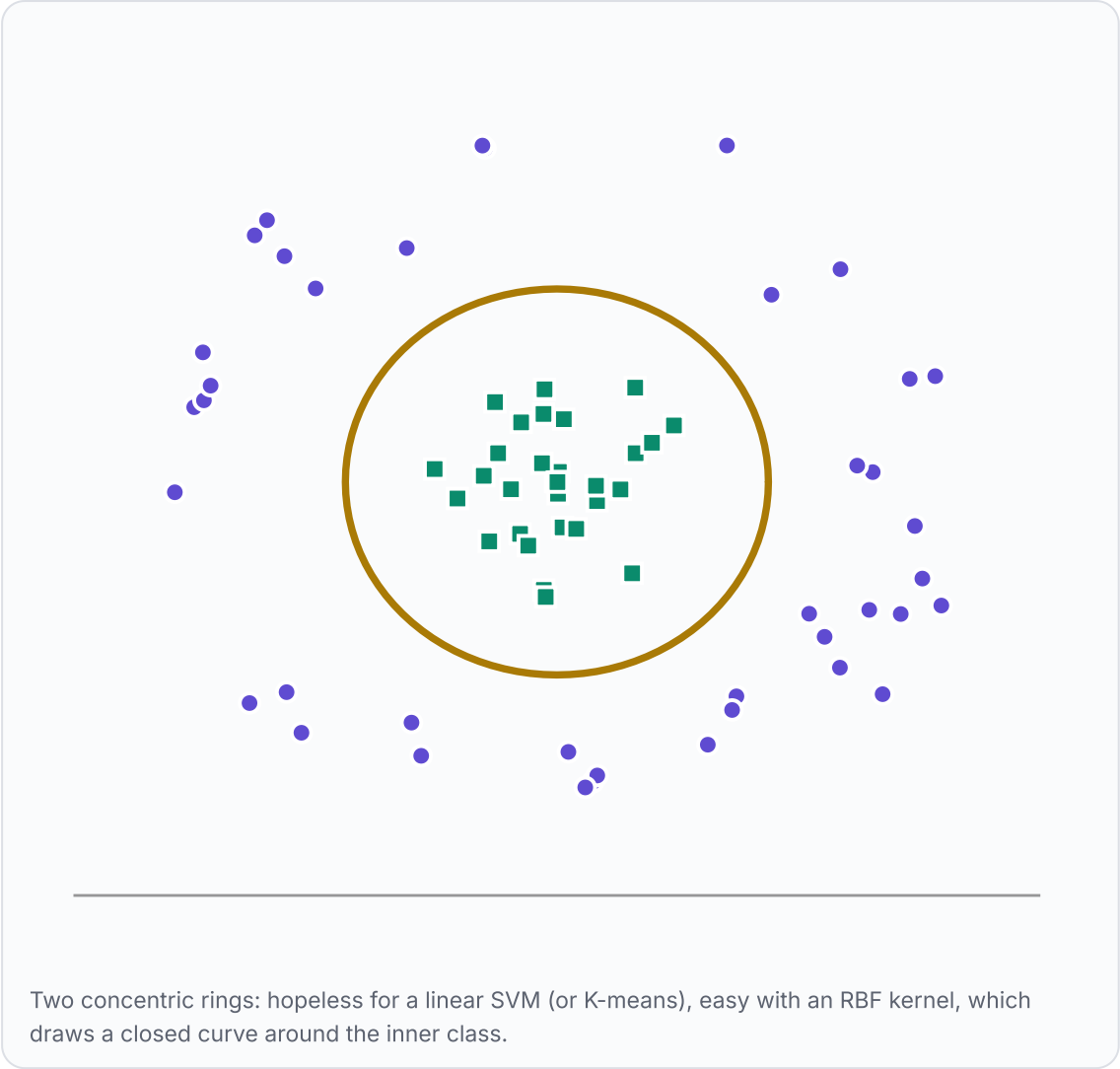
In 1-D, the outer class surrounds the inner one: no single cut-point separates them.



Map x → (x, x²). Now a straight line (x² = 2.25) separates them perfectly.

The **kernel trick** makes this cheap. The SVM's solution only ever needs inner products between points, so we can use a kernel function $K(x, x') = \phi(x)^\top \phi(x')$ that computes them directly, without ever building $\phi(x)$, which can even be infinite-dimensional.

KERNEL	$K(X, X')$	BOUNDARY
Linear	$x^\top x'$	Straight line / hyperplane
Polynomial	$(x^\top x' + c)^d$	Polynomial curves of degree d
RBF (Gaussian)	$\exp(-\gamma \lVert x - x' \rVert^2)$	Flexible, local "bumps"



Two concentric rings: hopeless for a linear SVM (or K-means), easy with an RBF kernel, which draws a closed curve around the inner class.

Tuning γ in the RBF kernel

- **Large γ:** each training point's influence is very local, so the boundary is wiggly: low bias, high variance (overfitting).
- **Small γ:** influence spreads far, so the boundary is smooth: high bias (underfitting).
- Tune C and γ **together** by cross-validation (e.g. a grid over both).

Practical notes

- Standardise features: distances and inner products are scale-sensitive.

- For class imbalance, use class weights (a bigger C for the rare class).
- Multi-class problems use one-vs-rest or one-vs-one combinations of binary SVMs.
- Training cost grows quickly with n, and probabilities need an extra calibration step.

Cheatsheet

Effect of C

Large C: narrow margin, few violations, low bias/high variance. Small C: wide margin, high bias/low variance. C ~ 1/λ.

Kernels

Linear $x^\top x'$ · Polynomial $(x^\top x' + c)^d$ · RBF $e^{-\gamma\|x-x'\|^2}$.

Complexity knobs, all models

More complex: ↑ degree, ↑ #features, ↓ λ, ↓ K (K-NN), ↑ C, ↑ γ (RBF), ↑ tree depth.

Online version with the interactive quiz: airesearchlabs.si/learn/ml/clustering-and-svm/c-parameter-and-kernels